Android Adware Detection using Soot and CFG

Jungsoo Park and Souhwan Jung*,

Soongsil University, Seoul, Korea
{jspark, sohwanj}@ssu.ac.kr

Received: October 10, 2022; Accepted: December 4, 2022; Published: December 31, 2022

Abstract

Adware is the most common type of malware. While considered not harmful in nature, it disrupts the user experience and generates unwanted revenue. Adware is also difficult to analyze and detect, therefore it is actively distributed through the Google Play Store. In the case of known adware detection, analysis takes a long time due to prolonged dynamic analysis to reveal Ads, and the result of network traffic analysis is difficult to determine traffic such as multimedia streaming, which results in a high probability of false positives. In this paper, we introduce a method to efficiently detect adware using static analysis. CFG(Control Flow Graph) analysis was performed using Soot to identify the adware signatures. 52 signatures were obtained through analysis of 1000 samples, and the optimized detection efficiency was measured while changing the depth (level) of the CFG. In addition, it achieves 91.92% of accuracy when analyzing 1380 normal and 4490 adware samples. This approach has similar detection rate in a shorter time compared to the existing dynamic analysis, and it does not require operation check as in previous dynamic analysis approaches.

Keywords: Android Adware, Android Malware, Soot, Control Flow Graph

1 Introduction

Android is already being used as the most used OS beyond Windows. As of April 2021, the Android OS usage was 39.24%, showing a steady increase as it is used not only in mobile but also in IoT and connected cars. With the increase in usage and development of Android, various Android malware is also appearing. In the second quarter of 2020, more than 1.2 million new Android malware appeared[1]. In particular, in the case of Android OS, since it is distributed to many users through mobile devices, a number of malicious codes are occurring for the purpose of personal information leakage and money stealing. of malicious codes were also generated[2]. In particular, since the main source of revenue for mobile is through advertisements to promote various products and services, Adware is also rapidly increasing[3]. As can be seen in [Figure 1], among Android malware, Adware is being raised as the biggest problem. Various studies have been conducted to detect adware, and Google also tried to respond through Android security updates. However, detection of Adware is still not performed properly, and damage occurred due to Adware such as Adware Campaign in 2020[4]. In order to respond to such adware, we created the Control Flow Graph using Soot, and then created the signature for the API used. Thereafter, adware detection was performed using the similarity of CFG to detect known adware based on static analysis, and the detection rate was measured. This paper is organized as follows. Chapter 2 examines research related to Adware and explains the technology proposed in Chapter 3. Chapter 4 examines adware detection performance based on the proposed technology and concludes in Chapter 5.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 13(4):94-104, Dec. 2022
DOI: *10.58346/JOWUA.2022.I4.006*

*Corresponding author: School of Electronic Engineering, Soongsil University, Hyungnam memorial building 1102, 369 Sangdo-Ro, Dongjak-Gu, Seoul, Korea, Tel: +82-02-824-1807

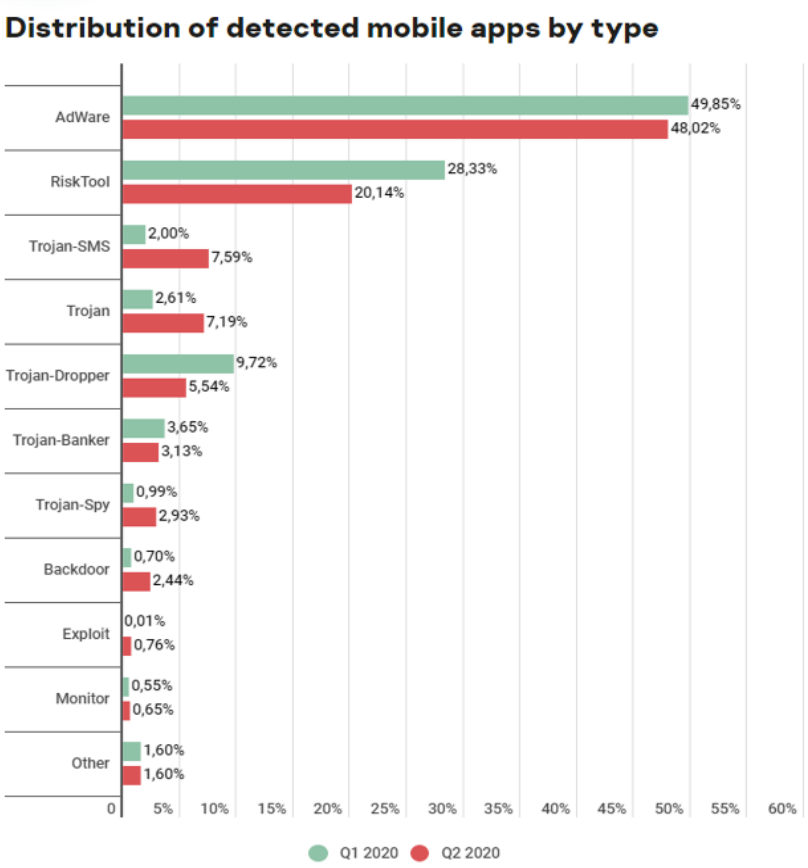


Figure 1: Distribution of detected mobile apps by type

2 Related Work

Advertising is used to promote a product, idea or service. With the development of the Android platform, advertisements through Android applications are continuously increasing. 96.1% of all apps registered on the Google Play Store in Q2 2021 are free mobile applications[5]. In the case of free applications, since most of the revenue is based on advertisements provided to users, various malicious actions are occurring in terms of posting and managing advertisements. Advertisements obtained from advertising providers are categorized into cost per click, cost per mile, cost per action, and the like, and pricing is based on the generated ad impressions. Android developers include the SDK of the advertisement library in the application code, and support to display advertisements to users with the help of necessary permissions. Advertisers pay a certain amount to application developers and ad inserts for each ad impression, i.e., each ad that is brought and displayed to the user. Developers can also get paid every time a user clicks on a displayed ad. This is exploited to bring advertisements, but it is not displayed properly to the user or the app triggers an attack in the form of clicking the advertisement without user activity. In the actual advertisement exposure method of Adware, when the user unlocks the lock, a full screen advertisement is always displayed, and the Close or Exit menu is not visible, so that the user cannot immediately quit and watch the advertisement for a certain period of time. After that, the Close button appears after a certain amount of time has elapsed, and by designating the interval at which advertisements are generated in the internal code, advertisements are continuously generated for at least 5 minutes. After that, if the home button or the back button is pressed to escape from the advertisement,

a selection menu to select another app appears or does not operate, causing inconvenience to the user. In the latest Adware, which acts similarly to logic bomb, after being installed, it is sometimes coded to start an activity after a certain amount of time. This delay is to prevent the malicious code from being easily classified as malicious because the dynamic analysis is monitored only for a certain period of time from the standpoint of analyzing the malicious code. In addition, as a method of performing an action for displaying an advertisement, it also includes a function of receiving an action for performing an advertisement from the C&C server[6][7][8][9]. Looking at the existing adware detection technology, a machine learning-based technology that detects adware based on static and dynamic functions has been proposed. Multi-class adware classification was performed based on the information collected from the manifest file and the dynamic function from the network traffic. In the case of *RandomForest*, 99.4% classification was performed and in *BayesNet* 96.14% classification was performed[10]. Through this, it was proved that the accuracy is superior to the existing binary detection problem. In another study, an analysis method using the CICAGM data set that captures network traffic by installing an Android application on an actual Android device was used to detect abnormal traffic through network traffic. For general and adware data sets, we collected and analyzed 1500 apps from Google Play Store, showing a detection rate of 93.81%[11][12][13]. However, it is necessary to analyze it dynamically and figure out whether a reward is given after the advertisement. In the case of dot and screen analysis, it is difficult for non-analyst analysts to respond to procedures such as closing the page and confirming the normal path to the connected URL. In the related studies described above, in order to respond to these contents, the analyst directly performed the analysis until AD was actually performed. In the case of this analysis, it takes a lot of time and does not analyze applications that do not reach AD. If not, the reliability of the data set may be lowered.

3 Proposed Scheme

Methods using static analysis and dynamic analysis were introduced to detect Android Adware. In the case of dynamic analysis, it is difficult if the analyst does not check the AD, identify the rewards from the actual AD, and separate AD from the adware. Therefore, in this paper, analysis is performed using signature and CFG using static analysis. We performed static analysis of Android Adware using analysis tools such as *Apktool, dex2jar, and jd-gui* to obtain the source code. [Figure 2] is an example of extracting the signature from the code by performing static analysis. After Android 9.0 version, the application operation running in the background is registered and provided in Notification Server. , a signature such as a combination that periodically regenerates AD by giving a delay was generated. The dataset used adware verified by AMD (Android Malware Dataset) and Trendmicro, and as shown in [Figure 3], a total of 52 signatures were generated by statically analyzing 1,000 apps.

3.1 CFG generation using Soot

Control Flow Graph (CFG) is a graph composed of a basic block type, and each node is composed of a form corresponding to a statement[14]. CFG analyzes the source code, checks the path, and outputs it in graph form. It is composed of nodes and edges. A node corresponds to a statement, and an edge means an execution path. Existing CFG creates a graph depending on the entry point executed as main, and in the case of CFG researched recently, a graph is created by events and event handlers that occur when a user executes. Because Android generates a lot of event features such as screen touch, it is advantageous to apply CFG, which is being studied recently. The method of creating CFG using a tool such as FlowDroid is already widely used in CFG application. In this paper, CFG was created and applied in a form similar to FlowDroid based on Soot. Basically, in extracting the call relationship, DEX

(a) Airpush: 0c584509fc333367d1a30826c213c8fd.apk

```
1 public class BootReceiver extends BroadcastReceiver {
2     public BootReceiver() {
3         super();
4     }
5     public void onReceive(Context arg3, Intent arg4) {
6         arg3.startService(new Intent(arg3, NotificationController.class));
7     }
8 }
```

```
1 public class NotificationController extends Service {
2 …
3     this.notifTexts.add(new String("You can see lots of additional details!"));
4     this.notifTexts.add(new String("Check your battery\'s voltage!"));
5     this.notifTexts.add(new String("Wanna know your battery\'s voltage?"));
6     this.notifTexts.add(new String("You can see your battery\'s voltage!"));
7     this.notifTexts.add(new String("Wanna know your battery\'s temperature?"));
8 …
9 }
```

(b) Youmi: 0bf6bef249ade12b1bb04747ea6e1bfd.apk

```
1 public class AdReceiver extends BroadcastReceiver {
2 …
3     public final void onReceive(Context context, Intent intent) {
4         …
5         Intent intent2 = new Intent(context, AdService.class);
6         jVar.a(intent2);
7         context.startService(intent2);
8 …
9 }
```

(c) Adware Campaign: com.pirateshipboat.racing3d.simulator

```
1 public void setShowClose(boolean z, long j) {
2     this.showClose = Boolean.valueOf(z);
3     if (j >= 100 && j <= 7500) {
4         this.closeDelay = j;
5     }
6 }
```

Figure 2: Signature extraction example

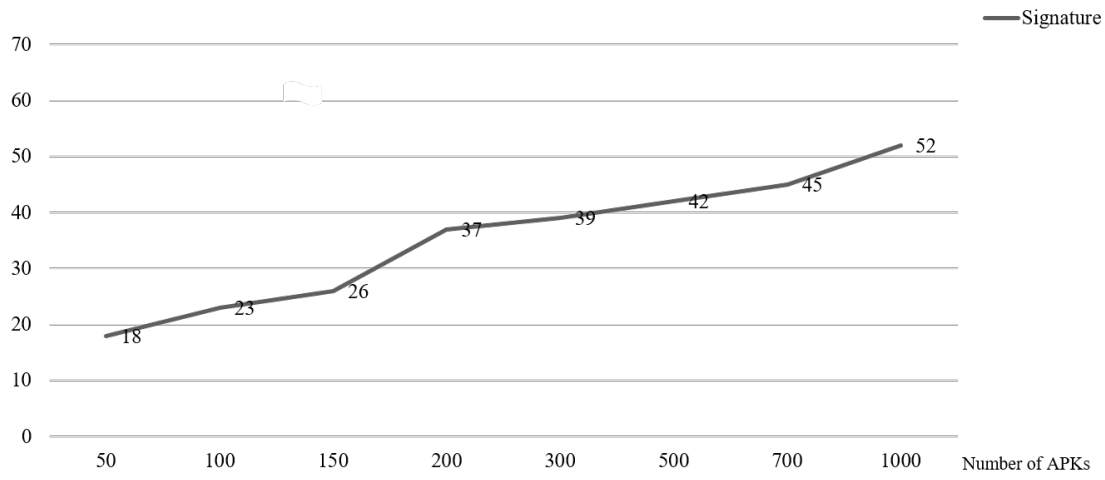
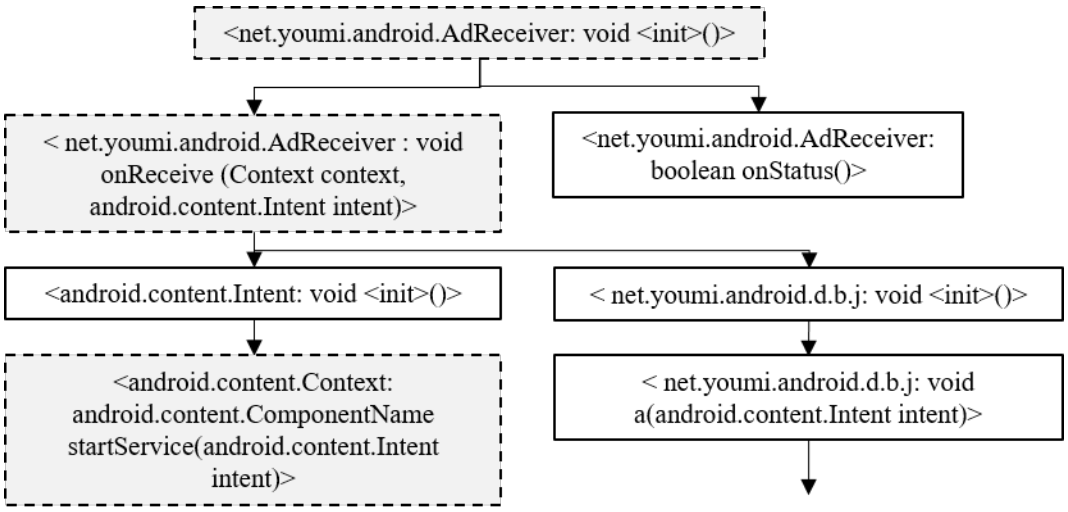


Figure 3: Number of signatures generated compared to APK file analysis

analysis is performed, and the DEX file is divided into String Type, Field, Proto, Method, and *Class_Defs* fields. The following process is performed to create the method call relationship of the class to be analyzed through DEX analysis. First, by accessing the Class_defs field, *class_data_off*, which identifies contents such as class name, access limiter, and parent class information, is analyzed. After that, the contents defined in *class_data_off* are brought to obtain member variables and method information, and the method is analyzed by bringing the list of *encoded_method_format*. Thereafter, by referring to the contents of the *Method_IDs* field, the type and number of parameters of the method, definition of the method return type, etc. are brought. After obtaining caller information by analyzing the Method_IDs field, the Dalvik bytecode command is parsed and call relationship is extracted. At this time, the offset value is extracted from the Method_IDs of the callee by analyzing the 2 bytes after the opcode of the Dalvik instruction, and the callee information is retrieved by referring to *Method_IDs* from the offset argument value to extract the call relationship between methods[15]. When CFG is generated in this form, it is completed in a connected form as shown in [Figure 4], and it is analyzed separately according



(a) Flow graph for violation criteria using CFG and signatures

```
00000036  new-instance        v1, j
0000003A  invoke-direct       j-><init>(String)V, v1, v0
00000040  new-instance        v0, Intent
00000044  const-class         v2, AdService
00000048  invoke-direct       Intent-><init>(Context, Class)V, v0, p1, v2
0000004E  invoke-virtual      j->a(Intent)V, v1, v0
00000054  invoke-virtual      Context->startService(Intent)ComponentName, p1, v0
```

(b) Decompiled result for violation criteria using CFG and signatures

Figure 4: Example of CFG creation

to the connection depth.

3.2 Adware behavior setting and analysis according to feature extraction

In order to extract the characteristics of Adware, 5490 applications classified as Adware were collected from AMD, TrendMicro, etc. as shown in [Figure 5]. And an initial data set was created by analyzing 1000 out of 5490 apps, extracting signatures and generating CFG. After that, five violations were set as shown in [Table 1]. The first is a signature-based detection that confirms that it is an already known adware package, and the second is a signature-based detection that checks if the user closes the advertisement and the next action is taken. The third is when it is connected to AlarmManager, which is to prepare for the case where a malicious application in the form of a Trojan is downloaded and operated after acting as a software notification manager, such as Adware known as Android.HiddenAds.728. The newly created application registers a broadcast receiver to receive system events and monitors when the device screen is turned on. Whenever the screen of an infected smartphone or tablet is activated, the Trojan will launch *com.google.android.gms.ads*. Use the *InterstitialAd* class to download and display ads. The fourth is a case of linking an advertisement to the outside, which means a case of connecting to an untrusted site. The fifth is an analysis by mixing CFG and signature to respond to the third operation by calling *BroadcastReceiver* with Android intent. did. For these five threat behaviors, the signature and CFG connection were applied to adware detection.

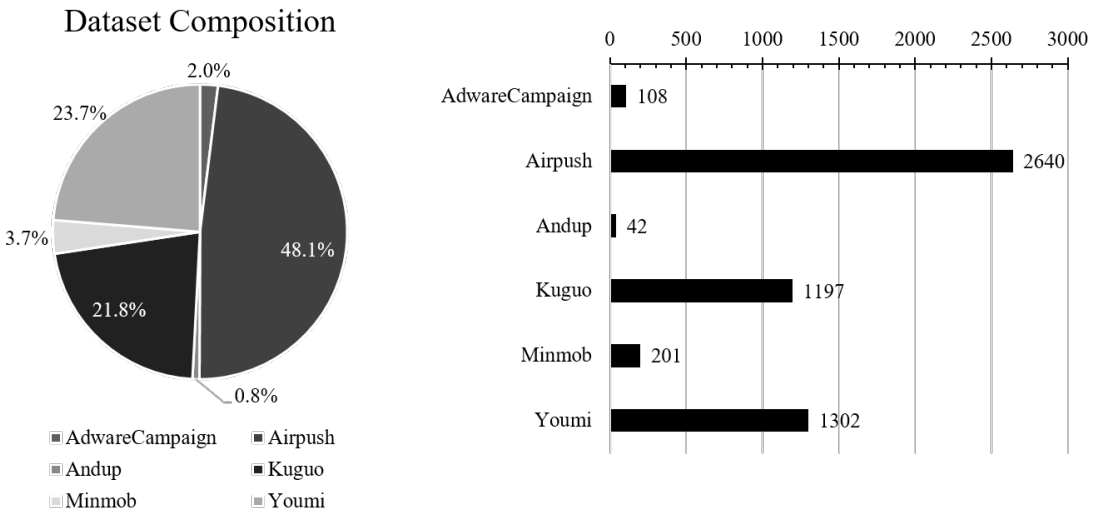


Figure 5: Adware analysis dataset

Table 1: To set up and detect adware violations

Violation No	Title	Description	Strategy
Violation 0	Known adware packages	Check for popular adware packages.	Signature based
Violation 1	Un-closable config	Ad should be able to be closed by user.	Signature based
Violation 2	AlarmManager accessing Notification	Ad related app should not access AlarmManager.	CFG based
Violation 3	Untrusted link to outside	Third-party ad app should not link their advertisement to outside link. (untrusted)	Signature based+CFG based
Violation 4	BroadcastReceiver to AdService	Ad-related service should not be started from BroadcastReceiver with android intent.	Signature based+CFG based

4 Evaluation

To evaluate the performance of the adware detection platform, the detection rate based on the first CFG and signature was measured. As shown in [Figure 6], the signature list and CFG were generated using Soot, and the data set was classified and analyzed according to the violation. The entire application was compared by performing tests while changing the CFG depth and the number of samples to be analyzed to extract the signature. Second, in order to detect the accuracy that may occur on the detection platform, samples of normal apps containing advertisements were compared with those obtained from the Google Play Store.

4.1 Malicious behavior detection rate by adware group

By changing the signature and CFG depth that were analyzed in advance, the detection rate of each group of Adware was checked. The depth of CFG was analyzed by dividing it into 3, 5, and 7, and the number of samples from which the signature was extracted was classified into 500 and 1000 and analyzed. First, in [Table 2], the depth of CFG is 3 and the number of samples extracted from the signature is 500. In this case, Adware Campaign and Andup were detected well, but among the 500 randomly selected samples, there were few Andup samples, so the detection rate was about 60%. The overall detection rate was 75.59%, which was a significantly lower detection rate compared to the previous study.

Second, in [Table 3], the depth of CFG is 3 and the number of samples extracted from signature is 1000. It can be seen that the overall detection rate increased as the sample of Kuguo, which was small when analyzed for the first time, was additionally added. In addition, most of the detection rates were increased in other samples. However, it can be seen that the detection rate is still lower than that of previous studies.

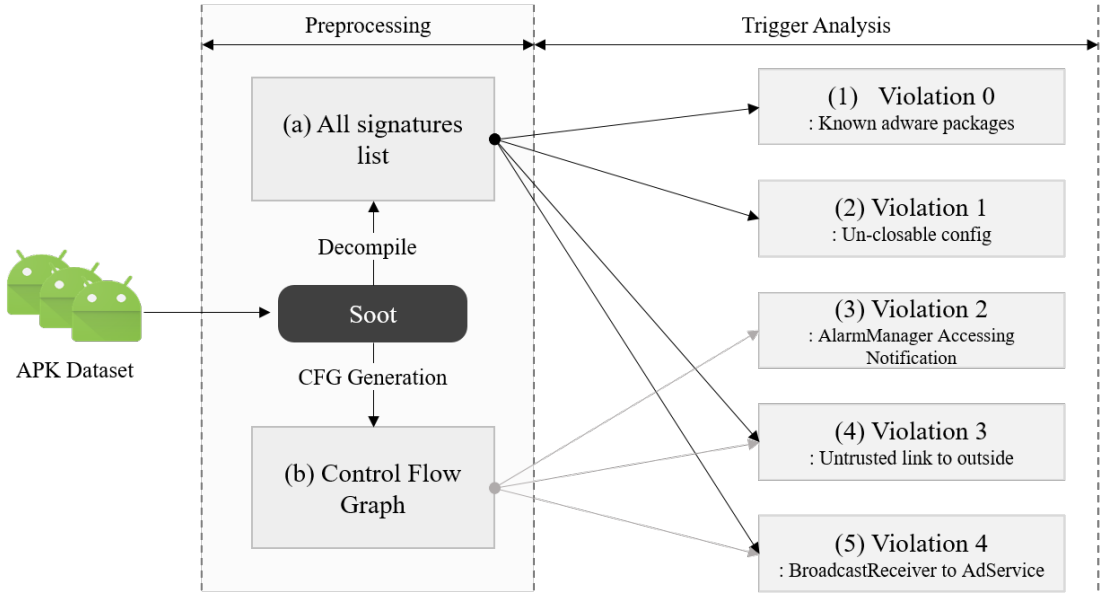


Figure 6: Adware behavior setting according to feature extraction

Table 2: Depth: 3, Number of Signature Extraction Samples: 500 Test Results

Adware Group	Total number of APK	Result		
		Detected more than 1 violation	Undetected	Detection rate (%)
Adware Campaign	108	106	2	98.15
Airpush	2640	2353	287	89.13
Andup	42	40	2	95.24
Kuguo	1197	744	453	60.89
Minmob	201	164	37	81.59
Youmi	1302	743	559	80.62
Total	5490	4150	1340	75.59

Table 3: Depth: 3, Number of Signature Extraction Samples: 1000 Test Results

Adware Group	Total number of APK	Result		
		Detected more than 1 violation	Undetected	Detection rate (%)
Adware Campaign	108	106	2	98.15
Airpush	2640	2353	287	89.13
Andup	42	40	2	95.24
Kuguo	1197	744	453	60.89
Minmob	201	164	37	81.59
Youmi	1302	743	559	80.62
Total	5490	4150	1340	75.59

Table 4: Depth: 5, Number of Signature Extraction Samples: 1000 Test Results

Adware Group	Total number of APK	Result		
		Detected more than 1 violation	Undetected	Detection rate (%)
Adware Campaign	108	106	2	98.15
Airpush	2640	2542	98	96.29
Andup	42	41	1	97.62
Kuguo	1197	1018	179	85.08
Minmob	201	173	28	86.07
Youmi	1302	1203	99	92.40
Total	5490	5083	407	92.59

Table 5: Depth: 7, Number of Signature Extraction Samples: 1000 Test Results

Adware Group	Total number of APK	Result		
		Detected more than 1 violation	Undetected	Detection rate (%)
Adware Campaign	108	97	11	89.81
Airpush	2640	2240	400	84.85
Andup	42	24	18	57.14
Kuguo	1197	759	438	63.41
Minmob	201	167	34	83.08
Youmi	1302	772	530	59.29
Total	5490	4059	1431	73.93

Third, [Table 4] shows the case where the depth of CFG is 5 and the number of samples extracted from the signature is 1000. This case showed the best detection rate, showing a detection rate of 92.59% overall, and it can be confirmed that most of the signatures and call relationships through CFG are properly formed. The most accurate efficiency during testing is the case shown.

Finally, [Table 5] shows the case where the depth of CFG is 7 and the number of samples extracted from the signature is 1000. In this case, it can be seen that the detection rate drops sharply even in Adware Campaign and Andup, which had good detection rates. This is because Adware performs similar types of behavior, but the final behavior in the actual Caller-Callee relationship cannot be the same, and when the depth is connected up to 7, different behaviors are not judged as the same, so the detection rate is rather low.

4.2 Adware detection rate analysis

In order to evaluate the performance of the adware detection platform, applications including AD collected from Google Play Store and actual malicious applications were randomly extracted and analyzed. In order to evaluate the model, it was classified into four methods. In the case of True Positive (TP), when the actual Adware is classified as Adware, In the case of False Positive (FP), when the general app is predicted as Adware, In the case of False Negative (FN), when the Adware is predicted as the general App , in the case of True Negative (TN), it is a case in which a general app is normally determined. For analysis of the detection rate of the analysis platform, it was classified into a test set and a training set. For the test set, 1000 samples out of 5490 adware were randomly selected and analyzed, and for the training set, 4490 samples were used excluding the samples. In the case of general apps, 1380 apps collected from the Google Play Store were selected as the training set and tested. The test results are shown in [Table 6] below. The rate of detecting adware as adware was 91.2%, and false positives for normal apps as malicious was 5.6%. The detection rate obtained through this is shown in [Table 7] below. Accuracy of about 91.92% was confirmed, which is similar to the results of other studies introduced

Table 6: Adware and general app analysis results

		Real Result	
		True	False
Evaluation Result	True	4097	81
	False	393	1299

Table 7: Classification performance evaluation result

	Precision	Recall	Accuracy	F1 score
Result	0.9806	0.9124	0.9192	0.9452

previously. However, if the results of the existing research are the result of the analyst's direct execution until the adware is dynamically generated, the proposed analysis method shows superior performance in that it can be analyzed automatically after signature and CFG generation.

5 Conclusion

In this paper, we propose the detection of recent stealthy Adware. CFG and signature were generated for the detection, and 6 adware sample groups were analyzed for 5 threat information. Through this, we confirmed the detection rate for each group of about 92.59%, and the accuracy of 91.92% was confirmed as a result of analysis compared to the actual normal app. Existing adware detection was limited in a way that the analyst had to stimulate the screen until Ads activates, the efficiency of this study is confirmed by showing a similar detection rate only with static analysis. Through this paper, a system to respond to Adware was constructed and evaluated. However, further research should be continued in that it is still difficult to respond to the operation of Adware through delayed attack (i.e., similar to logic bomb).

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the ITRC(Information Technology Research Center) support program(IITP-2022-0-01602) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation) and This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2020-0-00952, Development of 5G Edge Security Technology for Ensuring 5G+ Service Stability and Availability)

References

[1] V. Chebyshev. It threat evolution q3 2020 mobile statistic, April 2021. https://securelist.com/it-threat-evolution-q3-2020-mobile-statistics/99461/ [Online; Accessed on December 15, 2022].

[2] R. Srivastava el al. Android malware detection amid covid-19. In *Proc. of the 9th International Conference System Modeling and Advancement in Research Trends (SMART'20), Moradabad, India*, pages 74–78. IEEE, December 2020.

[3] V. Chebyshev. It threat evolution q1 2021. mobile statistics, September 2021. https://securelist.com/it-threat-evolution-q1-2021-mobile-statistics/102547/ [Online; Accessed on December 15, 2022].

[4] J. Huang. Distribution of free and paid android apps in the google play store as of august 2021, September 2021. https://www.statista.com/statistics/266211/distribution-of-free-and-paid-android-apps/ [Online; Accessed on December 15, 2022].

[5] Statista. Adware campaign identified from 182 game and camera apps on google play and third-party stores like 9apps, July 2019. https://malware.news/t/adware-campaign-identified-from-182-game-and-camera-apps-on-google-play-and-third-party-stores-like-9apps/30926 [Online; Accessed on December 15, 2022].

[6] D. Kim, S. Son, and V. Shmatikov. What mobile ads know about mobile users. In *Proc. of the 2016 Network and Distributed System Security Symposium (NDSS'16), San Diego, CA, USA*, pages 21–24. NDSS, February 2016.

[7] E. Erturk. A case study in open source software security and privacy: Android adware. In *Proc. of the 2012 World Congress on Internet Security (WorldCIS'12), Guelph, ON, Canada*, pages 189–191. IEEE, June 2012.

[8] T. Takahashi and T. Ban. *Android Application Analysis Using Machine Learning Techniques*. Springer, January 2019.

[9] A. Javaid, I. Rashid, H. Abbas, and M. Fugini. Ease or privacy? a comprehensive analysis of android embedded adware. In *Proc. of the 27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'18), Paris, France*, pages 254–260. IEEE, June 2018.

[10] S. Suresh, F.D. Troia, K. Potika, and M. Stamp. An analysis of android adware. *Journal of Computer Virology and Hacking Techniques*, 15:147–160, September 2019.

[11] K. Lee and H. Park. Malicious adware detection on android platform using dynamic random forest. In *Proc. of the 13th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'19), Sydney, Australia*, volume 994 of *Advances in Intelligent Systems and Computing*, pages 609–617. Springer, July 2019.

[12] A. Alzubaidi. Recent advances in android mobile malware detection: A systematic literature review. *IEEE Access*, 9:146318–146349, October 2021.

[13] O.S.A. Aboosh and O.A.I. Aldabbagh. Andoid adware detection model based on machine learning techniques. In *Proc. of the 2021 International Conference on Computing and Communications Applications and Technologies (I3CAT'21), Ipswich, United Kingdom*, pages 98–104. IEEE, September 2021.

[14] H. Shim and S. Jung. Icfgo : Ui concealing and dummy flow insertion method for inter-procedural control flow graph obfuscation. *Journal of the Korea Institute of Information Security & Cryptology*, 30(3):493–501, 2020.

[15] J. Ahn, J. Park, L. Nguyen-Vu, and S. Jung. Android application call relationship analysis based on dex and elf binary reverse engineering. *Journal of the Korea Institute of Information Security & Cryptology*, 29(1):45–55, 2019.

Author Biography

Jungsoo Park received the B.S and M.S. degree in Electronics Engineering from Soongsil University in 2013 and 2015, respectively, and the Ph.D. degree in software convergence from Soongsil Universtiy in 2021. From 2021, he has worked as an assistant professor at Busan University of Foreign Studies, Busan, South Korea. In 2022, he joined the School of Electronic Engineering, Soongsil University, Seoul, South Korea, where he currently serves as a Professor. His research interests include cloud security, mobile security, and Machine Learning.

Souhwan Jung received the B.S. and M.S. degrees in electronics engineering from Seoul National University, in 1985 and 1987, respectively, and the Ph.D. degree from the University of Washington, Seattle, WA, USA, in 1996. From 1996 to 1997, he was a Senior Software Engineer with Stellar One Corporation, Bellevue, USA. In 1997, he joined the School of Electronic Engineering, Soongsil University, Seoul, South Korea, where he currently serves as a Professor. He is an Executive Director of the Korea Institute of Information Security and Cryptology. He was also a Program Director of the Ministry of Knowledge Economy in Korea for information security area, from 2009 to 2011. He has led the Smart Security Service Research Center funded by the Korean Government for six years, since 2012. His current research interests include Android and Malware security, Cloud security, and the AI Security.