Interaction-Aware System Call Sequence Analysis for Android Malware Classification

Jae-Min Choi[1][0009−0008−9459−3535], Sang-Hoon Choi[2][0000−0002−9549−0887], and Ki-Woong Park[3][0000−0002−3377−223X]

[1] SysCore Lab, Department of Information Security, and Convergence Engineering for Intelligent Drone, Sejong University, Seoul 05006, South Korea
c.jaem7532@gmail.com
[2] SysCore Lab, Sejong University, Seoul 05006, South Korea
csh0052@gmail.com
[3] Department of Information Security, and Convergence Engineering for Intelligent Drone, Sejong University, Seoul 05006, South Korea
woongbak@sejong.ac.kr

Abstract. The increasing number of mobile devices and the expansion of IoT (Internet of Things) and digital systems have made them targets for more sophisticated Android malicious apps. Existing detection techniques often use randomly generated events using 'monkey' tools, but this has limitations for apps that rely on user input for malicious behavior. In this paper, we extracted system calls and changed them to sequences to classify families of malicious apps by type with the DTW(Dynamic Time Warping) Algorithm. We evaluated using the AndroZoo dataset and found that family classification is possible for types of Trojan, Adware, and Exploit. In this study, we used view tree-based interaction for classification without using the 'monkey' tool to extract system call logs. We also extracted all logs through system call filtering to extract more malware behavior-focused system call logs. In the future, we plan to use more sophisticated interaction tools and ML and DL for classification and detection.

Keywords: Android Malware · Malware Family Classification · Malware Analysis

1 Introduction

The increasing of mobile devices and expansion IoT and digital systems, the information stored in mobile devices has become information that can cause direct or indirect damage to the user's life or company beyond personal information. To take this information, attackers are building and distributing more sophisticated and insidious malicious apps than ever before.

Attackers are not only distributing malicious apps through phishing and scams, but also by registering legitimate apps in official stores (such as Playstore and Apple Store) and adding malicious behavior through subsequent updates [1]. Attackers are also taking advantage of the wide compatibility of mobile

devices and the scalability of development tools to deploy malicious apps that are undetectable by traditional security technologies.

In March 2025, attackers used a C#-based Android and iOS app development tool Microsoft '.NET MAUI' to deploy malicious apps that bypassed detection that focuses on analyzing and detecting existing DEXs (Dalvik Executable) and native libraries [2]. In addition, another malicious app family called 'Crocodilus' appeared in March 2025, which used a dynamic code loading method through an external .dex file to bypass security devices [3]. In June 2024, a malicious app called 'DroidBot' appeared [4]. This malicious app operated as a MaaS (Malware-as-a-Service), similar to RaaS (Ransomware-as-a-Service).

Malicious apps or malicious code distributed as XaaS (X-as-a-Service), tend to have similar code logic, attack methods, and penetration methods, and can be detected and blocked by similar detection and blocking technologies. Attackers are becoming more advanced and sophisticated in their attacks on mobile devices and deploying malicious apps.

Several static, dynamic, and hybrid analytics are being studied to detect and classify mobile malicious apps [5–7]. The above studies use 'monkey' tools to simulate user interactions to maximise the code coverage of malicious apps and collect system call logs for detection and classification. However, in the case of Trojan-type malicious apps, the focus is on taking user information. Therefore, Trojans usually have an input box that takes user input.

However, the 'monkey' tool has difficulty handling such inputs because it relies on random-based events (unless you write your own code). Also, collecting all system call logs for detection and classification is not enough to detect and classify actual malicious behavior, but if you use all of them as features, the resources used for Analysis will be considerable because all behaviors are recorded, including evasion logic intentionally planed by the attacker, internal storage for normal app activities, and access to external storage. In this research, we identify a number of those issues and present them as follows.

- *C01: Handling the attacker's intended victim's behavior:* In the case of Trojan Type, the purpose is to take the user's information. Therefore, in order for the user's input value to be sent to the C2 (Command & Control) server, the input value must be filled in according to the acttacker's intention.

- *C02: Handling screen transitions from malicious apps:* Some malicious apps use automated tools like 'monkey' to perform screen transitions to evade analysis. In such cases, it is not possible to extract system calls correctly.

- *C03: Handling user behavior for maximum coverage:* Some malicious apps have malicious behavior that does not start immediately up on app launch, but rather after a specific button is touched.

- *C04: Handling non-malicious behavior logs from system calls:* If an attacker is intentionally planting evasion logic or collecting all legitimate app activity, the resources used for analysis will be very large.

To solve the above problem, we propose a framework that extracts system calls in changed sequence logs through preprocessing and filtering of user interactions and logs for more accurate classification.

This thesis is organized as follows. Section 2 describes related work on system calls and describes the libraries used for our analysis. Section 3 describes the design of the framework proposed in this thesis and details its internal modules. Section 4 describes the results of classification using the framework. Section 5 presents conclusions, limitations, and future work.

2 Related Work

2.1 Android Malware Detection Based on System Call

Ahmed R. Nasser et al. proposed a DL-AMDet architecture for Android malicious apps based on static and dynamic features [5]. The study extracted permission request in the Manifest.XML file and API (Application Programming Interface) Calls through reverse engineering of the .dex file from static analysis, ran the malicious app for 15 minutes at 1-minute intervals to extract system calls generated by the malicious app, and conducted dynamic analysis by generating events using 'monkey'. Based on the above three features, the CNN-BiLSTM method was used for training, and finally, 99.935% accuracy was achieved. In this study, the hybrid method effectively detected malicious apps.

Xiaojian Liu et al. extracted system calls for Android malicious apps and converted them into sequences to detect malicious apps using a CNN-LSTM model [6]. In this study, the system call is extracted and the frequency of occurrence is calculated to create a data dictionary. Then, based on the dictionary, it is converted into numerical information and trained with the CNN-LSTM model. At this time, 1000 random events were generated using 'monkey' to conduct dynamic analysis. In the end, 87.92% accuracy was achieved. In this study, system call was used to effectively detect malicious apps.

Christopher Jun Wen Chew et al. proposed a behavior-based ransomware detection technique based on extracting system calls generated by crypto ransomware [7]. The system calls generated by crypto ransomware are extracted, and the system calls are patterned to create behavior patterns. Then, based on regular expressions, we created an implementation that can detect encrypted ransomware behavior in real time through FSM (Finite State Machine). In this study, presented the possibility of using it as part of a self-protection system on Android devices is explored.

2.2 Android View Client

The AndroidViewClient library by dtmilano is a Python library that helps automate and test the UI of Android apps [8]. The library extracts the UI (User Interface) structure based on the ADB (Android Debug Bridge) provided by Android and supports touch, text input, screen capture, structure analysis, etc., for each View. In this study, we used this library to identify touchable views and input boxes. Then, we utilized the identified views by touching them and typing text into the input box.

3 Proposed Framework

This section describes the proposed framework and each module within the framework in detail. The structure of the proposed framework is shown in Fig. 1

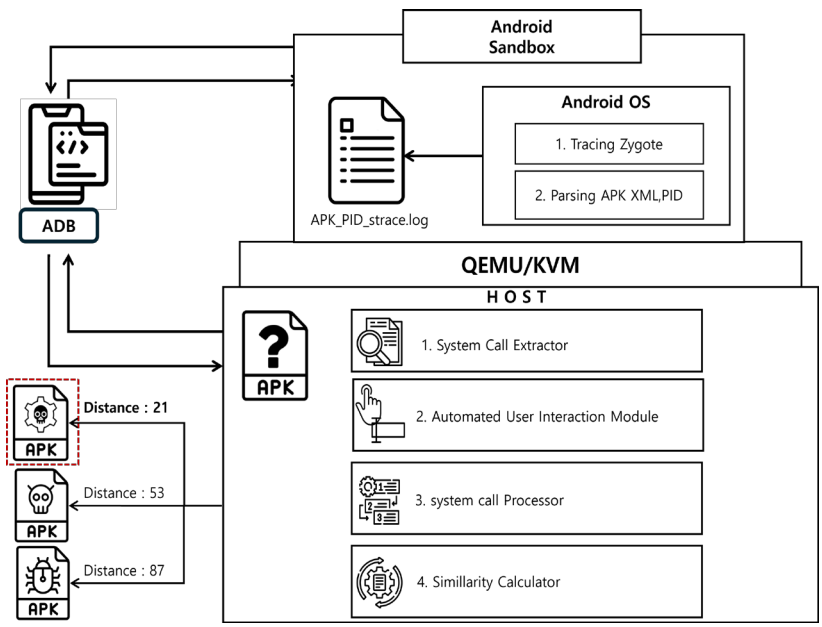

Fig. 1. Proposed Framework

3.1 System Call Extractor

Install the malicious app using ADB on an Android virtual machine created with QEMU (Quick Emulator)+KVM (Kernel-based Virtual Machine). Then, extract the XML using the Android 'aapt' tool. Extract the 'android.intent.action.MAIN' value in the extracted XML to launch the Android app. If there is a loading splash in the form of 'android.intent.category.LAUN-CHER', use that value. If the PID is extracted for the analysis app that is finally executed, it is impossible to extract the correct system call because it has already been executed.

To solve this problem, attach the 'strace' command to the 'zygote' or 'zygote64' process used by Android, including all child processes. This will allow to extract all system call logs from the moment the malicious app is executed. However, if we extract all system call functions, we will get a lot of noise. For example, functions such as 'ioctl' and 'get_clocktime' are not related to the actual

malicious behavior. Excluding these functions, we extracted only the functions listed in Table 1 to extract only the functions listed in Table 1.

Table 1. Filtering System Call Functions List

Category	**System Call Functions**
Process Control	clone, execve, fork, getuid, getuid32, geteuid, geteuid32
File I/O and Socket Connection	accept, bind, connect, getsockopt, mkdir, mkdirat, open, openat, pread64, read, readlinkat, recv, recvfrom, recvmsg, rename, renameat, rmdir, send, sendto, sendmsg, setsockopt, socket, stat, unlink, unlinkat, vfork, write, writev

Android requires the 'minsdk' and 'maxsdk' versions to be specified when developing an app. Therefore, if the app to be analyzed does not match the current virtual machine and SDK (Software Development kit), it cannot be executed. Therefore, we saved the app with the error and extracted the Android SDK version 11. Through the above process, we extracted 'zygote' and all system calls generated by the malicious app.

3.2 Automated User Interaction Module

Some malicious app types may start malicious behavior depending on the user's input. As mentioned in C01 and C03, Trojan-type malicious apps require user input to send the value to the C2 server, and the user must press the final send button, not just fill in the input box. In addition, there are cases where the screen is switched when a specific button is touched to prevent an evade analysis by automated tools such as C02.

To solve C01, C02, and C03, we used the 'AndroidViewClient' library to extract the view tree for the current screen. Then, we extracted all the view IDs based on the view tree, and used a recursive function to navigate through the view tree. At this time, we categorized touch and input box types based on the view type.

The touch-type view simulates the user's input through the touch event, and the input box type view is divided into various cases.

For input boxes, you may want to validate input values when accepting certain forms, such as email. Also, there are cases where only numeric values, such as age, card number, and zip code, are accepted. To solve this problem, we created a set of input values in the form of a dictionary, extracted the screen using the ADB 'screencap' command, and entered a different type of input if the input values were not entered correctly. This way, we can simulate the user interaction correctly.

There are cases where the screen switches when the above touch event occurs. To prevent this, we used the 'dumpsys' command to extract the activity of the current foreground state. Using this command, we can find the current

foreground activity, app, and compare it with the package name that was executed through it, and if the screen was switched, we solved it by using the 'back' command to return to the screen of the existing malicious app.

All of the above processes are recursive, meaning that we navigate from the parent view tree to the child view trees of the analysis app, and when the navigation is complete, we kill the process to end the analysis.

3.3 System Call Processor

The extracted system call logs contain all system calls made by 'zygote' and the malicious app. If all of these logs are used, the resources for analyzing them are considerable, as mentioned in C04 above, and there are difficulties in classifying them correctly. To solve this problem, the system call processor goes through the following steps.

From the extracted system call logs, we extract the system call logs of the app PID to be analysed and the child process that occurred in the app. Then, to extract the system call of the actual malicious behavior, we defined the parameter according to the system call function as shown in Table 2 and extracted the system call containing the parameter.

Table 2. Filtering System Call Parameter List

Functions	**System Call Parameters**
openat()	base.apk, /storage/emulated/0, /data/user/0, /sdcard
Version 7 wrtiev()	onCreate, onDestroy, onResume, LAUNCH_ACTIVITY, handlePauseActivity, destroy, handleRelaunchActivity, performPause
Version 9 wrtiev()	MAIN_ACTIVITY, performCreate, performPause, performRestartActivity, performDestroy, handleStartActivity, RESUME_ACITIVTY, apache
write()	<?XML, getaddrinfo, Timeout, http, telnet
clone()	!zygote
mkdirat()	/storage/emulated/0, /data/user/0, /sdcard
renameat()	/storage/emulated/0, /data/user/0, /sdcard
connect()	AF_INET, AF_INET6
pread64()	table
socket()	!PF_LOCAL
recvmsg()	!-1
recvfrom()	!72, !232

To apply the extracted system call logs to the DTW algorithm, we mapped each function to an integer as shown in Table 3

Based on the above process, the final extracted system call log is much smaller in size than the full system call log, including 'zygote', and contains only malicious behavior due to noise removal. The data was reduced from a maximum of 140 MB to a minimum of 41 KB.

Table 3. System Call Functions to Integer for DTW Algorithm

System Call Function	Integer	System Call Function	Integer
clone()	1	recv()	19
execve()	2	recvfrom()	20
fork()	3	recvmsg()	21
getuid()	4	rename()	22
getuid32()	5	renameat()	23
accept()	6	rmdir()	24
geteuid()	7	send()	25
geteuid32()	8	sendto()	26
accept()	9	sendmsg()	27
connect()	10	setsockopt()	28
getsockopt()	11	socket()	29
mkdir()	12	stat()	30
mkdirat()	13	unlink()	31
open()	14	unlinkat()	32
openat()	15	vfork()	33
pread64()	16	write()	34
read()	17	writev()	35
readlinkat()	18		

3.4 Similarity Calculator

To evaluate the similarity using the final extracted data, we used the DTW algorithm to evaluate the similarity. DTW algorithm is useful for evaluating the similarity of time series data regardless of the difference in data length through time base correction. In addition to the DTW algorithm, we also used Cosine Vector distance similarity and the SBD (Shape-based distance) algorithm, but the classification was not correct, so we finally evaluated the similarity with the DTW algorithm.

Before applying the DTW algorithm, the converted sequence logs extracted the maximum and minimum lengths within each type to select the maximum and max ratio values according to the distance difference. Even within the same family, we added a correction for the absolute length difference by calculating the case where the code changed according to the production time and adopted a new attack method.

Finally, the DTW algorithm was used to measure the distance value of each family within each type and normalise it to extract it.

4 Classification and Evaluation

This section details the classification results for the AndroZoo dataset [9] using our framework. We analyzed 282 samples with a VirusTotal Score over 30, divided into Adware (14 families), Exploit (2 families), and Trojan (39 families). The experimental setup is described in Table 4.

Table 4. An Environment for Framework

Environment	**Environment Spec**
Host OS	Ubuntu 64bit 22.04
Guest OS	Android-x86-11.0, Android-x86-9.0, Android-x86-7.1
CPU	Intel(R) Core i7-13700KF
Memory	DDR5 32GB

Table 5. Adware DTW with Sample1-5

Family	**sample1**	**sample2**	**sample3**	**sample4**	**sample5**
droidkungfu-1	999.0	157.8	999.0	999.0	406.1
droidkungfu-2	999.0	999.0	999.0	999.0	999.0
fakerun	**156.2**	999.0	999.0	999.0	999.0
gappusin	999.0	999.0	67.3	72.7	999.0
gingermaster	999.0	377.3	999.0	999.0	**0.0**
ginmaster	999.0	999.0	**8.3**	19.1	999.0
iadpush	999.0	999.0	55.3	56.4	999.0
ksapp	999.0	999.0	46.0	50.4	999.0
morepaks	999.0	999.0	38.1	41.0	999.0
nandrobox	999.0	999.0	999.0	999.0	999.0
plankton	999.0	**101.4**	999.0	999.0	383.5
utchi	999.0	999.0	21.9	**10.6**	999.0
wapsx	999.0	999.0	999.0	999.0	999.0
waps	999.0	125.5	999.0	999.0	392.6
youmi	999.0	999.0	32.3	35.7	999.0

Table 6. Exploit DTW with Sample6

Family	**sample7**
gingerbreak	**0.0**
lotoor	999.0

4.1 DTW Distance by Type

Proceed to extract the DTW Distance by malicious app type. At this time, representative values were extracted to create data within the same family. This process took about 20 times more time compared to the unfiltered data. The process of calculating DTW distance values using the filtered data in this experiment took about an hour. After extraction, the data that was labelled as 'family-n' was above to distinguish the different sequence patterns by version within the same family.

For example, in the case of the 'droidkungfu-1', 'droidkungfu-2' family of adware, depending on the version, the way of showing advertisements is divided into advertisement output through WebView and advertisement output through video output. In the case of Trojan type, it is divided into the case of sending user value (such as Gmail ID, phone number, phone model) to the C2 server after executing the malicious app and the case of send user input value to the C2 server by touching the button after receiving all user input value.

The above distinction was made to recognise cases where malicious behavior within the same family is different.

4.2 Classification Result

For evaluation, we randomly extracted 10 sequence logs from all data sets and compared the distance using DTW algorithm with the sequence logs of each family. We assumed that the same family within the same type would show the same sequence behavior across versions. We conducted the experiment based on the hypothesis that 'malicious app families can be classified by comparing their similarity', and the final results are shown in Tables 5, 6, and 7.

At the data in Tables 5, 6, 7, for each family, there is a family that has the minimum number of distance. This means that sequence log belongs to that family. we compare the classified result with the labelled result to confirm that it belongs to the actual family. The data is different from the reference sequence logs, and we verified that the correct sequence logs can be classified into families.

4.3 Limitation

In our proposed study, the sequence logs generated by the framework were finally evaluated for similarity using the DTW algorithm. The dataset is relatively small to train ML (Machine Learning) and DL (Deep Learning) models, so we did not extend it to ML and DL. In addition, in the case of the interaction module, the process of verifying whether each data is entered correctly is based on the screencap, and if the data is entered, it is considered to be entered correctly. However, if the attacker did not write logic for input value validation in the input box, incorrect values may be recognized as correct values. As such, we think that interaction processing requires more accurate interaction processing than just view tree, screencap verification, and image comparison.

Table 7. Trojan DTW with Sample7-10

Family	**sample7**	**sample8**	**sample9**	**sample10**
adrd	**0.0**	999.0	999.0	42.6
andcom	65.6	999.0	999.0	56.2
andup	999.0	69.7	61.0	999.0
artemis	999.0	65.9	62.5	999.0
ddlight	999.0	**43.7**	55.8	999.0
dougalek	63.9	999.0	999.0	6.4
fakedoc	999.0	72.2	57.2	999.0
fakelogo	42.5	999.0	999.0	22.0
fakerun	999.0	110.5	999.0	999.0
gamex	999.0	74.0	65.2	999.0
geinimi	33.0	999.0	999.0	34.7
gingermaster	999.0	999.0	999.0	999.0
ginmaster	999.0	77.7	76.5	999.0
golddream	999.0	109.1	91.3	999.0
gopf	50.1	999.0	999.0	999.0
iconosys	999.0	55.1	61.7	999.0
jifake	33.0	999.0	999.0	22.0
jsmshider	64.4	999.0	999.0	999.0
kmin	42.6	999.0	999.0	**0.0**
ksapp	999.0	78.4	63.7	999.0
lovetrap	999.0	96.1	89.5	999.0
morepaks	41.7	999.0	999.0	23.4
nandrobox	999.0	61.2	**31.3**	999.0
opfake	50.8	999.0	999.0	39.2
pjapps-1	999.0	999.0	80.1	999.0
pjapps-2	54.7	999.0	999.0	44.2
placms	32.5	999.0	999.0	21.1
ramnit	999.0	80.1	999.0	999.0
roguesppush	85.1	999.0	999.0	999.0
rufraud	59.8	999.0	999.0	17.4
smcc	999.0	82.9	60.9	999.0
smszombie	81.3	999.0	113.7	999.0
stiniter	26.9	999.0	999.0	33.5
uuserv-1	999.0	54.3	59.9	999.0
uuserv-2	77.1	999.0	82.6	999.0
vdloader	999.0	99.9	999.0	999.0
waps	50.9	999.0	999.0	28.0
winge	48.3	999.0	999.0	20.8
yzhc	33.0	999.0	999.0	22.0
zitmo	46.2	999.0	999.0	21.1
zsone	47.3	999.0	999.0	20.6

5 Conclusion

In this paper, we propose a framework to extract more accurate system calls by performing the attacker's intended interaction with malicious apps, pre-process and convert the system calls to generate sequence logs, and classify malicious app families based on them. Using the proposed framework, we verified that 14, 2, and 39 families of adware, exploit, and trojan types show the same sequence patterns. We also tested the DTW algorithm on 10 malicious apps based on the sequence logs to show that it is possible to classify them correctly. This shows that it is possible to classify malicious app families through system calls extracted through an accurate interaction process rather than interaction processing based on random events.

To date, system call logs have been proposed as a key feature in malicious app detection and classification research using various dynamic and hybrid analytics. Most of the proposed methods for extracting system calls in these studies use 'monkey' tools, which simulate interactions by generating events based on randomness. This makes it difficult for the attacker to perform their intended behavior. If a system call is extracted based on the interaction simulation method used in this framework, it is expected that it can be utilized as a better feature.

Acknowledgments. This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) (2022-11220701, 40%), and the National Research Foundation of Korea (NRF) grant funded by the Korean Government (Project No. RS2023-00208460, 60%).

References

1. Kaspersky, https://securelist.com/it-threat-evolution-q1-2024-mobile-statistics/112750/, last accessed 2025/06/27
2. McAfee, https://www.mcafee.com/blogs/other-blogs/mcafee-labs/new-android-malware-campaigns-evading-detection-using-cross-platform-framework-net-maui/, last accessed 2025/06/27
3. Threatfabric, https://www.threatfabric.com/blogs/exposing-crocodilus-new-device-takeover-malware-targeting-android-devices, last accessed 2025/06/27
4. SecurityWeek, https://www.securityweek.com/droidbot-android-trojan-targets-banking-cryptocurrency-applications/, last accessed 2025/06/27
5. Liu, X., Zhang, Y., Duan, Y., Hou, B.: Android Dynamic Malware Detection Method Based on System Call Sequences. In: 2024 9th International Conference on Intelligent Computing and Signal Processing (ICSP), pp. 275-279. IEEE, Xian, China (2024). https://doi.org/10.1109/ICSP62122.2024.10744001
6. Chew, C.J.W., Kumar, V., Patros, P., et al.: Real-time system call-based ransomware detection. International Journal of Information Security 23, 1839–1858 (2024). https://doi.org/10.1007/s10207-024-00819-x
7. Nasser, A.R., Hasan, A.M., Humaidi, A.J.: DL-AMDet: Deep learning-based malware detector for Android. Intelligent Systems with Applications 21, 200318 (2024). https://doi.org/10.1016/j.iswa.2023.200318

8. dtmilano, https://github.com/dtmilano/AndroidViewClient, last accessed 2025/06/27
9. Allix, K., Bissyandé, T.F., Klein, J., Traon, Y.L.: AndroZoo: Collecting millions of Android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468–471. IEEE, May (2016). https://doi.org/10.1109/MSR.2016.061