Hindawi
Security and Communication Networks
Volume 2018, Article ID 9856537, 8 pages
https://doi.org/10.1155/2018/9856537

Research Article

A Security Sandbox Approach of Android Based on Hook Mechanism

Xin Jiang,[1] Mingzhe Liu ,[1] Kun Yang,[1] Yanhua Liu,[1] and Ruili Wang[2]

[1]*State Key Laboratory of Geohazard Prevention and Geoenvironment Protection, Chengdu University of Technology, Sichuan, China*
[2]*Institute of Natural and Mathematical Sciences, Massey University, Auckland, New Zealand*

Correspondence should be addressed to Mingzhe Liu; liumz@cdut.edu.cn

Received 13 May 2018; Accepted 28 June 2018; Published 19 July 2018

Academic Editor: Xuyun Zhang

As the most widely applied mobile operating system for smartphones, Android is challenged by fast growing security problems, which are caused by malicious applications. Behaviors of malicious applications have become more and more inconspicuous, which largely increase the difficulties of security detection. This paper provides a new security sandbox approach of Android based on hook mechanism, to further enrich Android malware detection technologies. This new sandbox monitors the behaviors of target application by using a process hook-based dynamic tracking method during its running period. Compared to existing techniques, (1) this approach can create a virtual space where apk can be installed, run, and uninstalled, and it is isolated from the outside and (2) a risk assessment approach based on behavior analysis is given so that users can obtain an explicit risk prognosis for an application to improve their safety. Tests on malware and normal application samples verify this new security sandbox.

1. Introduction

The security of Android is highly valued by and of wide concern to the industry [1] since a huge number of mobile applications are developed and used based on Android system. Various security issues of Android apps are continually being discovered and discussed, ranging from sensitive data leakage [2–4] to privilege escalation [5–7]. The proliferation of malicious apps on Android devices and the theft of user privacy data have become major problems for the development of the Android biosphere [8].

Previous methods have normally used Android internal sandbox to protect Android system in the public network [9, 10]. However, the internal sandbox hardly analyzes the risk behaviors of Android applications. Most of the prior studies for risk behaviors of Android applications rely on static analysis and dynamic analysis. Research on static analysis of Android application behavior, including MDroid [11] and MHealth Apps [12] static analysis tools has achieved good results in the application of behavior. On the other hand, static analyses may have their limitations, for instance, detecting sensitive data leakage [13–16] and analyzing capability leakage [5, 17]. In addition, the effectiveness of static analysis is restricted due to the distinctive features of Android's programming paradigm. Android apps are based on the Android OS, which can be regarded as a giant set of libraries containing both Java code and native code (so far, Android has consisted of more than 13 million lines of code [18]). Reference [19] proposed a verifiable diversity ranking search scheme over encrypted outsourced data while preserving privacy in cloud computing, which also supports Android data.

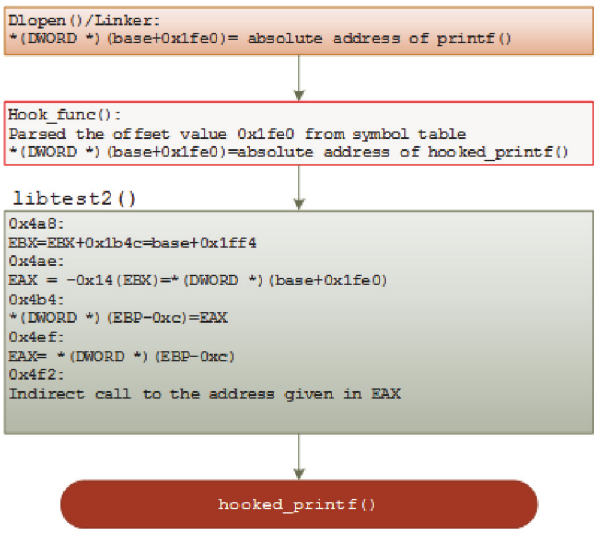
Contrary to static analysis, dynamic analysis executes selected program paths and thus can precisely identify property violations [20, 21]. Nonetheless, Android apps are tightly coupled with the Android OS which consists of a set of libraries containing both Java and native code and complex interprocess communications. To address this, Qi [22] proposed the taint propagation information tracking system based on TaintDroid, by modifying the Android system source code. Reference [23] implemented a tool named DroidInjector, which reproduced the malicious behavior of the application through the simulation of the application behavior. Gharib [24] put forward a system named ProfileDroid, which can depict the application behavior from 2 aspects of semantics and behavior and improve the accuracy of behavior analysis. Ardeshiricham [25] then designed a more detailed system named DroidScope. Reference [26] proposed an Activity call graph analysis method to generate UI interaction script automatically for Android applications. It can not only depict the application behavior semantically but also analyze the behavior characteristics of the application components. Lin has designed and implemented a tool named sandbox [27]. It can analyze sensitive actions from 2 aspects of general behavior and internal behavior. Reference [28] presented a scheme named SecDisplay for trusted display service; it protects sensitive data displayed from being stolen or tampered surreptitiously by a compromised OS. To detect DDoS attack, Cheng [29] proposed an abnormal network flow feature sequence prediction approach which could fit to be used as a DDoS attack detector in the big data environment and solve the aforementioned problems. And that method can also be used for detecting risk behaviors of Android app. Liu [30] designed a more practical privacy protection data aggregation protocol based on a new trusted model to avoid attack from DDoS.

In general, dynamic analysis on the application behavior can accurately judge the malicious behavior; however it cannot guarantee that all the codes for the implementation of application coverage, so that there might be a high missing rate. Static analysis can ensure the code coverage, but the accuracy of static analysis is easily affected by code obfuscation technology. For this reason, the paper begins with the application of Android risk behavior perspective, dynamic analysis based on the idea of the design and implementation with hook mechanism. It can automatically install, start, and uninstall the application and can simulate user actions in the application after the start, at the same time to monitor and record the behavior of the application. On this way, we put forward an evaluation method based on risk behavior. A security risk for the application of behavior is analyzed, so that users about risk related applications can have a clear anticipation.

2. Preliminary

2.1. Information Entropy. Information entropy, also known as Shannon entropy, is proposed by Shannon to solve the problem of quantitative measurement of information. In information theory, entropy is used to measure the expected value of a random variable. It represents the amount of information lost in the process of information transmission before being accepted and is also the entropy of information. In thermodynamics, the definition of entropy is the logarithm of the number of possible states of a system; its physical meaning is a measure of the degree of disorder in the system. Entropy is the measure of uncertainty of random variables in information theory. Information entropy considers that the magnitude of information of a message is directly related to its uncertainty. Named after Boltzmann's H-theorem, Shannon denoted the entropy H of a discrete random variable X with possible values $x_1, \ldots, x_n$ as

$$H(X) = E(I(X)) \tag{1}$$



```
Dlopen()/Linker:
*(DWORD *) (base+0x1fe0)= absolute address of printf()

Hook_func():
Parsed the offset value 0x1fe0 from symbol table
*(DWORD *) (base+0x1fe0)=absolute address of hooked_printf()

libtest2()
0x4a8:
EBX=EBX+0x1b4c=base+0x1ff4
0x4ae:
EAX = -0x14(EBX)=*(DWORD *) (base+0x1fe0)
0x4b4:
*(DWORD *) (EBP-0xc)=EAX
0x4ef:
EAX= *(DWORD *) (EBP-0xc)
0x4f2:
Indirect call to the address given in EAX

hooked_printf()
```

FIGURE 1: Example of how the hook function intercepts the call to printf() and reroutes the call to hooked_printf().

Here E is the expected value, and I is the information content of X. $I(X)$ is itself a random variable. If p denotes the probability mass function of X then the entropy can explicitly be written as

$$\begin{aligned} P(X) &= \sum_{i=1}^{n} p(x_i) I(x_i) = \sum_{i=1}^{n} p(x_i) \log_b \frac{1}{p(x_i)} \\ &= -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \end{aligned} \tag{2}$$

2.2. A Hook Example. As printf function, for example, if the hook module wants to intercept the calls to printf() and redirect to another function, it should write the redirected function address to the offset addresses of the symbol printf defined in the relocation sections, after the linker loaded the dynamic library into memory.

To replace the call of the printf() function with the call of the redirected hooked_printf() function, as shown in the software flow diagram in Figure 1, a hook function should be implemented between the dlopen() and libtest() calls. The hook function will first get the offset address of symbol printf, which is 0x1fe0 from the relocation section named .rel.dyn. The hook function then writes the absolute address of hooked_printf() function to the offset address. After that, when the code in libtest2() calls into the printf(), it will enter the hooked_printf() instead.

3. Methods

The goal of sandbox is to avoid both apps and Android system code modifications. The design of sandbox is to directly modify the apps virtual-memory tampering with ART internals representation of Java classes and methods. ART-sandbox consists of two components. The first component is the core engine written in C and the other one is the Java side that is

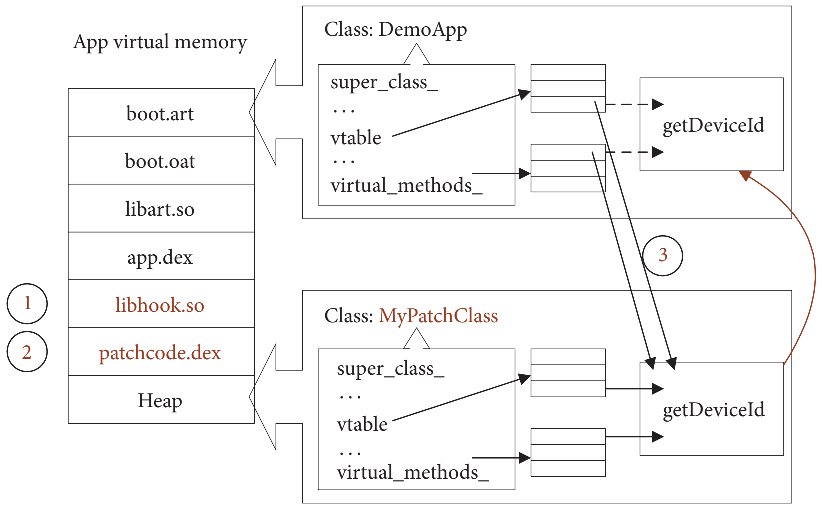


FIGURE 2: App virtual-memory layout.

a bridge for calling from user-defined Java code to sandbox's core. The core engine aims to find target methods in virtual memory, load user-supplied DEX files, hijack the vtable, and set native hooks. Moreover, it registers the native methods callable from the Java side. Sandbox is configured by reading a user-supplied JSON formatted configuration file containing the target methods list.

Figure 2 represents the apps memory layout while sandbox hooking library is enabled. The hooking library is loaded inside the apps virtual memory (step 1), and then sandbox loads the user-defined patch code by DexClassLoaders methods (step 2). After this, sandbox uses its internal functions to retrieve target methods reference. It can hook these methods by both vtable and virtual methods hijacking (step 3).

To get the target methods reference, sandbox uses the JNI function FindMethodID. Sandbox overwrites the target methods entry within both the vtable and virtual methods array by writing the address of the methods patch code. The original methods reference is not modified by sandbox and its address is stored inside the internal data structures. This address will be used to call the original method implementation.

3.1. Hook Virtual Space. When the Android application starts the Activity, the ActivityManager.start-Activity() method will eventually be invoked no matter what API call is passed. This call is a remote Binder service (speeding up the call), and the Android application will first look up the Binder service cache in the local process. Virtual app(VA) intervened in the invocation process by the following way.

(1) Replace the local ActivityManagerServise(AMS) Binder service for the proxy object that is constructed for the VA to take over the call. This step is realized by Java reflection technology.

(2) After taking over the AMS, when the startActivity is invoked to open more applications, the Activity in the VA modification Intent is the occupied Activity that has been declared in VA. The goal of this step is to directly start Activity without AndroidManifest.xml.

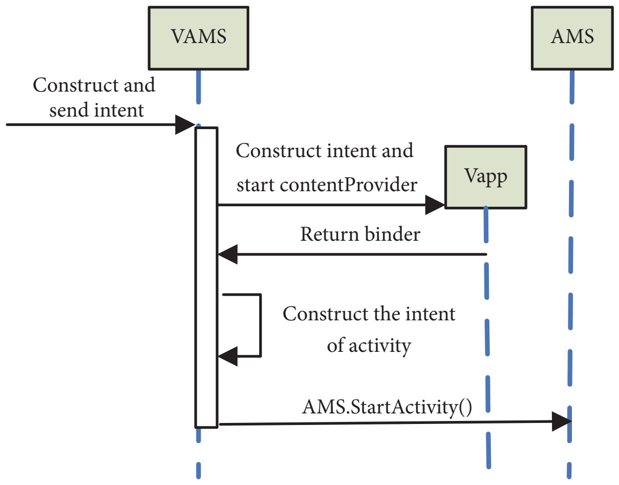

VAMS
AMS
Construct and send intent
Construct intent and start contentProvider
Vapp
Return binder
Construct the intent of activity
AMS.StartActivity()

FIGURE 3: Diagram of starting Activity.

(3) When the multiple application process is started, the message processing callback is increased by the ActivityThread.mH.mCallback. This step takes over the message callback of more than one application master threads.

On the basis of the above modification, the multiple application Activity startup process can be divided into the following two steps: start Activity (shown as in Figure 3) and resume Activity (shown as in Figure 4).

3.2. APP Risk Behavior. An application invokes an API that truly reflects the behavior of the application in the Android system. For example, an application that generates network behavior will certainly invoke the API associated with the network communication. An application that generates file behavior will certainly call the file-related APIs. Hence, to

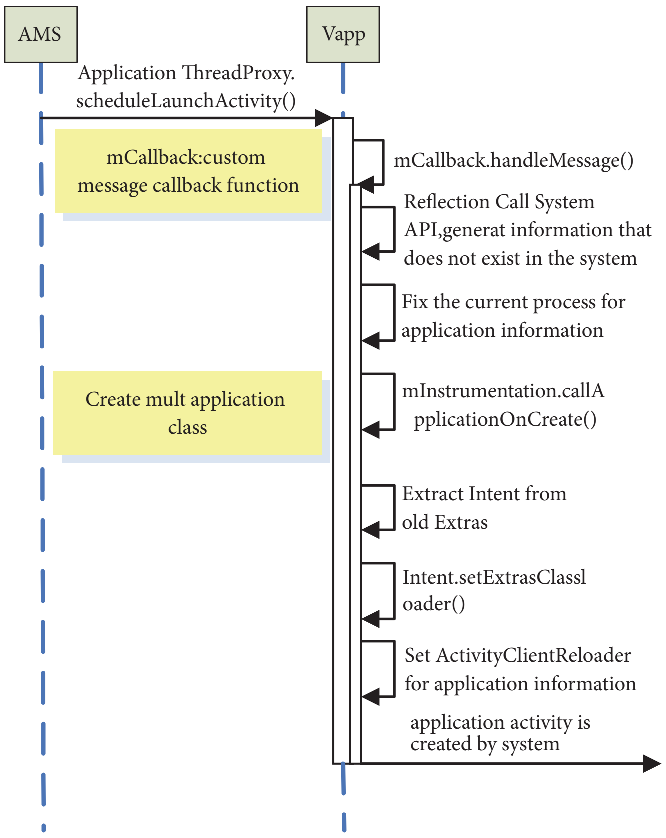

AMS
Vapp
Application ThreadProxy.
scheduleLaunchActivity()
mCallback:custom message callback function
mCallback.handleMessage()
Reflection Call System API,generat information that does not exist in the system
Fix the current process for application information
Create mult application class
mInstrumentation.callA pplicationOnCreate()
Extract Intent from old Extras
Intent.setExtrasClassl oader()
Set ActivityClientReloader for application information
application activity is created by system

Figure 4: mCallback resume Activity information from Intent.

describe an application's behavior, one can use it as a standard for invoking the API. Once users have installed malicious applications in Android, these apps typically have some of the following common malicious behaviors in Android:

(1) The backstage sending the charge message and calling the toll telephone.

(2) Theft of user information (including mobile phone messages, call records, mobile phone IMEI, IMSI number, and user-used operators);

(3) Access to the user's location information, to open the mic recording and camera in the backstage.

(4) Backstage networking, transmission of user information, and consumption of user network traffic.

(5) Camouflage process, in the backstage to kill other mobile phone processes (such as Alipay application process) and then camouflage another process to cheat.

According to the description above, if an application invokes one or more APIs required to implement the above behavior, there is a certain degree of risk that the user can install the application. Table 1 shows some of the APIs and their behavior levels.

When the risk is large to a certain extent, users should be informed. The level of risk represented by different APIs is not the same for risk APIs. In general, the user's economic interests as a direct risk measurement criteria:

(1) APIs that may directly cause loss of property to the user, with the highest degree of risk

(2) The API that can obtain or disclose user's privacy, its risk degree being secondary

(3) To modify the system settings and User Configuration, damage to the system environment of the API, the degree of risk being relatively low.

Based on the theory of information entropy, this paper proposes a new approach to evaluate the risk behavior of Android using information entropy. One installs the application into the sandbox and runs and simulates user actions for a fixed number of times, such as 500. Suppose that, in this process, all the sensitive APIs invoked are $k1, k2, \ldots, kn$, and set the information entropy used to evaluate the risk behavior of the application to δ, and set s as the total API numbers that has occurred for all behavior:

$$s = \sum_{i=1}^{n} k_i \tag{3}$$

$$\delta = -\sum_{i=1}^{n} \frac{k_i}{s} \times \log_b \frac{k_i}{s} \tag{4}$$

Because malicious applications generally focus on risk behavior, they frequently invoke sensitive APIs. This paper uses a set of sensitive APIs to describe and characterize an application. If δ is greater, its entropy will be more than the normal application of information entropy. By calculating an

TABLE 1: APIs and risk levels defined by authors.

Evaluation Project	Danger level	Evaluation Project	Danger level
Virus scanning	high	Apply data to any backup	medium
Sensitive word Information	medium	Apply Signature Not verified	medium
Advertising SDK Detection	low	Sensitive function calls	medium
Third-party SDK detection	low	Java Layer Dynamic debugging	low
Java Code decompile	high	Load Dex from SDcard	low
So file crack	high	Implicit invocation of intent components	low
Tampering and two-time packaging	high	WebView Remote Code	high
Dynamic injection attack	high	Database injection	high
Interface Hijacking	high	ContentProvider Data Disclosure	high
Input listening	high	Encryption method not safe to use	high
HTTP Transport data	high	HTTPS not verified	medium
WebView PlainText Store password	high	Download any apk	medium
PlainText digital certificate	high	Global writable Internal files	medium
Debug Log functions	high	DDoS	medium
Resource File Disclosure	medium	Residual test information	low
Dynamic Debug Attacks	medium	WebView Bypass Certificate validation	low
Activity Component Export	medium	Unsafe use of random numbers	low
Service component Export	medium	Intent Scheme URL	low
Broadcast receiver Component Export	medium	Fragment injection attack	low
Content Provider Component Export	medium		

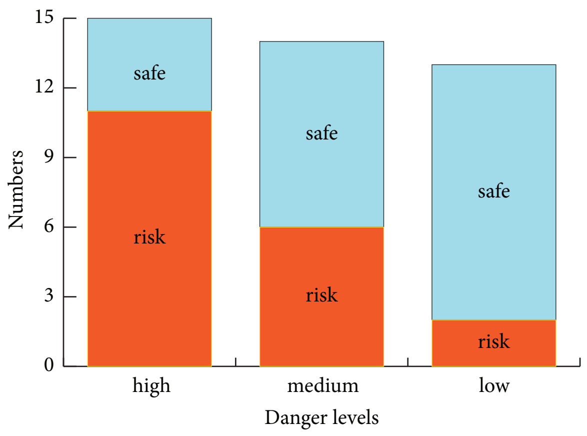


FIGURE 5: Test results for authclient.apk.

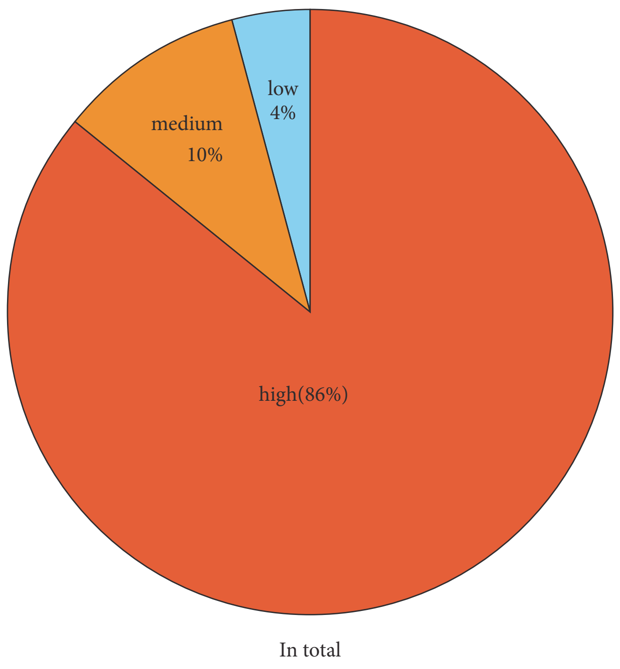


FIGURE 6: Found problems for authclient.apk.

application on the sensitive API set of information entropy δ, one can judge the level of risk.

4. Experiments and Results

We firstly test one app which is named authclient.apk downloaded from Google. Figures 5 and 6 show the test results.

This paper collects 1200 malicious Android apps that have been identified by each major mobile phone's security platform as a sample of malicious applications and crawls the top-ranked downloads in Google's official App Store, a total of 400 applications, as a normal application sample. In general, Google's official App Store is generally considered to be a more secure app store, and the application of the top-ranked Google stores has withstood the test of countless users and security vendors. It can be considered a safe and normal application. Based on the above 2 sample sets, the sandbox is used to monitor and record its API calls, and the results are statistically categorized as shown in Figures 7 and 8.

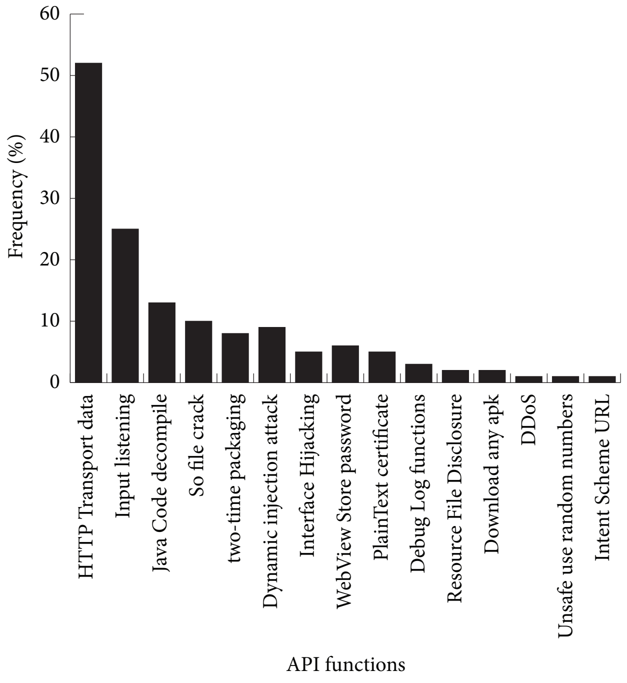


Figure 7: The information entropy of malicious apps.

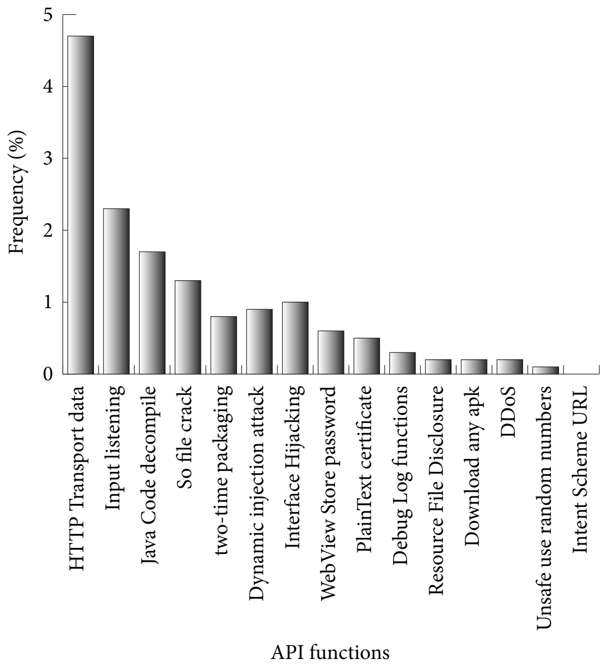


Figure 8: The information entropy of normal apps.

From the above results, network API calls occupy the main position for both malicious and normal application, and that is a trend in the future development of Android applications. In addition to the network API, there are malicious applications to other sensitive API calls and normal application of the obvious difference. As can be seen from Figures 6 and 7, malicious applications have significantly more frequent calls to sensitive APIs than normal applications, and these APIs can capture the user's IMEI code, IMSI code, phone number, SIM card sequence code, etc., so these APIs are obviously more risky than network APIs much higher.

5. Discussion and Conclusion

With the rapid development of mobile Internet, the Android applications have become an indispensable tool for people's daily life, and understanding the risks of Android applications is very important. This article focuses on the analysis and evaluation of the Android application risk behavior, monitors and records the behavior of Android applications through the Android sandbox, and uses the information entropy theory to analyze and evaluate the risk behavior of Android applications. This method can provide reference for application store review and also make Android users have a clear assessment of the risks associated with the application. Based on the above methods, this paper collects more than 1200 malicious applications and 400 normal applications, calculates the information entropy of both, compares them, and verifies the validity of the method described in this paper.

However, before being widely deployed in practical applications, this proposed sandbox has to tackle the privacy and efficiency challenges in sharing schemes [31]. We need to send users' data to clouds and guarantee the security of users data while ensuring rapid transmission of instant data [32, 33] and the security of public storage clouds [34, 35]. Note that mining data from multiple data sources to extract useful information [36, 37] for better understanding of security risk evaluation should be considered in the future study.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Foundation of Science, China (41274109), the Innovative Team Project of Sichuan Province (2015TD0020), and the New Zealand Marsden Fund.

References

[1] S. Blackshear, A. Gendreau, and B. Y. E. Chang, "Droidel: a general approach to android framework modeling," in *Proceedings of the ACM Sigplan International Workshop on State of the Art in Program Analysis*, pp. 19–25, 2015.

[2] A. Feizollah, N. B. Anuar, R. Salleh, and F. Amalina, *Comparative Evaluation of Ensemble Learning and Supervised Learning in Android Malwares Using Network-Based Analysis*, Springer International Publishing, 2015.

[3] Y. Zhou, X. Zhang, X. Jiang, and V. W. Freeh, "Taming information-stealing smartphone applications (on android)," in *Proceedings of the International Conference on Trust and Trustworthy Computing*, pp. 93–107, 2011.

[4] M. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '12)*, pp. 101–112, April 2012.

[5] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "CHEX: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS '12)*, pp. 229–240, October 2012.

[6] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "AdDroid: privilege separation for applications and advertisers in Android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security (ASIACCS '12)*, pp. 71-72, Seoul, Republic of Korea, May 2012.

[7] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. R. Sadeghi, and B. Shastry, "Towards taming privilege-escalation attacks on android," in *Proceedings of the Annual Network and Distributed System Security Symposium*, vol. 130, pp. 346–360, 2012.

[8] P. Zhao, K. Bian, T. Zhao et al., "Understanding smartphone sensor and app data for enhancing the security of secret questions," *IEEE Transactions on Mobile Computing*, vol. 16, no. 2, pp. 552–565, 2017.

[9] Z.-X. Shen, C.-W. Hsu, and S. W. Shieh, "Security semantics modeling with progressive distillation," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3196–3208, 2017.

[10] F. Roesner, T. O. Kohno, and D. Molnar, "Security and privacy for augmented reality systems," *Communications of the ACM*, vol. 57, no. 4, pp. 88–96, 2014.

[11] K. Moran, M. Tufano, C. Bernal-Cárdenas et al., "Mdroid+: a mutation testing framework for android," in *Proceedings of the the 40th International Conference*, pp. 33–36, Gothenburg, Sweden, May 2018.

[12] M. Hussain, A. Al-Haiqi, A. Zaidan et al., "A security framework for mHealth apps on Android platform 1," *Computers & Security*, vol. 75, pp. 191–217, 2018.

[13] V. Rastogi, Y. Chen, and W. Enck, "AppsPlayground: automatic security analysis of smartphone applications," in *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy (CODASPY '13)*, pp. 209–220, ACM, February 2013.

[14] S. Arzt, S. Rasthofer, C. Fritz et al., "FLOWDROID: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps," *ACM Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[15] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in *Proceedings of the International Conference on Trust and Trustworthy Computing*, pp. 291–307, 2012.

[16] A. P. Fuchs, A. Chaudhuri, and J. S. Foster, "Scandroid: Automated security certification of android applications," in *Proceedings of the 31st IEEE Symposium on Security*, 2009.

[17] Y. Zhou and X. Jiang, "Systematic detection of capability leaks in stock android smartphones," in *Proceedings of the Nineteenth Annual Network and Distributed System Security Symposium (NDSSS '12)*, 2012.

[18] D. Katz and A. Nagy, "VuFind: solr power in the library," in *Open Source Technology: Concepts, Methodologies, Tools, and Applications*, 2015.

[19] H. P. Yuling Liu and J. Wang, "Verifiable diversity ranking search over encrypted outsourced data," *CMC, Computers, Materials & Continua*, vol. 55, no. 1, pp. 037–057, 2018.

[20] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," in *Proceedings of the 28th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '13)*, pp. 641–660, October 2013.

[21] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps," in *Proceedings of the International Conference on Mobile Systems, Applications, and Services ACM*, pp. 204–217, 2014.

[22] W. Qi, W. Ding, X. Wang et al., "Construction and mitigation of user-behavior-based covert channels on smartphones," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 44–57, 2018.

[23] W. Fan, Y. Sang, D. Zhang, R. Sun, and Y. Liu, "DroidInjector: a process injection-based dynamic tracking system for runtime behaviors of android applications," *Computers & Security*, vol. 70, pp. 224–237, 2017.

[24] A. Gharib and A. Ghorbani, "DNA-Droid: a real-time android ransomware detection framework," in *Proceedings of the International Conference on Network and System Security*, pp. 184–198, 2017.

[25] A. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," in *Proceedings of the 20th Design, Automation and Test in Europe Conference and Exhibition (DATE '17)*, pp. 1691–1696, March 2017.

[26] Y. Liu, S. Wang, Y. Yang, Y. Chen, and H. Sun, "An automatic UI interaction script generator for android applications using activity call graph analysis," *EURASIA Journal of Mathematics, Science and Technology Education*, vol. 14, no. 7, pp. 3159–3179, 2018.

[27] L. Xin, "Malware detection technology research of android platform based on sandbox," *Electronic Design Engineering*, vol. 24, no. 12, pp. 48–50, 2016.

[28] J. Cui, Y. Zhang, Z. Cai, A. Liu, and Y. Li, "Securing display path for security-sensitive applications on mobile devices," *CMC: Computers, Materials & Continua*, vol. 55, no. 1, pp. 017–035, 2018.

[29] J. Cheng, R. Xu, X. Tang, V. S. Sheng, and C. Cai, "An abnormal network flow feature sequence prediction approach for ddos attacks detection in big data environment," *CMC: Computers, Materials & Continua*, vol. 55, no. 1, pp. 095–119, 2018.

[30] Y. Liu, W. Guo, C. Fan, L. Chang, and C. Cheng, "A practical privacy-preserving data aggregation (3PDA) scheme for smart grid," *IEEE Transactions on Industrial Informatics*, 2018.

[31] Z. Cai, H. Yan, P. Li, Z.-A. Huang, and C. Gao, "Towards secure and flexible EHR sharing in mobile health cloud under static assumptions," *Cluster Computing*, vol. 20, no. 3, pp. 2415–2422, 2017.

[32] C. Wang, J. Shen, Q Liu I, Y. Ren, and T. Li, "A novel security scheme based on instant encrypted transmission for internet of things," *Security and Communication Networks*, vol. 2018, pp. 1–7, 2018.

[33] J. Shen, C. Wang, T. Li, X. Chen, X. Huang, and Z.-H. Zhan, "Secure data uploading scheme for a smart home system," *Information Sciences*, vol. 453, pp. 186–197, 2018.

[34] L. Yang, Z. Han, Z. Huang, and J. Ma, "A remotely keyed file encryption scheme under mobile cloud computing," *Journal of Network and Computer Applications*, vol. 106, pp. 90–99, 2018.

[35] L. Jin, L. Yatkit, C. Xiaofeng, L. Patrick, and L. Wenjing, "A hybrid cloud approach for secure authorized deduplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 90–99, 2015.

[36] R. Wang, W. Ji, M. Liu et al., "Review on mining data from multiple data sources," *Pattern Recognition Letters*, vol. 109, pp. 120–128, 2018.

[37] J. Wu, J. Long, and M. Liu, "Evolving RBF neural networks for rainfall prediction using hybrid particle swarm optimization and genetic algorithm," *Neurocomputing*, vol. 148, pp. 136–142, 2015.