Harnessing Broadcast Receivers for Classification of Android Malware Threats

Nikolaos Chrysikos[1], Panagiotis Karampelas[1] and Konstantinos Xylogiannopoulos[2]
[1] Hellenic Air Force Academy, Dekeleia, Greece
[2] University of Calgary, Calgary, AB, Canada; Stetson University, DeLand, FL, USA

nikoschrysikos2000@gmail.com
panagiotis.karampelas@hafa.haf.gr
kostasfx@yahoo.gr

Abstract: With the increasing number of malicious attacks, the way how to detect and classify malicious apps has drawn attention in mobile technology market. In this paper, we proposed a classification model to seek and track malware Apps broadcast receivers in such devices. To identify the family of apps, static features of each app was extracted and a novel deterministic classifier is employed to categorize malware apps. With such, we can act against malware of known family, since we understand its functions, and prevent it from spreading out in larger scale, affecting extensively our society. Detailed description of the classification model is provided, as well the core technologies of this novel malicious android applications' model are presented. From experiments performed on a set of Android-based malware apps, we observe that the proposed classification model achieves highest accuracy, true-positive rate, false-positive rate, precision, recall, f-measure in comparison to other methods implemented in published experiments. The proposed classification model is promising since the average accuracy reaches an average of 97.31% and can effectively be applied to Android malware categorization, providing early detection of the capabilities of malware and the prospect of warning users of threatens ahead.

Keywords: Malware Classification, Android Malware Threats, Broadcast receivers, Malware APKs

1. Introduction

According to the latest statistics, as of October 2023 (DataReportal, 2023), there were 5.60 billion unique mobile users recorded globally, constituting two-thirds of the global population. This widespread ownership spans diverse age groups, educational backgrounds, and origins, integrating smartphones into daily life. Common usage patterns encompass activities such as audio and video calling, instant messaging, news reading, social media engagement, healthcare monitoring, smart-home controlling, gaming, photo/video capturing, navigation, music listening, emailing, financial transactions, and e-government service utilization (Xu, 2019). With this multitude of functions, smartphones become repositories for personal data, encompassing phone calls, messages, photos, videos, medical records, financial information, and more.

The abundance of personal information stored on smartphones renders them enticing targets for cybercriminals. Their toolkit encompasses various malicious applications, including AdWare, SpyWare, Downloaders, Bankbots, Ransomware, Backdoor tools, hack tools, etc. (securelist.com, 2023). These applications are adept at disguising themselves as legitimate mobile apps. However, once downloaded onto smartphones, they become weaponized, infiltrating users' personal data for illicit purposes. Given that the Android operating system holds a 70% share in the mobile phone market, cybercriminals have chosen it as their primary target. Consequently, Android smartphones face a likelihood that is 50 times greater of being infected with malware compared to iOS smartphones.

Hence, numerous security analysts strive to devise innovative methods for the detection and classification of mobile malware, employing diverse techniques. Current Android malware analysis methods broadly fall into three categories: static analysis, dynamic analysis, and a hybrid approach that merges aspects of both.

Static analysis concentrates on scrutinizing, reviewing, and comparing static information, such as program code, permissions, intents, and broadcast receivers extracted from a suspicious application, with known malware information to identify common patterns or signatures. However, static analysis is susceptible to obfuscation techniques that alter the code, rendering the detection of malicious payloads more challenging. Moreover, it is ineffective against zero-day attacks, as there are no pre-existing signatures for such malware, making them undetectable.

On the other hand, dynamic analysis observes runtime behaviors of the applications in real or virtual mobile environments, examining API calls, system calls, data transmission, hidden icon operations, and network traffic. Hybrid analysis combines the strengths of both dynamic and static analyses, achieving enhanced results by merging their outputs. Modern hybrid approaches also incorporate Deep Neural Networks (DNN), which appear

to yield better results by combining static and dynamic techniques, however their performance is not deterministic and thus not always reliable especially when a different dataset is tested.

Our approach is focused on the static analysis of one of the very important characteristics of an Android application which is broadcast receivers. Broadcast receivers () are used by applications to receive broadcast messages sent from the Android system or other Android applications, when an event occurs. For example, when the battery is almost depleted or the system boots up, broadcast messages will be sent to other applications, so actions will be taken. There are three types of broadcast intents according to Bigsin et al (2021), normal, sticky and ordered.

- Normal broadcasts: Normal broadcasts are sent simultaneously to all registered receivers, and then they cease to exist.
- Ordered broadcasts: Ordered broadcasts are delivered to one receiver and any receiver in the delivery chain can stop the propagation of the broadcasts.
- Sticky broadcasts: Eventually sticky broadcasts are accessible even after they have been sent and can be re-delivered to future receivers.

As can be understood, broadcast receivers pose a potential threat to smart devices, as they can receive messages from both the Android system and other applications (Google, 2024). In recent years, there has been growing interest in the research community regarding the exploitation of broadcast receivers. Several researchers have proposed various methods for utilizing broadcast receivers in malware detection (Mohsen et al., 2017; Tian, 2016; Bisgin et al, 2021).

In this field of study, our proposed methodology exploits the existence of broadcast receivers to perform Android malware classification. We claim that our methodology:

- Exclusively employs broadcast receivers as input data for analyzing and classifying malware applications, achieving exceptionally high accuracy. This constitutes a competitive advantage, considering that the majority of real-world malicious Android applications obfuscate most of their code. Therefore, an approach requiring additional features may perform poorly or not at all if some of the necessary features for the classification is obfuscated or missing.
- Has the potential to outperform machine learning techniques, as these methods rely on probabilistic metrics, whereas our method represents a deterministic classification approach.

The rest of the paper is organized as follows: The related work section, presents relevant papers that utilize features such as broadcast receivers or permissions in malware classification. Section 3, describes in details the proposed methodology while section 4 presents the dataset used and the conducted experiments to verify the effectiveness of the proposed methodology. Section 5 presents the results and discusses the limitations of the methodology. Finally, the conclusions and tentative future work are presented.

2. Related Work

There have been a great number of studies on detecting malicious Android applications based on technical information such as broadcast receivers, permissions or any other information included in the AndroidManifest.xml file. Through the analysis of such information, the researchers have developed several methodologies which attempt to detect whether a mobile application is malicious or not. A variety of techniques have been used for that purpose utilising permissions, receivers and other features integrated in mobile applications. The analysis of those features have been attempted to be done with a wide range of methods including data mining techniques and diverse machine learning approaches.

Lin et al. (2013) proposed SCSdroid, utilizing thread-grained system call sequences for detection accuracy, achieving a high rate of 95.97%. However, there are case in which some benign families have almost the same sequences as malware families and thus the classification is not always accurate. Another limitation of this methodology according to the authors is that it requires a balanced training with malicious and benign applications in order to perform well.

Mohsen and Shehab (2016) investigated broadcast receiver usage patterns, achieving a robust 97% malware prediction accuracy by combining permissions and receivers. The same researcher with a new team investigated broadcast receiver usage patterns in Android applications to detect malware. They proposed a data mining malware detection mechanism based on statically registered broadcast receivers, achieving a malware

prediction accuracy of 97%. Their findings revealed that malicious apps intensively use receivers compared to benign ones, making this aspect a valuable indicator for classification (Mohsen et al., 2017).

Massarelli et al. (2020) introduced AndroDFA, (detrended fluctuation analysis) for extracting features from malicious applications. They utilized Support Vector Machine (SVM) for classification, achieving an accuracy of 82%. Notably, their approach can be used on physical devices rather than exclusively on emulators, addressing the challenge of modern mobile malware detecting emulated environments and hiding malicious behavior.

In another approach, Fang et al. (2020) suggested an Android malware classification method based on DEX files, achieving a precision, recall, and F1 score of about 0.96. The authors highlighted the effectiveness of their method, reducing future extraction time by 2.999 seconds compared to alternative approaches. In the specific technique that selected features are converted into a picture and to text and then a fusion algorithm is used for the classification.

Milosevic et al. (2017) introduced a machine learning-aided static analysis technique, achieving a notable 95.1% F-score for source code-based and 89% for permission-based classification. It is noteworthy, that the authors compared a permission-based classification method with a permission-based clustering approach and the result was that the first method outperforms the clustering-based approach. Thus, they conclude that the permission clustering-based approach cannot be used in malware detection since benign application may use the same permissions and thus an agnostic approach does not help in this case. Similarly, since broadcast receivers have legitimate use in the Android applications, a clustering approach will not have good results.

Ngamwitroj and Limthanmaphon (2018) generated signatures using both permissions and broadcast receivers to identify malware applications. They employed a dataset comprising 800 applications to create these signatures, subsequently utilizing the same dataset to detect malicious applications with a success rate of 86.56%.

An additional notable effort to leverage inter-component communication mechanisms, specifically broadcast receivers, is the ICCDetector (Xu et al., 2016). This tool utilizes various application components such as Activity, Service, Broadcast Receiver, and Content Provider, as well as intents and intent filters, to categorize Android applications as either benign or malicious. Employing a dataset comprising 5,264 malware applications and 12,026 benign applications, the conducted experiments demonstrated an impressive accuracy of 97.4%.

Another similar approach is named AppoScopy, which introduces a signature-based specification language for mapping the control and data flow within Android applications. This method specifically records broadcast receivers and the associated data linked to these receivers. By employing the proposed specification language, researchers generate signatures for the detection of malware applications (Feng et al., 2014). The effectiveness of this methodology was assessed using 1,027 malware samples, resulting in an accuracy of 90%.

In contrast with the above-mentioned methods, that combine receivers and other features such as permissions, opcodes system calls and actions which then are processed by machine learning algorithms, our work focuses on the broadcast receivers alone in a static deterministic approach, out of all features. Thus, our methodology classifies a malicious mobile application to the related malware family analysing the broadcast receivers achieving a 97.3% accuracy rate outperforming all other methods of similar approach.

3. Classification Framework

3.1 Problem Statement

The algorithms developed by researchers mentioned in related work section implement a wide range of methods with varying data inputs collected with static analysis such as opcodes extracted from .smali files, permissions , receivers, intents extracted from AndroidManifest.xml file and with dynamic analysis such as api dependency graphs and system calls. All these algorithms may follow a different approach of data collection and processing, but on the final step of decision making and classification, they share one common characteristic, they use probabilistic classification algorithm (either Machine Learning or some Neural Network model). The main weakness of these algorithms, which we aim to combat with our heuristic approach, is that their output is always prone to statistical error. Our method is based on a deterministic categorization model that achieves the highest accuracy compared to mentioned work.

We therefore consider a classification problem in which as input is used a malicious smartphone application which should be classified into one of the known categories of malware. To solve this problem, a classification

method is proposed that uses the application broadcast receivers which are then compared with the corresponding receivers sets of each category of malware.

3.2 Methodology

The core idea behind this methodology is classification of malicious applications (APKs) by similarity. In details, that means we extract specific features from applications of pre-classified data, and we process them in order to create unique sets of these features for each family. Then we extract the same data from an unknown sample, and we compare it with these sets. The sample is categorized to the family that is more similar with (Fig. 1).

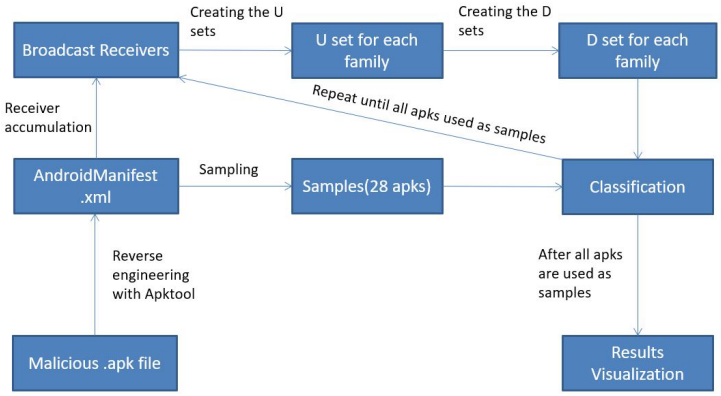

Figure 1: Flow diagram depicting the methodology used

The overall methodology spans in six stages:

3.2.1 Reverse Engineering

In order to obtain the data needed and used for the classification process, we initially have to extract the AndroidManifest.xml files out from the applications. In order to achieve that, we use a decoding tool for Android applications, named Apktool[1]. Apktool is a tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications. It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc.

Its main features are:

- Disassembling resources to nearly original form (including resources.arsc, classes.dex, .png files and XMLs)
- Rebuilding decoded resources back to binary APK/JAR
- Organizing and handling APKs that depend on framework resources
- Smali Debugging
- Helping with repetitive tasks

In our methodology, we are interested in XML files since the broadcast receivers are included in AndroidManifest.xml. In the AndroidManifest.xml file, broadcast receivers are defined within the <receiver> element, each linked to an <intent-filter> element. These intent filters play a pivotal role in dictating the circumstances under which a receiver is activated. Intent filters encompass a spectrum of criteria such as actions, categories, and data types, outlining the specific broadcasts to which a receiver should respond. This mechanism allows control over the events that trigger the receiver's functionality. However, the flexibility of intent filters also introduces a potential vulnerability. Malicious applications may strategically exploit specific intent filters to intercept sensitive broadcasts or manipulate communication channels between components. The careful analysis of intent filters is, therefore, imperative in identifying and mitigating potential security risks associated with broadcast receivers in the Android ecosystem. In the following table, we have included some representative instances of broadcast receivers that we can extracted during our analysis:

[1] https://ibotpeaches.github.io/Apktool/

Table 1: Examples of Broadcast Receivers found during receiver accumulation stage.

Family	Apk's hash	Broadcast Receiver
AndroRAT	3f1fe2984a0573fa0e466c0c6b81e856	BootReceiver
BankBot	d55243c458f1da54a2473714f5ab4de8	org.starsizew.Aa
Boqx	0c2f86b8c066cd4f8573b115b19b578c	com.cdfg.ad.poster.ReceiverAlarm
Dowgin	0a1bead476e2de6e2c94f000ca1aab84	AppReceiver
DroidKungFu	b39667306cf1ec7a52b689ca5214a1f4	com.cdjm.reader.control.Receiver
FakeAV	049241403fdbc4cabd880aabddd69cf0	DefenderAppWidgetProvider
Fjcon	73f75636d00c766c6096e6051e474a76	InstallShortcutReceiver
GingerMaster	519bab12b8d846e4a599d1f25ffb0e49	receiver.ShutdownReceiver
Mecor	0abd91480a05102fa8ff8d915e7e6584	com.google.android.gcm.GCMBroadcastReceiver
RuMMS	0dd1d8d348a3de7ed419da54ae878d37	com.nihopgezk.azszafsi.qjsaenabjy
SimpleLocker	58bbfc398808946b482f587b27d7af0e	a.b.c.d.Event
Ztorg	54938db23973a525d1c427df86f3ef6b	dl.ph.yy.mf.BootReceiver

3.2.2 Receiver Accumulation

In order to gather the features needed, we extract the broadcast receivers contained in the XML files and we store them in text files, for every apk. Thus, the output of this stage is a list of files containing information about the broadcast receivers associated with each application. Specifically, each text file is named after its corresponding application, ensuring easy reference. These files serve as valuable repositories, containing the extracted names of broadcast receivers for subsequent stages of analysis. To enhance systematic organization, all these files are arranged within folders, designated with the names of the respective APK families.

3.2.3 Sampling

In this stage, a systematic recurring approach is adopted to create a representative sample for analysis. Initially, one application is linearly selected from the queue of applications of each family, serving as a sample for later evaluation. Following this step, the testing dataset, with cardinality 28, is created, comprising the selected applications. Subsequently, we proceed to create the Union (U) and Difference (D) sets as described in section 3.2.4 and 3.2.5 and then classify the testing dataset.

The stages from the sampling to the classifying are performed repeatedly, until every apk has been utilized at least once as a sample in the testing dataset, following a cyclic process. To elaborate, consider a dataset with the largest family of 3,000 apps. The process, in order to sample every application of the family, needs to iterate 3,000 times. In this way, for the largest family every application is selected once, while for smaller families we have to calculate the iteration ratio of the family with the largest one. For example, if a family has 10 apks and the largest one has 3,000 apks then the iteration ratio is 300 and the first apk of the smaller family will be used as sample for the first 300 testing sets, the second apk will be used for the next 300 testing sets and so on.

Following the above described process, for the experimental dataset of section 4.1 with the completion of all iterations, an average score of the results for all iterations of the model for each family is calculated and presented. In this case, the largest family is Dowgin with 3,371 apks, so our model will repeat steps 3.2.3 to 3.2.6 3,371 times. This approach aims to exhaustively test the model's integrity by employing every apk as a sample. This is crucial since most families display great variance of different receivers. Random sampling under these circumstances, proves inadequate in forming a representative testing set for each family.

3.2.4 Creating the U set

For all families, we find their unique receivers that each application uses and then we store all these receivers in a text file. In terms of set theory, we calculate the union of the subsets, where subset is considered each

application receivers, for example the subsets A, B and C, as it is illustrated in Figure 2. The output process is the Union (U) set of all unique broadcast receivers from all the applications in the same family.

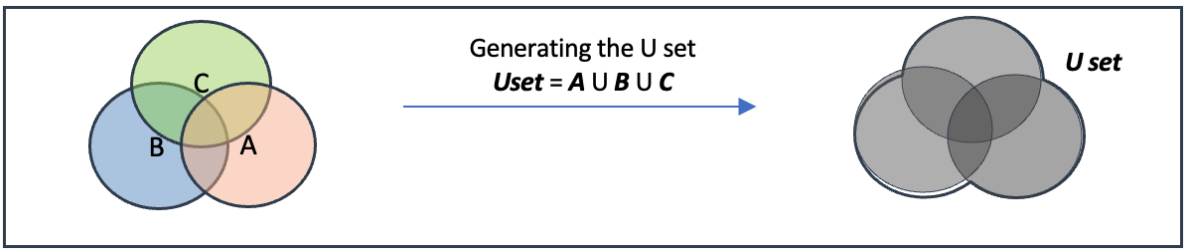

Figure 2: A visual representation of applications' A B and C receivers Venn Diagram as sets and the computation of U set.

3.2.5 Creating the D set

For the U sets for every family, we detect the broadcast receivers that are common in at least two families, and we remove them. Continuing the aforementioned example, in Figure 3, for the U set of A B and C subsets, we calculate the difference of A with B and C, of B with A and C and of C with A and B and then we calculate the union of all differences as the D set.

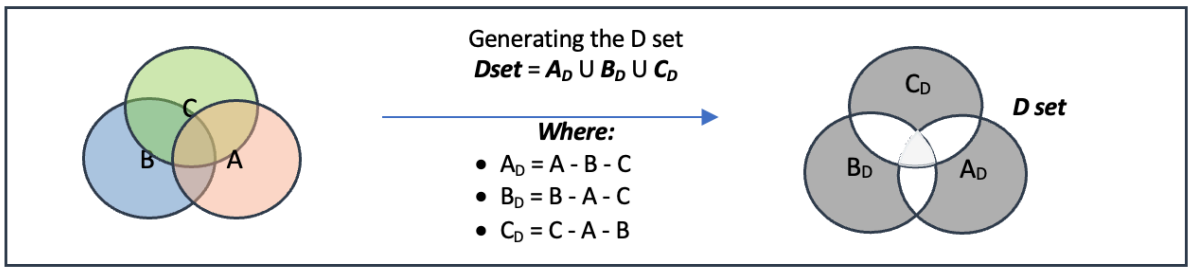

Figure 3: A visual representation of applications' A B and C receivers using Venn Diagram as sets and the computation of D set.

3.2.6 Classification

During the classification phase, the D sets, which are sets of broadcast receivers created in the prior step, are utilized. The heuristic model employed for classification involves comparing the similarity between an application's receivers and the D sets. Thus, we compare the D sets with the current sample group, where each sample represents an apk. The process involves taking the receivers of each application in the sample group and comparing them with the receivers in each D set belonging to different families. For every shared receiver detected between the sample and a D set, we increment the similarity score of the sample with that specific family. The similarity score is calculated as the total number of common receivers divided by the number of receivers in the corresponding D set.

Let's illustrate this with an example: Consider a sample application with 5 receivers. If it shares 3 receivers with a D set from a family having 200 receivers, the similarity score with that family would be 3/200. On the other hand, if the same application has 1 common receiver with a D set from a family with 18 receivers, the similarity score would be 1/18. In this case, the application would be categorized into the second family since 3/200 is less than 1/18.

It's crucial for the success of this methodology that broadcast receivers are unique to their respective families and are not detected in applications from other families. To achieve this, the D sets were crafted, ensuring the reliability of the classification process and minimizing the risk of misjudgements and errors.

4. Experiments

4.1 Dataset and Hardware Description

The dataset originally consists of 24,650 malware samples categorized in 71 families (Wei et al, 2017). A segment of it has been used for the experiment. More specifically, out of 32 families in the original dataset, only 29 use broadcast receivers and one of them, Fobus, has only two malicious samples whose AndroidManifest.xml contain receivers, and thus it is exempted from analysis and classification. In addition to that, it should be noted that not all samples of the 28 families use receivers which means that the actual number of samples that are going to be used is less than the samples that a family contains.

Table 1 provides an overview of the malware families in the part of the dataset we used. For each family, we show its type indicating the main purpose of the family. In total, these 28 families contain 14,288 samples but as mentioned, not all of them have broadcast receivers. Specifically, 10,693 malicious applications are used, which is 68.3% of the original dataset.

Table 2: Overview of the malware families of the dataset

Families	Type	# of Varieties	# of Samples	# of Samples Used	Used Percentage
AndroRat	Backdoor	1	46	45	97.8
Andup	Adware	1	45	43	95.5
Aples	Ransom	1	21	21	100
BankBot	Trojan-Banker	8	740	646	87.3
Bankun	Trojan-Banker	4	70	54	77.1
Boqx	Trojan-Dropper	2	215	213	99.1
Cova	Trojan-SMS	1	17	4	23.5
Dowgin	Adware	1	3385	3371	99.6
DroidKungFu	Backdoor	6	546	486	89.0
FakeAngry	Backdoor	2	10	10	100
FakeAV	Trojan	1	5	5	100
FakeDoc	Trojan	1	21	21	100
FakeInst	Trojan-SMS	5	2172	1997	91.9
FakePlayer	Trojan-SMS	1	21	5	23.8
Finspy	Trojan-SMS	1	9	9	100
Fjcon	Backdoor	1	16	16	100
Fusob	Ransom	2	1277	1270	99.4
GingerMaster	Backdoor	7	128	117	91.4
GoldDream	Backdoor	2	53	53	100
Koler	Ransom	1	69	4	5.8
Kuguo	Adware	1	1199	10	0.8
Lotoor	HackerTool	3	333	15	4.5
Mecor	Trojan-Spy	1	1820	1820	100
RuMMS	Trojan-SMS	2	402	350	87.1
SimpleLocker	Ransom	1	173	11	6.4
SlemBunk	Trojan-Banker	3	174	18	10.3
Youmi	Adware	1	1301	76	5.8
Ztorg	Trojan-Dropper	1	20	3	15
Total / Average	-	62	14288	10693	68.3

4.2 Experimental Environment

Both analysis and classification were performed in a laptop that utilizes an AMD Ryzen 7 3750 H with Radeon Vega Mobile Gfx processor. The computing system was equipped with 16 Gb of Memory. The overall methodology lasted about 4 hours starting by the extraction of the receivers from each application, processing all the applications, classifying them, and repeating the same process for all samples and producing the final results. The whole process utilized, on average, 70% of the processing power of the processor and 8 Gb of memory to store the temporary data.

5. Results

This section summarizes the results of the methodology that was applied in the previously described dataset. In Figure 4, it can be seen the accuracy achieved and the number of applications per family that were classified along (True Positives). Light grey bars indicate lower True Positive Ratio while dark grey bars indicate high TPR For example, our methodology accurately classifies 2 applications of FakeAngry family, that results in 100% accuracy, but as the above table implies, there are 8 more FakeAngry applications in the dataset, that not classified in the correct family, or they were unclassified. On the other hand, in Mecor we can see that the algorithm achieves 100% True Positive Ratio, which means all Mecor applications were correctly classified to the family, but we have 85.85% accuracy, that indicates 14.15% of applications were incorrectly categorized as Mecor.

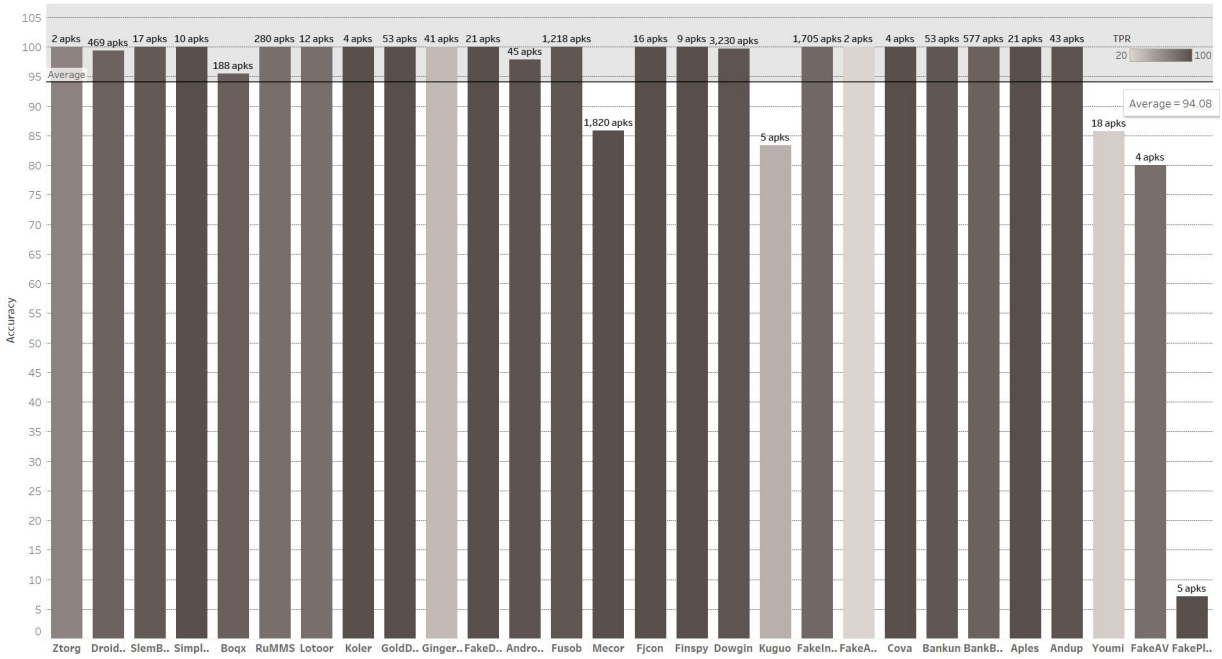

Figure 4: The results of the proposed methodology in the test dataset

Table 3: TP vs. Accuracy

Families	Total Predictions	True Positive Rate	True Positives	Accuracy
AndroRat	46	98	45	97.83
Andup	43	98	43	100
Aples	21	100	21	100
BankBot	577	89	577	100
Bankun	53	96	53	100
Boqx	197	90	188	95.43
Cova	4	100	4	100
Dowgin	3,239	97	3230	99.72

Families	Total Predictions	True Positive Rate	True Positives	Accuracy
DroidKungFu	472	88	469	99.36
FakeAngry	2	20	2	100
FakeAV	5	80	4	80
FakeDoc	21	100	21	100
FakeInst	1,705	85	1705	100
FakePlayer	70	100	5	7.14
Finspy	9	100	9	100
Fjcon	16	100	16	100
Fusob	1,218	96	1218	100
GingerMaster	41	35	41	100
GoldDream	53	100	53	100
Koler	4	100	4	100
Kuguo	6	40	5	83.33
Lotoor	12	80	12	100
Mecor	2,120	100	1820	85.85
RuMMS	280	80	280	100
SimpleLocker	10	100	10	100
SlemBunk	17	94	17	100
Youmi	21	24	18	85.71
Ztorg	2	67	2	100
Average	-	84.17	-	94.08
Unclassified	483	15.83	-	-

Table 3 presents the overview of the malware families of the dataset used for the proposed methodology that provides the number of apks that each family has, and the number of apks actually used for our algorithm.

The experiment's findings yield valuable insights. On average, 84.17% of APKs were successfully classified into their respective families. Notably, families with diverse receivers exhibit a lower classification percentage. Importantly, the overall false positive ratio is minimal, standing at 5.92%. Upon reviewing the table above, it becomes apparent that certain malicious families, namely FakeAngry, GingerMaster, Kuguo, and Youmi, exhibit a low positive rate. This indicates that numerous applications from these families were not accurately classified into their respective categories. However, intriguingly, the model demonstrates high accuracy for these families. This suggests that the applications categorized within FakeAngry, GingerMaster, Kuguo, and Youmi are indeed correctly placed despite the challenges posed by the lower positive rates. An interesting observation arises when excluding the FakePlayer family from the calculations, resulting in a significant improvement in the overall methodology's accuracy, reaching an average of 97.31%. Specifically, the accuracy for the FakePlayer family is merely 7.14%. This implies that among the 70 APKs in the FakePlayer family, only 5 genuinely belong to FakePlayer, while 64 belong to DroidKungFu Variety 2, and 1 to Dowgin. This misclassification issue extends to DroidKungFu applications utilizing the "Receiver" receiver common to both families. Despite the shared receiver, misclassifications occur due to the FakePlayer family having only 2 unique receivers compared to DroidKungFu's 67. Consequently, the similarity based on receivers is higher for FakePlayer (1 out of 2) and significantly lower for DroidKungFu (1 out of 67). This disparity increases the likelihood of an APK belonging to the FakePlayer family rather than other families based on the methodology's functioning. In conclusion, these results underscore the efficiency and effectiveness of the proposed methodology, particularly when dealing with an unbalanced dataset, as exemplified in our experiments.

However, it's crucial to note that not all families encompass receivers; out of 32 families, only 28 include them. Furthermore, not every APK within these families necessarily incorporates receivers. The classification process introduces a potential challenge, as a legitimate application might be erroneously categorized within a family due to shared receivers with a malicious counterpart, leading to the assumption of malware presence. This classification hinges on the concept of similarity, emphasizing that the fewer unique receivers a family possesses,

the greater the likelihood that an APK using one of these receivers may be classified into that family, irrespective of its actual association.

6. Conclusions

In the realm of mobile security, addressing the challenge of detecting malicious attacks has become increasingly critical, especially with the growing sophistication of such threats. The urgency to categorize malicious apps into families based on their functionality has led to the widespread use of machine learning algorithms for this purpose. However, these algorithms are susceptible to errors due to their reliance on probabilistic methods. In this study, we mitigate these errors by employing a deterministic classification method.

In contrast to more complex hybrid classification algorithms that leverage extensive features from static and dynamic analyses, our proposed method takes a unique approach. Since the proposed methodology uses a deterministic approach, the classification of the malicious APKs is realized with high certainty based on the receivers they utilize. The observed accuracy is 97.31% if algorithm's accuracy for the FakePlayer family is considered as an outlier, due to specific APKs in the dataset possessing receivers commonly associated with families they do not belong to. Upon comparing the outcomes of our approach with those of the algorithms discussed in the related work section, it becomes evident that our methodology attains the highest accuracy score. Despite this limitation, the underlying logic of our algorithm serves as a novel contribution to the malware classification problem. This deterministic approach can be adopted and utilized as input or a foundation for a malware classification model. Moreover, it offers valuable guidance to fellow researchers exploring implementations with different features and possibly across various operating systems.

References

Bisgin, H., Mohsen, F., Nwobodo, V. and Havens, R., 2021. Enhancing malware detection in Android application by incorporating broadcast receivers. *International Journal of Information Privacy, Security and Integrity*, *5*(1), pp.36-68.

DataReportal, 2023. Digital around the World. [online] DataReportal – Global Digital Insights. Available at: https://datareportal.com/global-digital-overview (Accessed: 15 January 2024).

Fang, Y., Gao, Y., Jing, F.A.N. and Zhang, L.E.I., 2020. Android malware familial classification based on dex file section features. *IEEE Access*, *8*, pp.10614-10627.

Feng, Y., Anand, S., Dillig, I. and Aiken, A., 2014, November. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering* (pp. 576-587).

Google, 2024. Broadcasts overview. Available at: https://developer.android.com/develop/background-work/background-tasks/broadcasts (Accessed: 15 January 2024).

Lin, Y.D., Lai, Y.C., Chen, C.H. and Tsai, H.C., 2013. Identifying android malicious repackaged applications by thread-grained system call sequences. *computers & security*, *39*, pp.340-350.

Massarelli, L., Aniello, L., Ciccotelli, C., Querzoni, L., Ucci, D. and Baldoni, R., 2020. Androdfa: android malware classification based on resource consumption. *Information*, *11*(6), p.326.

Milosevic, N., Dehghantanha, A. and Choo, K.K.R., 2017. Machine learning aided Android malware classification. *Computers & Electrical Engineering*, *61*, pp.266-274.

Mohsen, F. and Shehab, M., 2016, November. The listening patterns to system events by benign and malicious android apps. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)* (pp. 546-553). IEEE.

Mohsen, F., Bisgin, H., Scott, Z. and Strait, K., 2017, October. Detecting Android malwares by mining statically registered broadcast receivers. In *2017 IEEE 3rd International Conference on Collaboration and Internet Computing (CIC)* (pp. 67-76). IEEE.

Ngamwitroj, S. and Limthanmaphon, B., 2018, February. Adaptive Android malware signature detection. In *Proceedings of the 2018 International Conference on Communication Engineering and Technology* (pp. 22-25).

securelist.com, 2023. Mobile malware statistics, Q3 2023. [online] Available at: https://securelist.com/it-threat-evolution-q3-2023-mobile-statistics/ (Accessed: 15 January 2024).

Tian, D., 2016. *Detecting vulnerabilities of broadcast receivers in Android applications*. University of Ontario Institute of Technology (Canada).

Wei, F., Li, Y., Roy, S., Ou, X. and Zhou, W., 2017, July. Deep ground truth analysis of current android malware. In *International conference on detection of intrusions and malware, and vulnerability assessment* (pp. 252-276). Springer, Cham.

Xu, K., Li, Y. and Deng, R.H., 2016. Iccdetector: Icc-based malware detection on android. IEEE Transactions on Information Forensics and Security, 11(6), pp.1252-1264.

Xu, X. ed., 2019. Impacts of mobile use and experience on contemporary society. IGI Global.