Towards Demystifying Android Adware: Dataset and Payload Location

Chao Wang*‡
chaowang_@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Tianming Liu*
Tianming.Liu@monash.edu
Monash University
Clayton, Australia

Yanjie Zhao‡
yanjie_zhao@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

Lin Zhang†
zhanglin@cert.org.cn
The National Computer Emergency Response Team/Coordination Center of China (CNCERT/CC)
Beijing, China

Xiaoning Du
Xiaoning.Du@monash.edu
Monash University
Clayton, Australia

Li Li
lilicoding@ieee.org
Beihang University
Beijing, China

Haoyu Wang†‡
haoyuwang@hust.edu.cn
Huazhong University of Science and Technology
Wuhan, China

ABSTRACT

Adware represents a pervasive threat in the mobile ecosystem, yet its inherent characteristics have been largely overlooked by previous research. This work takes a crucial step towards demystifying Android adware. We present a comprehensive, well-annotated Android adware dataset **AdwareZoo**, which comprises **15,996** adware samples across **118** distinct adware families collected from both security reports and app repositories, providing a robust foundation for in-depth analysis and future research. We identify adware family payloads by isolating packages from adware samples for VirusTotal rescanning. Our analysis unveils critical insights into the adware ecosystem, highlighting distinctive patterns in family naming conventions and exposing the unexpected classification of widely-used ad networks as adware. We identify diverse payload location strategies, with a notable finding that over 30% of adware families employ payloads beyond conventional Java/Kotlin code. Furthermore, we reveal several evasion techniques utilized by adware, including package name obfuscation and dynamic payload loading. This research not only offers a robust foundation for understanding and combating adware but also highlights the pressing need for increased scrutiny of mobile advertising practices.

KEYWORDS

Android Adware, Mobile Advertising, Android Malware Dataset

*Chao Wang and Tianming Liu are the co-first authors.
†Haoyu Wang and Lin Zhang are the corresponding authors.
‡The full name of the authors' affiliation is Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology.

ASEW '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1249-4/24/10
https://doi.org/10.1145/3691621.3694948

ACM Reference Format:
Chao Wang, Tianming Liu, Yanjie Zhao, Lin Zhang, Xiaoning Du, Li Li, and Haoyu Wang. 2024. Towards Demystifying Android Adware: Dataset and Payload Location. In *39th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3691621.3694948

1 INTRODUCTION

In the domain of cybersecurity, adware is generally defined as software that displays unwanted advertisements [14]. Commonly categorized as greyware [15] or potentially unwanted applications (PUAs) [16], adware occupies an ambiguous space between benign software and overtly malicious programs like viruses and trojans.

As advertising has become a prevalent revenue model on mobile platforms, mobile adware has emerged as a pervasive threat. A recent report on the mobile malware landscape by Kaspersky indicates that over 40% of detected mobile malware are adware [12]. Beyond displaying aggressive, unwanted advertisements, mobile adware is known to exhibit various threat-posing behaviors, including stealing privacy information [1], propagating viruses and trojans [3], and committing ad fraud [2].

Despite its ubiquity and multifaceted risks, mobile adware has received disproportionately **less attention** from the research community compared to common mobile malware. Existing research on mobile adware has generally focused on adware identification [17, 30] and addressing whether adware should be considered as malware [23]. Unlike with general Android malware, no previous work has concentrated on demystifying the characteristics of adware. This gap partly originates from **the lack of a dedicated, well-annotated adware dataset**. While mobile malware analysis has made significant strides, with reliable and comprehensive Android malware datasets available such as MalGenome [38], Drebin [18], AMD [33], Rmvdroid [31], and Malradar [32], these resources have comprehensively elucidated Android malware in terms of their characteristics, behaviors, and implementation details on a family basis, which have provided valuable insights for

combating malware. However, there has not been such demystifying work or a dedicated dataset available for adware. The general malware datasets mentioned above contain only a limited number of adware samples that have not been specifically annotated, meaning we neither know which malware samples are adware nor can we identify the specific payload for a given adware.

The location of the payload is crucial information, which offers vital insights into the implementation strategies of adware and facilitates adware detection approaches. Moreover, it serves as a starting point for analysis in demystifying adware. The lack of specific adware-focused research and well-annotated datasets presents a concerning gap in the landscape of mobile security, potentially hindering the formation of effective countermeasures and leaving users and systems vulnerable to evolving adware threats.

To address these limitations and fill the existing research gap, our work takes a crucial step towards demystifying Android adware by constructing a novel, comprehensive adware dataset (§2.1) and identifying adware family payloads (§2.2) within the dataset. Our dataset construction process involves collecting samples from both security reports and app repositories, implementing adware filtering, and classifying samples into distinct adware families using AVClass [29]. We then identify adware family payloads by isolating packages from adware samples for VirusTotal rescanning, while employing feature hashing to overcome obfuscation challenges. This comprehensive methodology resulted in a well-annotated adware dataset **AdwareZoo**, which comprises **15,996** adware APKs across **118** distinct adware families (with the top 20 families demonstrated in Table 1), providing a robust foundation for in-depth adware analysis and future research.

Furthermore, our analysis revealed significant insights into adware ecosystems. For instance, we uncovered patterns in adware family naming conventions and identified 10 widely-used top ad networks classified as adware, highlighting the pressing need to demystify adware characteristics for better understanding and mitigation of their security threats. We revealed diverse adware payload location strategies, with over 30% adware families using payloads other than traditional Java/Kotlin code. We also identified and characterized several key evasion techniques employed by adware, including package name obfuscation, hiding under trusted package names, utilizing SO library payloads, and dynamic loading of payloads. These findings offer valuable insights for future adware detection and analysis efforts, contributing to the field of mobile security.

In summary, our work makes the following key contributions:

- We present a comprehensive, well-annotated Android adware dataset **AdwareZoo**, comprising **15,996** adware APKs across **118** distinct adware families. We locate the payload for each adware family, paving the way for future in-depth adware analysis. We make available our dataset to boost related research efforts[1].
- We provide important insights into adware ecosystems, including adware family naming patterns, revealing widely-used ad networks being identified as adware, diverse payload location strategies, and key evasion techniques. These findings contribute significantly to understanding and mitigating adware threats.

[1]Dataset available at https://github.com/security-pride/AdwareZoo.

2 METHODOLOGY

Our study on Android adware focuses on two critical aspects: dataset construction and payload detection. In §2.1, we detail the construction process of the **AdwareZoo** dataset. This involves curating a diverse collection of adware samples from multiple sources, including security reports and app repositories. §2.2 introduce our approach for identifying adware payloads, which relies on isolating packages from adware samples for VirusTotal rescanning. We elucidate the key assumptions underlying this method, our strategies for combating obfuscation, and the payload identification process.

2.1 Adware Dataset Construction

A comprehensive and up-to-date dataset is imperative to facilitate an in-depth analysis of Android adware. While Gao et al. [23] made a valuable contribution by constructing an adware dataset in 2019 based on VirusTotal labels, this dataset covers only 26 adware families. Hence, we took the initiative to construct a new adware dataset **AdwareZoo**, contributing to future research in this domain.

Typically, there are two primary approaches for curating an Android malware dataset: analyzing security company reports [20, 32], and scrutinizing large app repositories [31, 34]. We employ both approaches to construct our dataset, which comprises **15,996** adware APK samples spanning **118** distinct adware families.

2.1.1 Collecting Adware from Security Reports. We first collected Android adware samples from reports from reputable security companies. This process began by crawling reports from a curated list of 19 sites associated with renowned security companies. This list, along with the corresponding crawlers, was sourced from a recent study [20], which identified these security companies with Gartner's list of best endpoint security platforms [4]. For each site, we utilized the respective crawler to seek out reports related to Android adware with the following keyword sets: [android, mobile, aggressive, security, malware] and [ad, ad fraud, ad library, adware, clicker]. This endeavor resulted in a collection of 2,364 report posts dated from 2008 to 2023. Subsequent filtering excluded reports that either did not mention "android" in their title and body or were not authored in English, leaving us with 797 reports. We further remove those unrelated to Android adware through manual inspection, narrowing down the list to 104 reports. We then attempted to gather the adware samples associated with each report. Therefore, we retained only those reports that included hash indicators of the app (MD5, SHA1, or SHA256) that facilitate app collection. Using the hash indicator, we download the corresponding APK file from VirusTotal [8] with its premium APIs. This phase yielded 54 reports with 1,410 corresponding APK files.

Sample Filtering. Our report filtering was solely focused on eliminating reports unrelated to Android adware, instead of directly ascertaining if the described samples were adware or not by simply browsing through the report. Indeed, the definition of adware can be somewhat ambiguous, especially since advertising often serves as a revenue stream for viruses and trojans, leading to challenges in establishing a definitive set of criteria for its identification. We thus resorted to VirusTotal for the filtration of adware samples following Gao et al. [23]. Specifically, if a sample was explicitly identified as adware by at least one security vendor on VirusTotal (e.g., `Adware.AndroidOS.Magic.A!c`), we classified it as adware.

This procedure resulted in **1,132** APK files from 37 reports being categorized as adware. Of the remaining 17 reports and their corresponding samples, 14 were Dropper Trojans with ad fraud modules, and the remaining 3 were Fakeapp Trojans exhibiting ad behaviors.

2.1.2 Obtaining Adware from App Repositories via Industry Partners. Recognizing the limited scope of Android adware samples collected from security reports, we extended our dataset by incorporating adware samples from extensive app repositories. This expansion was facilitated through collaboration with our industrial partner, from whom we acquired an additional **21,018** adware samples. According to their approach, this dataset was curated as follows: Firstly, a comprehensive Android malware dataset was maintained, comprising over 3 million malware APK files collected from Koodous [7] and Virustotal, covering a decade from 2011 to 2022. Secondly, all malware samples were classified into distinct malware families using the widely adopted AVClass [29], a malware labeling tool that converts labels from various antivirus engines into a single family name for the malware. For each family, a maximum of 300 samples were then randomly selected. Lastly, the same sample filtering procedure as aforementioned was applied to all the selected samples to obtain this adware dataset. We deem this dataset a valuable addition to our research owing to its substantial size, extensive time span, and comprehensiveness in terms of malware family coverage.

2.1.3 Adware Family Classification. With the two sources combined, we now have a total of **22,150** unique adware APKs differentiated by file hashes. The next step was to obtain a family classification of these adware samples for subsequent analysis. To accomplish this, we turned to AVClass, leveraging its capabilities for malware label aggregation. Given that VirusTotal [8] is the most comprehensive security platform aggregating multiple antivirus engines, we used the VirusTotal scanning reports, which contain labels from these engines for each adware sample, as input for AVClass. In our pursuit to ensure that our dataset is representative and reflects the prevalence of adware, we implemented an additional filtration step: families with fewer than 20 samples or those for which, on average, less than two security vendors identified their samples as adware were eliminated. However, we made exceptions for seven families which, despite having fewer than 20 samples each, were identified as adware by at least five security vendors on average. They were either associated with popular advertising networks or specific adware behaviors. Given their significance in the adware ecosystem, we decided to retain these families in our dataset. Consequently, our meticulously curated dataset **AdwareZoo** encompasses **15,996** unique APKs spanning **118** distinct families.

2.2 Adware Family Payload Detection

We next attempt to locate the payload for each adware family. In the realm of software security, "payload" typically refers to the component of malware responsible for executing malicious activities. Within the scope of Android adware, a payload is the specific code component facilitating adware behaviors, such as delivering unwanted advertisements or performing ad frauds. Such payload is usually encapsulated in a Java/Kotlin code package within a DEX file (the executable file format in Android). However, payloads may also reside in resource folders as SO files (Android C++ library files accessible with JNI methods) or other executable files (e.g. additional DEXs and JARs) that can be loaded dynamically, a commonly used feature for Android software, especially malware [28, 35].

2.2.1 Key Assumption. We propose our payload detection approach based on a straightforward and intuitive assumption that, **a payload for an Android adware family, if separated and repackaged into a single app for VirusTotal scanning[2], should still be recognized by the antivirus engines as belonging to the same family**. Chen et al. [22] also used a similar approach to identify potentially harmful Android libraries, a huge proportion of which are adware payloads, by separating the library code for scanning. We evaluate this assumption in §3.1.

2.2.2 Obfuscation and Feature Hash. For each adware family, we first obtain all samples belonging to that family, then unpack and de-compile them using ApkTool [6]. As Android's codespace follows Java's package structure, and the payload for an adware family should repeatedly occur in different samples within the family, an intuitive approach would be to rank package names by their occurrence frequency among samples and choose those with higher frequencies as payload candidates for separate scanning. However, adware often employs obfuscation techniques, rendering package names unreliable for differentiating packages. To counter this, we substituted package names with feature hashes for package differentiation, a method adopted from LibRadar [27], a widely used tool for detecting third-party libraries in Android apps. The feature hash is calculated based on the frequency of different Android API calls within a package, a metric that is notably resilient to obfuscation[3].

2.2.3 Payload Identification Process. We began by calculating a feature hash for all second-level packages (e.g., com/a) of each sample in the family[4]. Whenever a new feature hash emerges for the family, we extract the corresponding package folder in its entirety (including any code files within), and repackage it into a standalone APK file for subsequent scanning[5].

However, as mentioned earlier, payloads may also be hidden in resource folders as other executable file formats. To address this, we examine the non-DEX components for all samples within the family. For each sample, we unpack the APK file, remove the DEX files within the root directory (those already de-compiled by ApkTool earlier), copy all other folders into a blank APK template, and repackage the template into a new APK for scanning.

Finally, we send our newly generated APKs for scanning and attempted to match the adware family name with the labels yielded by the antivirus engines in the scanning report (e.g., **Ashas** → Android/AdDisplay.**Ashas**)[6]. Whenever there is a match, we deem a payload to be found for that family. For APKs packed with resource

[2]For brevity, we will use "scanning" to refer to "VirusTotal scanning" in this paper.
[3]According to LibRadar [27], such feature hashes can uniquely identify different packages, as they observed no instances of two distinct packages sharing the same feature hash within their extensive dataset of one million apps.
[4]Focusing on level 2 packages strikes an optimal balance between thoroughness and efficiency. Scanning deeper levels, such as com/a/b, would exponentially increase the number of scans required, while level 2 typically provides sufficient payload differentiation.
[5]This is achieved by first creating a blank APK template using Android Studio, copying the folder into the unpacked template, and then repackaging the unpacked template into a new standalone APK file, again leveraging ApkTool.
[6]We also encountered and addressed an aliases problem during the matching process, as detailed in §3.1.

folders, if there is a match, we extract all executable files (APKs, DEXs, SOs, JARs, and ZIPs), and pack them one by one for scanning to further locate the exact payload.

3 EVALUATION

This section evaluates our adware family payload detection approach.

3.1 Verifying the Key Assumption

We first evaluate our key assumption introduced in §2.2.1, which serves as the basis for our payload detection approach.

Recall that we have successfully collected adware samples from 37 security reports released by reputable security companies (§2.1.1), we noticed that some reports indeed contain or infer the payload location of the described adware sample, which could serve as ground truth for our evaluation experiment. We then randomly selected ten of the reports that indicated the payload location and selected at most 5 adware samples for each report for the experiment. As a result, 36 samples with the indicated payloads were collected[7]. According to AVClass, they belong to 8 distinct adware families. For each of the samples, its payload is then separated and scanned by VirusTotal. We then simply search for each sample's family name within the labels yielded by the VirusTotal antivirus engines (e.g. **Ashas** → Android/AdDisplay.**Ashas**).

It turns out that, for the 36 samples, we were able to find an exact string match for 29 samples. For the remaining 7 samples, the lack of exact matches can be attributed primarily to discrepancies in family alias naming, notably in the cases of '**Ghostteam**' being referred to as 'Andr/**Ghosttm**-A' and '**Gunpoder**' as 'Trojan:**Gupno**'. This variation in naming across different antivirus engines, based on our further investigation, is quite common. Recognizing the need to address these inconsistencies, we embarked on a dedicated manual alias collection task for each of the 118 adware families from the labels obtained from VirusTotal. Such a manual process is a feasible one-time effort for existing adware families, since aliases are mostly simple rearrangements or minor alterations of the letters in the family names, such as `adviator/davitor`, `adpooh/poohad`, and `kalfere/WalkFree/reefwal`. After incorporating these collected aliases into our family name string matching process (which is also inherited in our payload identification), 35 out of the 36 samples were successfully identified in the VirusTotal re-scanning, validating the original assumption. The only sample that failed is because it only carries a partial payload.

3.2 Evaluating Payload Detection Result

To further assess the accuracy of our adware family payload detection, we randomly selected **30** families for verification. For each identified payload within these families, we conducted a manual check to confirm its authenticity as a payload of the family. The process and results are detailed below.

Payloads within DEX files. All 30 families were found to have payloads in the form of code packages within DEX files. Among these, **17** families' detected payload packages exhibited a consistent and unique package name that accurately represented the family, validating the correctness of our payload detection. The remaining **13** families showed multiple package names for their detected payload packages, which warrants further investigation. This multiplicity of package names potentially indicated the use of package name obfuscation, a prevalent evasion technique in adware families, as characterized in §4.2.1. To verify this, we manually examined the code structure within these payload packages to determine if they were similar despite different package names, as structural similarities would indicate the same payload under obfuscation. Our analysis revealed that 12 out of the 13 adware families indeed had similar and unique code structures among their respectively detected payloads, confirming both their use of obfuscation and the effectiveness of our approach in counteracting it, as described in §2.2.2. This only exception was the `Kuguo` family. Some of its identified payloads employ a sophisticated obfuscation technique called package flattening. This method completely disrupts the Java package structure by dispersing all code files across the root directory, making manual verification of payload package similarities challenging. However, we did identify some payloads within this family that were not subjected to such obfuscation, with the package name `com.kuguo` that corresponds to its family name. This partial confirmation allows us to verify at least some correct detections for this family. In conclusion, our evaluation indicates that the precision of the detected adware family payloads within DEX files ranges from **96.7%** to **100%**.

Payloads within resource folders. Among the 30 evaluated adware families, **8** were detected to contain payloads within resource folders. Specifically, 3 families had payloads in the form of additional APK files, 1 in additional DEX, 2 in JAR, and 2 in SO files. Notably, these payloads were observed across multiple samples within their respective families, and through decompilation and manual analysis, we also confirmed that these files indeed contained code related to adware behaviors, thus validating their classification as family payloads.

4 RESULTS

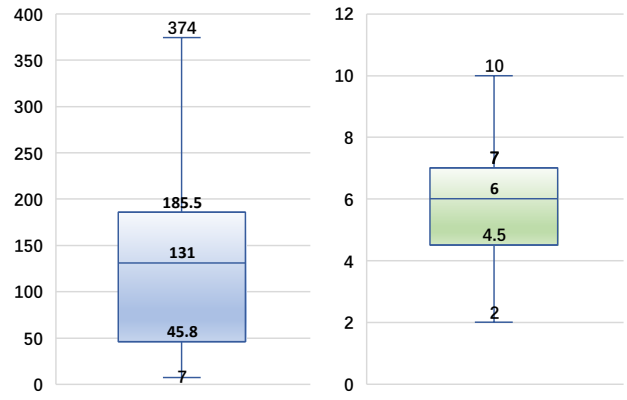
4.1 Dataset Overview

4.1.1 The Top 20 Adware Families. Table 1 presents a comprehensive overview of the top **20** Android adware families from our **AdwareZoo** dataset, ranked by sample quantity. This table encapsulates key metrics including the number of samples and average adware tag count for each family, which will be introduced in §4.1.2. Additionally, the table provides insights into family naming patterns (§4.1.3) and critical information on payload location (§4.2). This table serves as a representative showcase of our dataset's structure and content. To facilitate further research, we have made available the complete information for all 118 families in our dataset. While the corresponding samples for each family are too sizable for direct sharing, they are provided as lists of file hashes on our website and are available upon request. Our extensive dataset provides researchers with a rich, well-annotated resource for conducting in-depth analyses of Android adware ecosystems, offering a unique opportunity to explore the intricacies of adware families and their behavioral patterns.

[7]Some reports correspond to less than 5 samples. We collected 45 samples in total. However, based on our manual checking, the remaining 9 samples do not contain the payload indicated by the report and, therefore are excluded from the experiment.

Table 1: Top 20 Adware Families in our AdwareZoo Dataset

No.	Family Name	Sample Quantities	# Adware Tags	Naming Patterns		Payload Location			
				Named After Advertising Networks	Named After Behavior	DEX	Package Name	Resource	Resource Type
1	hiddad	597	4		√	√	android/support/v4/tools	√	.apk
2	dowgin	491	9			√	com/feiwo	√	.img
3	kuguo	374	7	√		√	com/kuguo	√	.so
4	ewind	360	3			√	com/dot	√	.jar
5	domob	309	4.5	√		√	cn/domob		
6	dianle	305	5	√		√	com/dianle		
7	revmob	299	4	√		√	com/revmob		
8	mobcore	297	7	√		√	com/mobcore		
9	cyfin	294	5			√	com/cyjh		
10	airpush	292	7	√		√	zmm0		
11	adviator	291	9	√		√	com/av111411	√	.apk
12	gappusin	281	9	√		√	com/waps		
13	zdtad	269	11			√	com/zdtpay	√	.jar
14	youmi	264	7	√		√	net/youmi		
15	leadbolt	263	5	√		√	com/wpuvbenaeamobfo		
16	plague	253	4			√	net/htfstudio		
17	obtes	247	9			√	ouo		
18	tgpotato	247	3.2			√	com/tgpotato		
19	adwo	241	7	√		√	com/adwo		
20	dasu	238	5			√	com/loader		



(a) Sample Quantities Distribution **(b) Adware Tags Distribution**

Figure 1: Boxplot Distributions of Sample Quantities and Adware Tags for the 118 Adware Families

4.1.2 Distribution of Sample Quantities and Adware Tags Across Families. The 118 adware families within our dataset are each characterized by two key metrics: sample quantity and average number of adware tags. The quantity of samples within each family serves as a proxy indicator of the family's prevalence in the wild, while the average number of adware tags—the mean number of antivirus engines on VirusTotal that identified samples within a family as adware—represents the level of agreement among security vendors. This metric not only reflects the reliability of classifying a family as adware, but also serves as a rough indicator of its potential severity or aggressiveness. A higher number of adware tags suggests stronger agreement among security vendors about the adware nature of the samples, potentially indicating more obvious or problematic behaviors. Conversely, lower numbers might point to more sophisticated evasion techniques or less certainty in classification.

Figure 1a presents a boxplot illustrating the distribution of sample quantities across adware families. The distribution exhibits a notable right-skewed pattern, meaning that while most families have relatively fewer samples, a small number of families possess substantially larger sample sizes, including outliers like the `hiddenads` family with 597 samples and `dowgin` with 491 samples[8]. The median sample size is 131, with a mean of 135.6 samples. The interquartile range spans from 45.8 to 185.5 samples, delineating the range within which the majority of families fall. Figure 1b depicts the distribution of average adware tag numbers across families. This distribution exhibits a relatively symmetrical and concentrated pattern. The median and mean are both approximately 6 tags (the precise mean being 6.01). Most adware families have between 4.5 and 7 tags. The overall range extends from 2 to 14 tags.

4.1.3 Adware Family Naming Patterns. In the field of Android malware analysis, researchers typically employ malware labeling tools such as AVClass or Euphony [24] to derive family names [37, 39]. These family names are aggregated from labels provided by multiple antivirus engines, each adhering to its naming conventions. Consequently, the resulting family names often present a diverse and chaotic landscape, with some names lacking clear semantic meaning. Despite the apparent disorder in the nomenclature of the 118 adware families in our dataset, we have identified two significant patterns in adware family naming conventions:

[8]These extreme outliers are not displayed in the boxplot to maintain visual clarity and focus on the primary distribution.

Named After Advertising Networks. A prominent naming convention we observed involves families being named after advertising networks, suggesting that samples within these families are classified as adware due to their adoption of specific advertising libraries. While it is impractical to identify all such families, particularly those associated with minor ad networks, we have identified at least **37** families following this naming pattern. The naming conventions, however, are not always straightforward. Some are directly named after the advertising company, while others are named based on the advertising library's package name. In some cases, names are derived from specific resource file names such as SO libraries (e.g., Family `Landlord` employs a SO library with a matching name, specifically `liblandlord.so`.). This variety likely reflects the diverse detection methods employed by antivirus companies to identify these adware families.

Interestingly, some families are named after well-known ad networks. By cross-referencing with the top Android ad networks listed on AppBrain [13], we found **10** such instances, including `Adcolony`, `AppLovin`, `InMobi`, `Mobwin`, `Airpush`, and `Leadbolt`, among others. While we address the reasons for `InMobi`'s identification in §4.2.1, the rationale behind most of these classifications remains unclear. Given the extensive reach of these networks in the mobile advertising market, this situation underscores the urgent need to demystify the characteristics of these adware families. It also calls for greater transparency and regulation in the rapidly evolving landscape of the mobile advertising ecosystem.

Table 2: Adware Families Named after Advertising Behaviors

Family	# Variants	Behavior Description
Hiddad Hiddenad Hiddenads	21	Invisible or out-of-app ads
Clicker	2	Fraudulent ad clicks generation
Noiconads	N/A	Displaying ads while hiding app icon
Mulad	N/A	Apps adopting multiple ad networks

Named After Behavior. We observed another notable naming convention where adware families are designated based on specific advertising behaviors. Our analysis identified at least **6** such families, with their names and corresponding behavioral descriptions presented in Table 2. Unlike families named after ad networks, samples within these families often display a scattered distribution of distinct payloads with diverse package names, indicating the presence of multiple variants within each family. The number of these variants is also included in the table, highlighting the diversity of implementing similar advertising behaviors in adware.[9] Notably, most of these advertising behaviors are fraudulent or aggressive in nature, underscoring their potential for harm and the critical importance of demystifying their characteristics.

We present a comprehensive, well-annotated Android adware dataset **AdwareZoo**, comprising key information including the number of samples, average adware tag count, and payload location information for each family. We identified two primary naming patterns for adware families: those derived from advertising networks (37+ families) and those based on specific advertising behaviors (6 families). Notably, 10 well-known ad networks are identified as adware, warranting further attention.

[9]We also provide payload information for each identified variant on our website.

4.2 Adware Family Payload Location

The distribution of payload locations across adware families offers crucial insights into the implementation strategies preferred by adware developers. Our analysis successfully identified the payload locations for **111** out of **118** adware families within our dataset. For the remaining 7 families we failed to locate payloads, one were due to the adoption of app packing techniques that hinder reverse engineering efforts, another adopted sophisticated obfuscation techniques, namely package flattening, which completely disrupts the Java package structure by dispersing all code files across the root directory, rendering our approach ineffective. Additionally, the "Mulad" family integrates multiple ad networks, resulting in the absence of a fixed payload. Consequently, separate scanning of any individual package within a "Mulad" sample fails to trigger identification as "Mulad". For the remaining four families, we were unable to locate payloads, and the reasons for this eluded our analysis. We hypothesize that these families may employ payloads dynamically loaded during app runtime, or that antivirus engines identify these families based on cross-package relationships that our approach does not account for.

We categorized the 111 families based on their payload locations, as illustrated in Figure 2. Our analysis reveals that an overwhelming majority of **107** families (96.4%) contained payloads within their DEX files[10], representing the most prevalent approach. Furthermore, **36** families (32.4%) housed their payloads in local resources. Notably, **32** families (28.8%) exhibited a hybrid strategy, integrating payloads in both Java code and local resources. This significant proportion of hybrid implementations indicates a trend towards more sophisticated and potentially more resilient adware architectures.

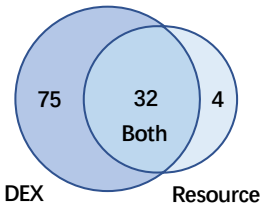


Figure 2: Distribution of Payload Locations

4.2.1 Payloads within DEX files. More than 90% adware families incorporate their payloads within the DEX files of the APK. This means they are implemented in Java/Kotlin, aligning with common Android development practices. For these families, we also identified the most frequently recurring package names associated with their payloads, as exemplified in Table 1. However, upon closer examination, several noteworthy findings reveal that the nature of these payloads is more complex than anticipated.

[10]This refers to the main classes*.dex files within the root directory of the APK. DEX files found in resource folders are categorized under "Resource" as they are loaded dynamically.

Package Name Obfuscation. As mentioned in §2.2.2, adware payloads often obfuscate their package names to evade detection. We hence characterize this phenomenon. Recall that we identify payloads in the form of feature hashes, which are calculated from the adware samples' second-level packages (e.g., com/a). For each feature hash identified as payload, we locate all packages with this hash across the adware family's samples and compare their package names. Our analysis revealed that **31** families exhibit different package names for payloads with identical feature hashes, providing concrete evidence of payload package name obfuscation. Notably, many of these obfuscated names convey minimal semantic information, likely an attempt to obscure their origin, such as the advertising network company behind them. This approach starkly contrasts with common ad networks, which often use explicit SDK package names like `"com/facebook/ads"` or `"com/amazon/device/ads"`. For instance, consider the "leadbolt" adware family. we observed its obfuscated payloads `"com/qzmpeyumuvalcpblod"` and `"com/dscevemibbpwdfxe"` sharing the same feature hash. These package names, however, offer no meaningful indications about the payload's functionality or origin.

Payload Version Diversity within Families. Next, we investigated whether multiple different feature hashes exist for payloads with the same package name within a family, indicating the presence of different payload versions. Our analysis identified **47** families exhibiting multiple payload versions for the same payload package name.

A particularly intriguing case is the `inmobi` family, which is named after a prominent yet not infamous Indian multinational mobile advertising network [9], an uncommon situation for adware families. Within our dataset, we identified four distinct versions of its payload. Further investigation revealed that these payloads are a batch of early InMobi SDK versions dating back to 2013, which exploited a then-existing Android WebView vulnerability to access sensitive privileged information such as photos, SMS, and even make phone calls [5]. This vulnerability has been fixed since Android 4.2 [10], and InMobi has long ceased such practices. Only a fraction of their SDK versions that exploited this vulnerability were identified as adware.

This case demonstrates that even renowned advertising networks have, at specific periods in their development history, adopted practices that could be perceived as privacy-invasive or potentially malicious. It highlights the complexity of the mobile advertising ecosystem and the challenges advertising libraries face in balancing functionality with user privacy.

Hidden in common packages. In addition to package name obfuscation, our investigation uncovered another prevalent strategy employed by adware payloads to evade detection: placing themselves under directories of widely used and trusted packages, such as `com.google` or `android`. This tactic likely aims to gain the trust of security personnel or exploit the whitelists of certain virus detection engines. Our analysis revealed that **17** adware families employed this sophisticated method, strategically positioning their payloads under common package names. Some families went even further, distributing their code across multiple common packages. Figure 3 illustrates the distribution of common packages under which these adware families hide their payloads.

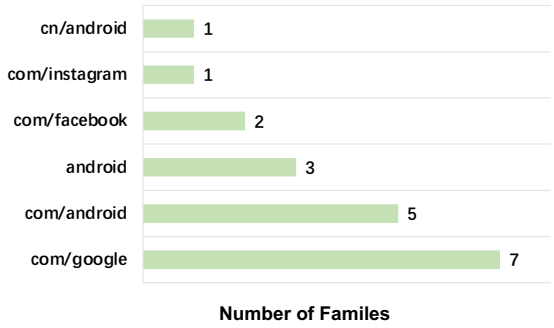


Figure 3: Common Packages Adware Payloads Hide Under

4.2.2 Payloads within resource folders. Our analysis revealed that **36** adware families carry payloads located within app resource folders. Of these, **8** families utilized SO files (Android C++ library files). These files are notably more challenging to analyze statically [11] compared to DEX files, making them an effective vehicle for adware payloads to conceal their intended behaviors. However, their development complexity likely contributes to their relatively low prevalence in our dataset, suggesting that creating SO libraries remains a hurdle for many adware payload developers.

Dynamic loading emerged as another prevalent evasion technique. This method typically involves loading JAR, DEX, or APK files during runtime to circumvent antivirus software checks. Our investigation uncovered **14** families with JAR and DEX files in their resource folders. Furthermore, **17** families contained additional APK payloads, with 6 of these families obfuscating the file extensions of the APKs to further evade detection.

Intriguingly, we identified **7** families where merely uploading the Manifest files resulted in an adware flag. This finding suggests that some antivirus engines adopted semantic checking of the Manifest file for detection.

Our analysis of adware family payload location reveals diverse payload implementation strategies, with 96.4% using DEX files, 32.4% utilizing resource folders, and 28.8% adopting a hybrid approach. We identified key evasion techniques including package name obfuscation, hiding under trusted package names, and employing SO files and dynamic loading. These findings provide a crucial understanding of adware development strategies, offering valuable insights for future adware detection and analysis.

5 LIMITATION AND FUTURE WORK

Our research represents a crucial step towards demystifying Android adware through the construction of a comprehensive dataset and the localization of adware payloads. However, certain limitations in this work warrant attention: Firstly, while our dataset is extensive, collecting adware from both security reports and app repositories, it may not encompass all existing adware families given the vast scale of the Android app ecosystem. Secondly, Our payload analysis method, which relies on app package structure and rescanning of isolated packages using VirusTotal, may be ineffective in certain scenarios, including: a) advanced obfuscation techniques such as package flattening, b) app packing methods that

conceal code structure, c) dynamically loaded payloads that are not present in the static APK, d) families identifiable only through inter-package relationships. Fortunately, these limitations collectively affected only 7 families for which we were unable to detect payloads. Lastly, in our evaluation of payload detection results (3.2), while we attempted to verify the identified payloads through dynamic testing, the wide time span of our dataset posed challenges. Many adware samples' remote communication servers had already ceased operations, impeding our ability to observe their real-time behavior. To mitigate this, we pivoted to conducting thorough static analysis on the adware samples to ensure the highest possible accuracy within these constraints. Despite these limitations, our research provides a solid foundation for understanding and combating adware.

Future work. This study has established a solid foundation for demystifying Android adware through dataset construction and payload localization. However, the intricate characteristics of adware and their distinctive features compared to general Android malware necessitate further in-depth investigation. In our future work, we plan to conduct a detailed, multifaceted analysis of the detected payloads to uncover the unique behavioral patterns of adware. We intend to examine various aspects, including aggressive advertising behaviors, ad fraud techniques, and potential privacy implications. Through these continued efforts, we aspire to build upon our current work to provide a deeper and more nuanced understanding of the Android adware landscape, ultimately contributing to the development of more targeted and efficient countermeasures in the realm of mobile security.

6 RELATED WORK

Android Malware Dataset. The development of Android malware datasets has been crucial for security research, with pioneering efforts like MalGenome [38] in 2011, which relied on security reports to collect 49 malware families. This was followed by more comprehensive collections such as AMD [33], Drebin [18], RmvDroid [31], MalRadar [32], and Cao et al. [20]. These datasets have significantly contributed to the advancement of Android security research and defense mechanisms. However, these datasets primarily focus on conventional malware, limiting their applicability for comprehensive adware analysis. While other researchers have developed specialized datasets for specific malware categories like piggybacking [19], ransomware [21], and banking malware [26], there remains a significant gap in datasets specifically tailored for adware research. Our work filled this gap with a comprehensive, well-annotated Android adware dataset across 118 distinct adware families.

Adware. While our study is not the first to investigate Android adware, to the best of our knowledge, we are the pioneering effort to demystify adware. Previous research has primarily focused on adware identification using various machine learning techniques. For instance, Suresh et al. [30] conducted classification experiments on a dataset comprising seven adware families, while Alani et al. [17] employed machine learning models for adware identification using 79 features extracted from network traffic of 250 adware samples across 5 families. However, these studies typically utilized limited datasets, lacking comprehensiveness and failing to locate or analyze the payloads themselves.

Research on ad libraries, while not specifically focused on adware, has contributed significantly to our understanding of the adware ecosystem. For instance, Li et al. [25] conducted a comprehensive study, identifying 240 ad libraries by analyzing approximately 1.5 million apps from Google Play. In a related effort, Zhang et al. [36] investigated 4,957 potentially malicious third-party libraries, which included various ad libraries. Their research delved into critical aspects such as repackaging techniques and potentially harmful behaviors associated with these libraries. Although these studies were not exclusively dedicated to adware analysis, they form an integral part of adware research by providing insights into the broader landscape of mobile advertising and its potential misuse.

Gao et al. [23] made a notable contribution to adware research by attempting to determine the maliciousness of adware through an analysis of VirusTotal's malicious labels. Their findings revealed that up to 50% of samples within an adware family could be flagged as malicious, providing valuable insights into the potential harm of adware. However, their study was limited to 26 pre-known adware families and did not provide a dataset specifically dedicated to comprehensive adware analysis.

In summary, prior research has contributed to various aspects of adware study, including identification techniques, ad library analysis, and maliciousness assessment. However, these efforts have often been limited in scope, focusing on small sets of known adware families or specific behaviors. Notably absent is a comprehensive, well-annotated dataset that facilitates in-depth payload analysis across a wide range of adware families. Our work addresses this gap by providing a large-scale adware dataset and locating adware payloads. This approach aims to enhance understanding of adware payload implementation strategies and support future research in demystifying and mitigating mobile adware threats.

7 CONCLUSION

This study represents a significant stride in understanding and addressing the pervasive threat of Android adware. Through our comprehensive dataset construction and payload identification, we have constructed **AdwareZoo**, a robust, well-annotated dataset of 15,996 adware samples across 118 distinct families, filling a critical gap in mobile security research. Our analysis has unveiled several key insights into the adware ecosystem. This work contributes to a deeper understanding of adware and highlights the pressing need for increased scrutiny and regulation of mobile advertising practices. Furthermore, the complexity and prevalence of adware underscore the importance of continued research to demystify adware and to develop more effective detection and mitigation strategies.

ACKNOWLEDGEMENT

This work was supported by the Key R&D Program of Hubei Province (2023BAB017, 2023BAB079), the National NSF of China (grants No.62072046, 62302181), the Knowledge Innovation Program of Wuhan-Basic Research (2022010801010083), the Xiaomi Young Talents Program, and the HUSTCSE-FiberHome Joint Research Center for Network Security.

REFERENCES

[1] 2015. New Android Malware Family Evades Antivirus Detection by Using Popular Ad Libraries. https://unit42.paloaltonetworks.com/new-android-malware-family-evades-antivirus-detection-by-using-popular-ad-libraries/.

[2] 2019. Adware Posing as 85 Photography and Gaming Apps on Google Play Installed Over 8 Million Times. https://www.trendmicro.com/en_us/research/19/h/adware-posing-as-85-photography-and-gaming-apps-on-google-play-installed-over-8-million-times.html.

[3] 2020. Fake Android Apps Communicate For Malware, Ad Fraud. https://www.trendmicro.com/en_us/research/20/b/malicious-apps-on-google-play-communicate-with-trojans-install-malware-perform-mobile-ad-fraud.html.

[4] 2021. Gartner Magic Quadrant for Endpoint Protection Platforms. https://www.gartner.com/en/documents/4001307.

[5] 2022. javascript-interfaces. https://securityqueens.co.uk/android-attack-javascript-interfaces-and-webviews/.

[6] 2023. Apktool. https://github.com/iBotPeaches/Apktool.

[7] 2023. Koodous. https://koodous.com/.

[8] 2023. Virustotal. https://www.virustotal.com/.

[9] 2024. inmobi. https://www.inmobi.com/.

[10] 2024. JavascriptInterface. https://developer.android.com/reference/android/webkit/JavascriptInterface.

[11] 2024. secret-inside-of-so-file. https://medium.com/@jinwei.han93/secret-inside-of-so-file-c073c8375c63.

[12] 2024. The mobile malware threat landscape in 2023. https://securelist.com/mobile-malware-report-2023/111964/.

[13] 2024. Well Known Ad-Networks. https://www.appbrain.com/stats/libraries/ad-networks.

[14] 2024. What is a adware. https://www.techtarget.com/searchsecurity/definition/adware.

[15] 2024. What is a greyware. https://www.malwarebytes.com/glossary/greyware?srsltid=AfmBOoqKBDEZ2hWoLZpLcj8j0FySWgliAuicMgFJ-cj49wkQJuuMwcLg.

[16] 2024. What is PUAs. https://en.wikipedia.org/wiki/Potentially_unwanted_program.

[17] Mohammed M Alani and Ali Ismail Awad. 2022. AdStop: Efficient flow-based mobile adware detection using machine learning. *Computers & Security* 117 (2022), 102718.

[18] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Ndss*, Vol. 14. 23–26.

[19] Chongyang Bai, Qian Han, Ghita Mezzour, Fabio Pierazzi, and VS Subrahmanian. 2019. DBank: Predictive Behavioral Analysis of Recent Android Banking Trojans. *IEEE Transactions on Dependable and Secure Computing* 18, 3 (2019), 1378–1393.

[20] Michael Cao, Khaled Ahmed, and Julia Rubin. 2022. Rotten apples spoil the bunch: an anatomy of Google Play malware. In *Proceedings of the 44th International Conference on Software Engineering*. 1919–1931.

[21] Jing Chen, Chiheng Wang, Ziming Zhao, Kai Chen, Ruiying Du, and Gail-Joon Ahn. 2017. Uncovering the face of android ransomware: Characterization and real-time detection. *IEEE Transactions on Information Forensics and Security* 13, 5 (2017), 1286–1300.

[22] Kai Chen, Xueqiang Wang, Yi Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Bin Ma, Aohui Wang, Yingjun Zhang, and Wei Zou. 2016. Following devil's footprints: Cross-platform analysis of potentially harmful libraries on android and ios. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 357–376.

[23] Jun Gao, Li Li, Pingfan Kong, Tegawendé F Bissyandé, and Jacques Klein. 2019. Should you consider adware as malware in your study?. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 604–608.

[24] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. 2017. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 425–435.

[25] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. An investigation into the use of common libraries in android apps. In *2016 IEEE 23Rd international conference on software analysis, evolution, and reengineering (SANER)*, Vol. 1. IEEE, 403–414.

[26] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (2017), 1269–1284.

[27] Ziang Ma, Haoyu Wang, Yao Guo, and Xiangqun Chen. 2016. LibRadar: fast and accurate detection of third-party libraries in Android apps. In *Proceedings of the 38th international conference on software engineering companion*. ACM, 653–656.

[28] Sebastian Poeplau, Yanick Fratantonio, Antonio Bianchi, Christopher Kruegel, and Giovanni Vigna. 2014. Execute this! analyzing unsafe and malicious dynamic code loading in android applications.. In *NDSS*, Vol. 14. 23–26.

[29] Silvia Sebastián and Juan Caballero. 2020. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference*. 42–53.

[30] Supraja Suresh, Fabio Di Troia, Katerina Potika, and Mark Stamp. 2019. An analysis of Android adware. *Journal of Computer Virology and Hacking Techniques* 15 (2019), 147–160.

[31] Haoyu Wang, Junjun Si, Hao Li, and Yao Guo. 2019. Rmvdroid: towards a reliable android malware dataset with app metadata. In *2019 IEEE/ACM 16th international conference on mining software repositories (MSR)*. IEEE, 404–408.

[32] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying android malware in the new era. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 2 (2022), 1–27.

[33] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.

[34] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 252–276.

[35] Yinxing Xue, Guozhu Meng, Yang Liu, Tian Huat Tan, Hongxu Chen, Jun Sun, and Jie Zhang. 2017. Auditing anti-malware tools by evolving android malware and dynamic loading technique. *IEEE Transactions on Information Forensics and Security* 12, 7 (2017), 1529–1544.

[36] Zicheng Zhang, Wenrui Diao, Chengyu Hu, Shanqing Guo, Chaoshun Zuo, and Li Li. 2020. An empirical study of potentially malicious third-party libraries in android apps. In *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 144–154.

[37] Yanjie Zhao, Li Li, Haoyu Wang, Haipeng Cai, Tegawendé F Bissyandé, Jacques Klein, and John Grundy. 2021. On the impact of sample duplication in machine-learning-based android malware detection. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 30, 3 (2021), 1–38.

[38] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.

[39] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online {AntiMalware} engines. In *29th USENIX Security Symposium (USENIX Security 20)*. 2361–2378.