Signature Based Malicious Behavior Detection in Android

Vikas Sihag[1,2](✉), Ashawani Swami[1], Manu Vardhan[2], and Pradeep Singh[2]

[1] Sardar Patel University of Police, Security and Criminal Justice, Jodhpur, India
{vikas.sihag,spu16cs04}@policeuniversity.ac.in
[2] National Institute of Technology, Raipur, Raipur, India
{mvardhan.cs,psingh.cs}@nitrr.ac.in

Abstract. User's security and privacy are of increasing concern with the popularity of Android and its applications. Apps of malicious nature attempts to perform activities like information leakage and user profiling, detection of which is a challenge for security researchers. In this paper, we try to solve this problem by proposing a behavior based approach to detect malicious nature of applications in Android. Events and behavioral activities of an application are used to generate signature, which then is matched with signature database for detection. Behavioral signatures are designed on the basis of information leakage attempt, jailbreak attempt, abuse of root privilege and access of critical permissions. 260 popular apps of different nature were evaluated in addition to 42 android apps, which were flagged malicious by Government of India. The proposed system shows promising results to detect malicious behaviors. It also defines the nature of malicious activity exploited by an app.

Keywords: Malware · Android · Security · Dynamic analysis · Information leakage

1 Introduction

Android, an open source OS provides us with an interactive platform for running multiple applications. Android OS since its release in 2008, has grown as the most prefered choice in the market with over 2.5 billion active devices worldwide and 74.13% share in December 2019 [16]. Smartphones today being equipped with sophisticated sensors, from camera and microphones to gyroscopes and proximity sensors, creates a new paradigm for user experience. Sensors generates data, which contains sensitive information thus opening new attack surfaces to be exploited by developers with malicious intent. When a malware (malicious app) is installed on an android device, the user is open to serious privacy and security issues such as information leakage and over privilege permission exploitation. Benign and alternative malicious applications are released on the android marketplaces [19,25,27]. These apps are also pushed into the market without third-party reviews [1,5,22]. Google Play Store, Android's official application hosting service has over 2.57 million apps generating about 140 million USD [17].

Spotted in 2010, the first Android Malware was DroidSMS, which would subscribe users to premium SMS services. Since then multiple genres of malware have targeted Android ranging from downloaders to clickers, spyware to banking trojans and adware to ransomware [3,6,12,18,23]. Recently CamScanner a popular document scanning application on Google Play store was identified to be infected with `AndroidOS.Necro.n` dropper, which once installed attempts to install another malware [21]. Recently, 983 cases of known vulnerabilities and 655 zero-days were found among the top 5,000 free apps (each with 1M to 500M downloads) available on Google Play Store.

Applications running on Android performs system interacts at different levels. These system and application interactions can be recorded in the form of logs. Logs are generally classified into either system generated logs or application generated logs. System generated logs record events taking place in the execution of the operating system in order to provide a trail that can be used to understand the activities of the system and to identify problems. Application generated logs are events logged by a running application on the OS. It logs, events, warnings and errors. Information logged in it is determined by the app developer not the OS. Malicious app developers avoid detection by evading logging of its behavior [26]. Application logs are thus less reliable than system logs for behavior detection. Logcat is an operating system debug tool designed for Android. It monitors the system in real-time and creates a dump of system and application log messages, thus making it a suitable choice to collect information from Android OS.

In this paper we try to detect application behavior using log dumps from logcat. The approach is based on system generated logs rather than application logs. As system logs will contain activities and events of all running applications, a filtering mechanism is then employed to constrain on monitoring the selected application only. Behavior based signatures are generated on the basis of malicious activities for purpose of information leakage, jailbreak, abuse of root privilege or access of critical permissions.

Paper Organization: Following Sect. 2 discusses the prerequisites. Section 3 overviews the proposed solution. Section 4 discusses dataset preparation, experimental setup, results and analysis. Related work is covered in Sect. 5. Concluding remarks and future scope are discussed in Sect. 6.

2 Background Information

Android OS has a linux based stacked architecture, with linux kernel closest to the hardware and interact with the device hardware. Stacked representation of Android architecture as illustrated in Fig. 1. Linux kernel is customized for smart devices with power, memory and computational constraints. It includes all the key hardware drivers for camera, keyboard, wireless, audio, screen etc. It is responsible for power management, process management and memory management. Above kernel, HAL provides standard interface to the Java API framework that exposes device hardware capabilities. Each process in Android version 5.0

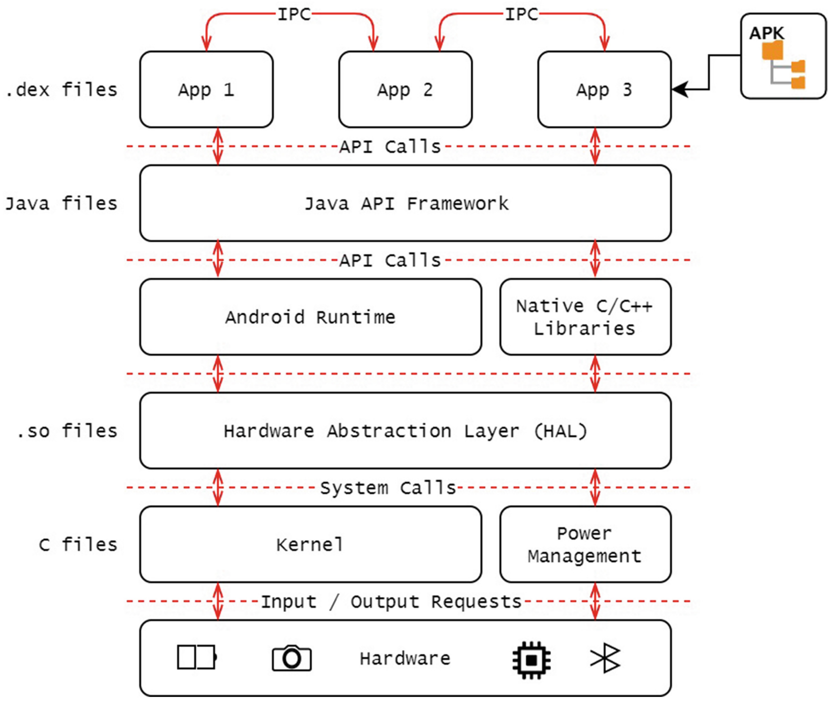

Fig. 1. Android architecture diagram

and above runs with its own Android Runtime (ART), which converts application dex bytecode into native code to be executed. Prior Android versions comprised of Dalvik Virtual Machine, a kind of Java VM optimized for low processing and memory constrained environment. The app works with its own copy of the virtual Dalvik VM. Dalvik for security was designed to efficiently operates multiple virtual machines. Dalvik executable (.dex) file of an application is optimized and run in its Dalvik VM. Most key components and utilities of Android system such as ART and HAL are designed using native codes with C and C++ compatible libraries. Above ART and Native Libraries resides Java API framework. Features of Android are accessed by apps using it to help developers write apps easily and quickly. It includes APIs to design UI, work with databases, handle user interaction, etc. On top of the stack are applications for user interactions. Android contains a set of preinstalled system apps to ensure minimum functionalities of SMS, internet browsing, contact management, calendar, music and more. These system apps provide vital capabilities that developers can access from their app, for instance sharing a message by system messaging app [11,20].

2.1 Android Security Mechanisms

Android due to its popularity, is a popular target of malware authors. Android OS installed with advanced hardwares and softwares generates enormous amount of valuable data for and about users. Cybercriminals are continuously crafting applications to maliciously take advantage of users' valuable data. Various security mechanisms employed by Android for application security are discussed below.

Android Permissions Framework. Permissions are key components and plays a vital role in the identification of malware. Permissions are intended to safeguard the user's privacy. Apps must request permission to user information such as text messages, contacts and certain system elements like the microphone, the camera and network access. Depending on the function, Android could automatically give the authorization or the scheme could encourage the customer to agree or reject the application. Permissions are categorized into Normal, Dangerous, Signature and SignatureOrSystem level permissions. *Normal* permissions are lower-risk permissions for apps requiring access to other apps. *Dangerous* permissions are those which accesses user data or vital device functionalities. Dangerous permissions are used by applications to gain device access and user privacy. *Signature* permission are allotted automatically during app installation only if app being installed has the same signature to an existing installed app with the permission. *SignatreOrSystem* earlier known as Signature|Privileged, is granted by system only to applications that are in a specific location or has the same signature as that of an app declaring it. It is designed for multiple vendors sharing specific features or folders on the system.

Android Sandboxing. Application sandboxing feature creates a contained environment for process to be executed. Each running app is limited to its sandbox, which is quarantined from other apps. An app may get at resources outside its sandbox, only the ones which were permitted by users during installation. Discretionary Access Control (DAC) is used for resource management outside the sandbox. If an app X tries to read application Y's resources, it is prevented because of the lack of user privileges. As apps are digitally signed with the developer's private key, apps with same developer's certificate are assigned same UID for resource and permission sharing. Thus malware authors with developer's key can design an app with the same certificate to access private resources of other apps developed by the developer.

Inter-component Communication (ICC). Android employs ICC mechanism or Binder for apps to communicate with each other or system. It is responsible of request migration from the originating process to the target process. Using ICC an application component can request data access from another component within the same application or other application within the same device or a remote service. For example, a product delivery application may use Map application for device's location by ICC.

2.2 Android Security Threats

Android smartphones being hub of users' private data is always under the lense of app developers. Users now have high probability of facing information leakage such as mobile number, email address, IMSI (International Mobile Subscriber Identifier), IMEI (International Mobile Equipment Identifier), Contact list, and other personal identification information. IMEI or IMSI numbers do not immediately expose user identity, but with malicious service or app having access to it there is always a risk of privacy violation [7,13]. Android security being based on permission model, lacks fine permission control and management.

Apart from threat on user's private information, there is always a threat on exploiting the underlying Android system. Attackers may exploit a vulnerability in the underlying linux sytem to gain root privilege of the device. Attackers with root privileged can easily over powers Android security framework. These attacks can result in unauthorized actions from malicious applications, which thus will cause security and privacy violations.

3 Proposed Solution

Our aim is to design and implement a system, that can collect and select real-time app interactions specific for a target application and check the behavioral activity for malicious nature.

The proposed solution offers a dynamic analysis approach to analyse application behavior. Figure 2 gives overview of the proposed architecture containing an Android system and a server. System generated logs are collected from the Android device at kernel-level. Thus any attempt by malicious application to evade application level logging would be detected. Collected logs are filtered and then matched with signatures for analysis.

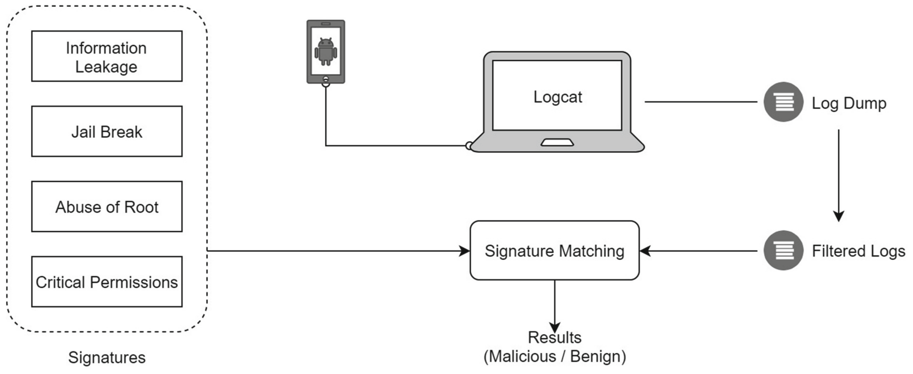

Fig. 2. Work flow of the proposed solution

Logging. Android OS and app activities of different types can be logged using the Android system debugger called logcat. It logs both system and application generated logs as depicted in Fig. 3. A log entry from logcat includes tag and its priority. A tag indicates the component responsible for its generation (for example, OpenGLRenderer). The priority defines the priority level amongst Assert (A), Error (E), Warning (W) and Verbose (V) from high to low.

A log entry format of logcat contains:

```
<date> <time> <PID>-<TID>/<package> <priority>/tag: <message>
```

PID stands for process identifier and TID is thread identifier. Below is a sample logcat entry.

```
14-11 17:31:21.320 74-113/com.example.application I/Application:
IN CLASS: (ENAppn.java:27)
```

Application generated logs are events logged by apps running on the OS. Information logged in application logs is determined by the app developer not the OS. Malicious app developers can easily avoid detection by not logging its activities. Application logs thus cannot be relied upon for behavior detection.

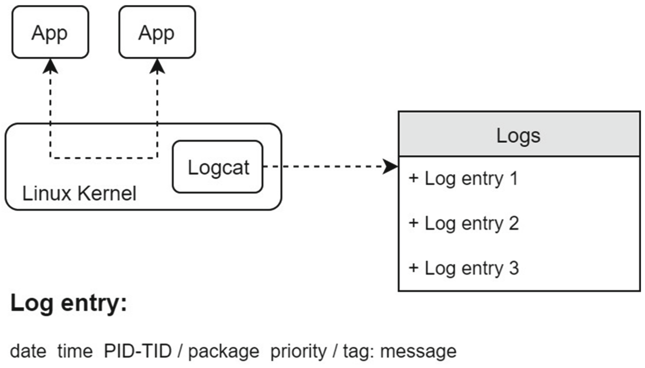

Fig. 3. Log collection

Log Filtering. The quantity of logs prevents manual analysis of kernel-level logs. A logging module at the kernel level gather records of log information from all events which take place on the operating system. The collected information consists of entries irrelevant w.r.t. our target activity. Furthermore, given the restricted storage capability on a smartphone the quantity of record information needs to be reduced. We only gather logs associated to interested system calls for malware detection in order to address these issues. Linux OS has around 300 system calls, which can be categorized based on their functionality. Thus system calls responsible for critical activities (such as process, memory and device management) are considered for further analysis. This enables us to acquire log data containing entries of concerned system calls. But it includes system calls related to all processes running on the OS. We therefore further remove entries related to irrelevant process by selecting only the concerned process entries. Using logcat tool process ID of concerned app and child processes are selected for filtering. Critical permissions accessed by application during runtime are captured in logs (Fig. 4).

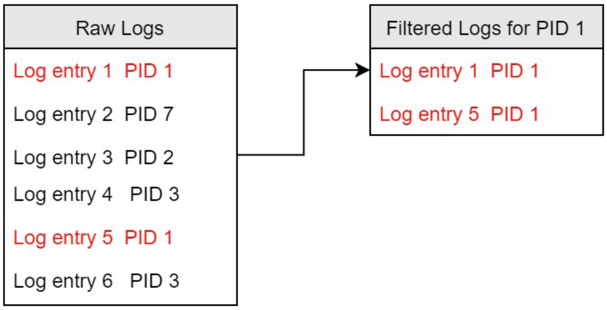

Fig. 4. Log filtering for selected process with PID 1

Signature Matching. The proposed system is based upon pattern matching of signatures with the filtered logs from the above step. Signatures are carefully crafted for high detection accuracy. They if designed to be too specific will lead to high false negative, and if too considerate of various activities will lead to high false positive. Keywords responsible personal information such as mail account, IMEI, phone number, etc. were considered for user's personal identifiable information. For detecting jailbreak application and applications exploiting root privilege, name of known exploit binary files and commands respectively were selected for signatures. And for detecting requested critical permissions at runtime, critical permission request command were listed as signatures. List of signature is given in Table 1.

4 Results and Analysis

A system prototype of the proposed solution was implemented. System logs for the targeted application were generated and filtered, followed by pattern matching with the signatures. Popular applications of different categories were collected.

In total 260 applications were collected from Google Play Store and other Third party app stores for analysis. Selection of applications was done on the basis of popularity. Popular applications from each category (social networking, games, productivity, messaging, etc.) were selected. Apart from the above,

Table 1. Signatures for detecting malicious behaviour

Threat	**Signatures and keywords**
Information leakage	IMSI and SIM
	IMEI and Android
	CLONE_FILES
	TEL and MAIL
	quattrowirelesssdk
	admob
	adwhirl
	flurryagent
Jail break	asroot
	exploid
	rageagainstthecage
	gingerbreak
Root abuse	ˆexecve(”/(system/)?(bin|sbin|xbin)/su”
	ˆopen(”.*/iptables”
	ˆexecve(”.*/(chmod|chown|ls\b|mkdir|rmdir)”
	busybox
	daemonsu
	kingroot
	kinguser
	supersu
	superuser
	adfree
	greenify
	kerneladiutor
	setcpu
	shootme
	stericson
	titanium
Critical permissions	SEND_SMS_NO_CONFIRMATION
	WRITE_SMS
	READ_LOGS
	INSTALL_PACKAGES
	MOUNT_UNMOUNT_FILESYSTEMS
	READ_PHONE_STATE
	READ_HISTORY_BOOKMARKS
	WRITE_SMS
	READ_SMS
	READ_CONTACTS

42 applications listed malicious by Government of India were also selected for analysis. Analysis results are shown in Table 2 and 3.

Table 2. Analysis results of popular applications

Threat	Number of detection
Information leakage	91
JailBreak	10
Abuse root	15
Critical permissions	277

Table 3. Analysis results of 42 Apps listed malicious by GOI.

Threat	Number of detection
Information leakage	43
JailBreak	12
Abuse root	16
Critical permissions	63

Of 260 applications, 43 applications which taking sensitive data from device and 2 application fetching email data. 21 applications were also detected showing advertisement services. Additionally we have found 10 applications which try to jailbreak the device, 4 applications which is try to root the device, 4 applications run *rageagainthecage* which is used to root the device. Additionally, we found out 16 applications which use superuser activity in the device.

5 Related Work

Takamasa et al. in [8] describe the system architecture for signature pattern based detection of malicious applications for Android. Ma et al. in [13] introduces a novel attack vector called as "shadow attacks", which tries to evade the behavior based detection by partitioning a process into multiple shadow processes. Deguang Kong et al. in [10] describes a system which automatically assess the review to behaviour reliability of mobile applications. According to author, Autoreb uses artificial intelligence to identify linkage between user reviews and privacy compromises. They collected reviews from multiple sources thus to identify issues from reviews towards the application. Burguera et al. in [4] describes the system and framework to perform behavioral analysis of application to detect malware. Crowdroid collected information traces about application from multiple real-time users. It employs crowdsourcing approach to differentiate normal and malicious behavior. It also illustrated similarity between newly generated test malware and real samples. [4] considered system calls as they give low level information. They were able to detect all malicious execution for their self developed malicious apps. Andrea et al. in [15] presents the behaviour based detection system which analyses and correlates function. The functions selected were responsible for package-level, user-level, application-level and kernel-level interactions. A framework based on machine learning is introduced in [11] by Kong and Jin to address permission prediction problem. It captures the relations amongst textual descriptions, permissions, and app category. They tested the approach on 11k applications ranging from 30 categories. [14] and [2] classified malicious and benign applications based on system calls captured using strace tool. They then employ machine learning for classification. Wu et al. in [24] used call frequency and dependency for detection. They then fed LASSO, RF and SVM based machine learning classifiers to achieve accuracy of 93%. System logs and system calls are employed by Jang et al. in [9] for malware detection and familial classification. They reported accuracy of 99% for their approach.

6 Conclusion

We have presented a behavior based android malware detection approach. System generated logs are collected and filtered at runtime for application under analysis to be further matched with generated signatures. Signatures were generated taking into account the application behavior responsible for information leakage, jail break attempt, priviledge escalation attempt and access of critical permissions at runtime. Behavior based detection approaches provides insight into running applications as compared to static analysis approaches and thus need to be employed by market places for detecting behavior deversion of applications from intended behavior. Our approach was found to be effective as it gives considerable insight into malware interactions responsible for security and privacy compromises.

In future, a multi level hybrid approach including static analysis can be envisioned to improve scalability and efficiency of the detection system.

References

1. Bhandari, S., Panihar, R., Naval, S., Laxmi, V., Zemmari, A., Gaur, M.S.: Sword: semantic aware Android malware detector. J. Inf. Secur. Appl. **42**, 46–56 (2018). https://doi.org/10.1016/j.jisa.2018.07.003
2. Bhatia, T., Kaushal, R.: Malware detection in Android based on dynamic analysis. In: 2017 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), pp. 1–6. IEEE (2017). https://doi.org/10.1109/CyberSecPODS.2017.8074847
3. Bose, A., Hu, X., Shin, K.G., Park, T.: Behavioral detection of malware on mobile handsets. In: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, pp. 225–238 (2008). https://doi.org/10.1145/1378600.1378626

4. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for Android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26 (2011). https://doi.org/10.1145/2046614.2046619
5. Cai, H., Meng, N., Ryder, B., Yao, D.: DroidCat: effective Android malware detection and categorization via app-level profiling. IEEE Trans. Inf. Foren. Secur. **14**(6), 1455–1470 (2018). https://doi.org/10.1109/TIFS.2018.2879302
6. Dash, S.K., et al.: DroidScribe: classifying Android malware based on runtime behavior. In: 2016 IEEE Security and Privacy Workshops (SPW), pp. 252–261. IEEE (2016). https://doi.org/10.1109/SPW.2016.25
7. Doulamis, A., Pelekis, N., Theodoridis, Y.: Easytracker: an Android application for capturing mobility behavior. In: 16th Panhellenic Conference on Informatics, pp. 357–362. IEEE (2012). https://doi.org/10.1109/PCi.2012.22
8. Isohara, T., Takemori, K., Kubota, A.: Kernel-based behavior analysis for Android malware detection. In: 2011 Seventh International Conference on Computational Intelligence and Security, pp. 1011–1015. IEEE (2011). https://doi.org/10.1109/CIS.2011.226
9. Jang, J.W., Yun, J., Woo, J., Kim, H.K.: Andro-profiler: anti-malware system based on behavior profiling of mobile malware. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 737–738 (2014). https://doi.org/10.13089/JKIISC.2014.24.1.145
10. Kong, D., Cen, L., Jin, H.: Autoreb: automatically understanding the review-to-behavior fidelity in Android applications. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 530–541 (2015). https://doi.org/10.1145/2810103.2813689
11. Kong, D., Jin, H.: Towards permission request prediction on mobile apps via structure feature learning. In: Proceedings of the 2015 SIAM International Conference on Data Mining, pp. 604–612. SIAM (2015). https://doi.org/10.1137/1.9781611974010.68
12. Lindorfer, M., Neugschwandtner, M., Platzer, C.: Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 2, pp. 422–433. IEEE (2015). https://doi.org/10.1109/COMPSAC.2015.103
13. Ma, W., Duan, P., Liu, S., Gu, G., Liu, J.C.: Shadow attacks: automatically evading system-call-behavior based malware detection. J. Comput. Virol. **8**(1–2), 1–13 (2012). https://doi.org/10.1007/s11416-011-0157-5
14. Mas' ud, M.Z., Sahib, S., Abdollah, M.F., Selamat, S.R., Yusof, R.: Analysis of features selection and machine learning classifier in Android malware detection. In: 2014 International Conference on Information Science & Applications (ICISA), pp. 1–5. IEEE (2014). https://doi.org/10.1109/ICISA.2014.6847364
15. Saracino, A., Sgandurra, D., Dini, G., Martinelli, F.: Madam: effective and efficient behavior-based Android malware detection and prevention. IEEE Trans. Dependable Secur. Comput. **15**(1), 83–97 (2016). https://doi.org/10.1109/TDSC.2016.2536605
16. Statista: Mobile Operating Systems' Market Share Worldwide from January 2012 to December 2019. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/
17. Statista: Number of Apps Available in Leading App Stores as of 4th Quarter 2019. https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

18. Sun, M., Lui, J.C.S., Zhou, Y.: Blender: self-randomizing address space layout for Android apps. In: Monrose, F., Dacier, M., Blanc, G., Garcia-Alfaro, J. (eds.) RAID 2016. LNCS, vol. 9854, pp. 457–480. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45719-2_21
19. Sun, M., Wei, T., Lui, J.C.: Taintart: a practical multi-level information-flow tracking system for Android runtime. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 331–342 (2016). https://doi.org/10.1145/2976749.2978343
20. Sun, M., Zheng, M., Lui, J.C., Jiang, X.: Design and implementation of an Android host-based intrusion prevention system. In: Proceedings of the 30th Annual Computer Security Applications Conference, pp. 226–235 (2014). https://doi.org/10.1145/2664243.2664245
21. Team, K.: Malicious Android app had more than 100 million downloads in google play. https://www.kaspersky.com/blog/camscanner-malicious-android-app/28156
22. Wang, W., et al.: Constructing features for detecting Android malicious applications: issues, taxonomy and directions. IEEE Access **7**, 67602–67631 (2019). https://doi.org/10.1109/ACCESS.2019.2918139
23. Wang, X., Sun, K., Wang, Y., Jing, J.: DeepDroid: dynamically enforcing enterprise policy on Android devices. In: NDSS (2015). https://doi.org/10.14722/ndss.2015.23263
24. Wu, W.C., Hung, S.H.: DroidDolphin: a dynamic Android malware detection framework using big data and machine learning. In: Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, pp. 247–252 (2014). https://doi.org/10.1145/2663761.2664223
25. Xu, K., Li, Y., Deng, R.H.: ICCDetector: ICC-based malware detection on Android. IEEE Trans. Inf. Foren. Secur. **11**(6), 1252–1264 (2016). https://doi.org/10.1109/TIFS.2016.2523912
26. Zhang, H., Luo, S., Zhang, Y., Pan, L.: An efficient Android malware detection system based on method-level behavioral semantic analysis. IEEE Access **7**, 69246–69256 (2019). https://doi.org/10.1109/ACCESS.2019.2919796
27. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 1105–1116 (2014). https://doi.org/10.1145/2660267.2660359