Behavior Speaks Louder: Rethinking Malware Analysis Beyond Family Classification

Fei Zhang†, Xiaohong Li†, Sen Chen†, Ruitao FENG*
†College of Intelligence and Computing, Tianjin University, Tianjin, China
{zhangfei, xiaohongli, senchen}@tju.edu.cn
*Faculty of Science and Engineering, Southern Cross University, Australia
ruitao.feng@scu.edu.au

***Abstract*—The classification of malicious families is essential in Android malware analysis. However, inconsistent naming standards across different antivirus companies hinder accurate identification and understanding of malicious behaviors. This study conducts an extensive analysis of Android malware families to address these challenges. First, we compared family definitions from various antivirus companies and found significant inconsistencies in the level of detail and descriptions of malicious behaviors. These inconsistencies undermine effective malware classification and analysis. Second, we assessed the alignment between described and exhibited malicious behaviors, revealing that family definitions often provide only a broad outline, omitting critical details. Additionally, evolving malware behaviors often surpass existing family definitions. To address these issues, we propose using specific behavior labels to directly indicate malicious behaviors in malware attack chains. Leveraging large language models (LLMs) and a detailed analysis of Android malicious behaviors, we identified six key behavior labels. To streamline the labeling process, we designed the AMBL framework, which automates the generation of behavior labels for malware. Our novel feedback mechanism-based LLM analysis method establishes relationships between APIs and behavior labels, crucial for accurate label updating. Through AMBL, a dataset with behavior analysis reports has been outputed and open sourced. An online survey and manual analysis are also conducted to validate the effectiveness of the AMBL framework and the reliability of the dataset.**

***Index Terms*—Android malware, family definitions, malicious behaviors**

I. Introduction

Currently, Android is the most popular operating system in the world, with the number of apps on Google Play reaching 2.4 million [1]. However, the openness and flexibility that make Android so appealing to users and developers also render it a prime target for malicious actors. Alarmingly, the number of Android malware instances has surged from 22,000 in 2012 to 33 million by Jan 2023 [2]. This sharp increase in malware poses significant challenges to effective malware mitigation, underscoring the growing importance of robust Android malware detection.

Current research on Android malware primarily focuses on three areas: binary classification [3]–[6], family classification [7]–[9], and behavior classification [10]. Binary classification [3] aims to determine whether an app is benign or malicious and has achieved a great performance. However, these models often function as black boxes, providing little insight into the specific malicious behaviors of the malware or the attack chain. Understanding these behaviors is crucial for enhancing detection techniques and remediation efforts. Family classification [7] categorizes malware into specific families to facilitate behavioral analysis. Researchers can use the classification results to dig into subsequent malicious behaviors, analyze attack chains, and devise defensive measures. Effective analysis requires ❶ Fast response: Quickly retrieving family definitions based on family names; ❷ Behavioral logic: Understanding the specific behavioral logic of malware; ❸ Complete behaviors: Encompassing all malicious behaviors within family definitions.

To explore whether the current family definitions meet these criteria, we conducted a study from two perspectives (as shown in Figure 1).

- **Definition Comparison**: We collected 139 Android malware family definitions from two antivirus (AV) companies to assess consistency. This involved examining the uniformity of definitions for the same family across different AV companies and identifying overlaps or duplications in family definitions.
- **Behavior Comparison**: We developed an automated framework to extract malicious behaviors from malware and compared them with those specified in their family definitions. This analysis aimed to determine the accuracy and comprehensiveness of family definitions in identifying malicious behaviors.

We have identified the following shortcomings in family definitions:

- **Inconsistency & Overlap**: Different AV companies have varying definitions for the same malware family, with instances of duplication in family naming and definitions complicating the retrieval of accurate family definitions.
- **Broader Definition**: Some malware families have overly broad behavior descriptions, hindering the understanding of specific behavioral logic.
- **Beyond Definition**: The iterative nature of malware evolu-

This research was supported by the National Natural Science Foundation of China (No.62332005).

* The corresponding author is Ruitao FENG.

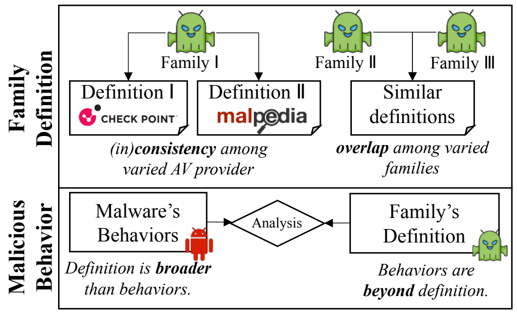


Fig. 1: Two Perspectives of the Study

tion leads to behaviors beyond the scope of family definitions, obstructing comprehensive attack chain analysis.

Unlike family classification, behavior classification [10] seeks to directly label malicious behaviors of malware. This approach, however, faces challenges due to the lack of standardized behavior labels and the evolving nature of malware, which necessitates continuous updates to the behavior labels. In this paper, we propose using multiple behavior labels to directly indicate all malicious behaviors of malware. This type of approach allows for updating labels as new malicious behaviors emerge. However, we face several challenges:

- **Lack of Standards**: There is no formal universal standard for categorizing specific malware behaviors, making the summarization of comprehensive behavior labels challenging.
- **Manual Annotation Costs**: While Android malware datasets exist, annotating them with precise behavior labels at the program level is prohibitively expensive.

In this paper, to ensure the effectiveness of malicious behavior labels, we first summarized six malicious behavior labels based on dangerous permissions and sensitive APIs with the assistance of large language models (LLMs). Additionally, we constructed a novel prompt template for LLM to establish the relationships between APIs and labels. We designed the **A**utomated **M**alicious **B**ehavior **L**abeling (AMBL) framework, which integrates various sources of information such as the official Android API documentation, malware behavior analysis reports, and static program analysis results. This framework enables the direct addition of behavior labels to malicious apps, thus reducing the need for costly manual labeling. Finally, we utilized this framework to establish a dataset containing 1,000 samples of malware, which we made open-source for the research community. We also conducted an online survey to evaluate whether the malicious behavior labels generated by AMBL can accurately and comprehensively represent the malicious behaviors of malware. The results collected online confirmed the effectiveness of the malicious behavior labels. Additionally, we manually analyzed the complete attack chains of certain malware samples in the dataset to verify the accuracy and completeness of the labels, ensuring the quality of the dataset.

In summary, we make the following main contributions:

- **Comprehensive Study**: This is the first work to systematically identify imperfections in current malware family definitions, providing detailed analyses with examples and statistical results.
- **Malicious Labels**: We summarize six malicious labels that represent various aspects of malicious behaviors based on extensive pre-understanding of malware analysis.
- **AMBL Framework**: We implement a framework that automatically generates behavior labels for malware, providing extensive information about malware, including permissions, intents, APIs, function call graphs, and malicious labels.
- **Open-Source Dataset**: We generate comprehensive malicious analysis reports for 1,000 malware, making the dataset open-source [1] for further research and application by the community.

II. Related Work

ML-based methods [11]–[14] have been widely applied in the field of security, among which ML-based Android malware detection can be broadly categorized into binary classification and family classification. For binary classification, numerous studies have achieved remarkable detection results [6], [15]–[18]. For instance, Yerima et al. [15] proposed a method based on a Bayesian classification model using APIs, system commands, and permissions to detect Android malware. Wu et al. [18] utilized data flow APIs as classification features for Android malware detection. However, due to the black-box nature of machine learning-based classification methods, binary classification results can not aid researchers in analyzing the attack chain of malware in detail.

Family classification has been considered an effective method for further malware analysis [7]–[9]. M. Fan et al. [7] constructed frequent subgraphs to represent common behaviors of malware within the same family, improving classification accuracy. Y. Bai et al. [8] applied Siamese networks to malware family classification, showing innovative progress in this field. Despite the widespread use of family classification, our survey found inconsistencies in naming and defining malware families among AV companies. This inconsistency can hinder rapid response to malware threats. Additionally, continuous updates in malware can lead to behaviors that exceed existing family definitions.

Inspired by advancements in program analysis [19]–[22], some researchers [10], [23]–[25] have initiated efforts to directly identify malicious behaviors at the program level. R. Feng et al. [24] summarized four fine-grained malicious behaviors to describe malicious behaviors of IoT malware comprehensively. Q. Qiao et al. [10] generated malicious behavior labels from detailed analysis reports of malware and identified malicious behaviors. However, the behavioral labels of the analyzed malware are highly imbalanced in this paper, which raises concerns among researchers regarding the generality of these behavioral labels.

[1]https://github.com/Behavior-speaks-loader/Behavior-speaks-loader

Our work, through a comprehensive study, systematically illustrates the limitations of malware family classifications, thereby proposing the use of multiple behavioral labels to directly identify the malicious activities of malware. We have designed the AMBL framework to explicitly assign behavioral labels to malware, and ultimately, we have open-sourced a dataset with detailed labels for further research by academia and industry.

III. Empirical Study

In order to comprehensively understand the shortcomings of using family definitions for malware analysis, we conduct an empirical study from two perspectives: the consistency of malicious family definitions by contrasting definitions provided by different vendors, and the relationship between the malicious behaviors exhibited by malware and their corresponding family definitions, therefore, providing insights into the severity of the obstacles to analysis in the current ecosystem.

A. Comparison of Family Definitions

The lack of standardized protocols governing the naming and definition of malware families by security vendors poses a significant challenge in the field. Through an investigation of public malware family information [26], it becomes evident that security vendors define malware families according to their own criteria, leading to a proliferation of families with varying and often inadequate definitions. In light of this, our study aims to compare the definitions of malware families provided by two major security vendors, Malpedia and CheckPoint, to shed light on the consistency and overlap in these definitions.

To conduct this comparison, we attempted to scrape Android malware family information from various major AV companies. However, considering the authority of the sources and the access limitations of web scraping, we ultimately selected family definitions provided by Malpedia [27], which houses information on 72 Android malware families, and CheckPoint [28], which catalogs 67 Android malware families.

1) Consistency in Definitions of Malware Families: By comparing the family definitions collected from Malpedia and CheckPoint, We found the existence of malicious families defined by both AV companies. To assess the consistency of definitions for the same malicious families across different security vendors, we conducted a comparative analysis focusing on 16 common malicious families found in both Malpedia and CheckPoint databases. Our analysis aimed to identify variations in definitions, particularly in terms of covered malicious behaviors and granularity.

We observed notable discrepancies in the definitions provided by different security vendors for the same malicious families. For instance, in the case of the SharkBot family, CheckPoint includes anti-analysis malicious behaviors such as sandbox evasion, whereas Malpedia does not. This discrepancy is illustrated in the upper part of Figure 2. Furthermore, variations in the granularity of describing malicious behaviors were evident. For example, while Malpedia identifies Catelites as a banking Trojan that steals banking credentials, CheckPoint provides a more detailed description, including creating fake icons, sending fake system notifications, reauthenticating with Google Services, and soliciting credit card details. This discrepancy is illustrated in the middle of Figure 2. Table I summarizes the findings for the 16 malicious families, indicating differences in the mentioned malicious behaviors and granularity of descriptions. Notably, 4 families (AndroRat, SharkBot, FlyTrap, and AhMyth) exhibit differences in malicious behaviors, while 9 families (Catelites, FurBall, Gustuff, etc.) show variations in granularity.

Our research suggests that while identifying the malicious software belonging to a particular family is straightforward, determining the exact definition of that family based solely on its name is challenging. We speculate that discrepancies in definitions between security vendors may arise from efforts to summarize malicious behaviors to expedite identification, potentially introducing biases.

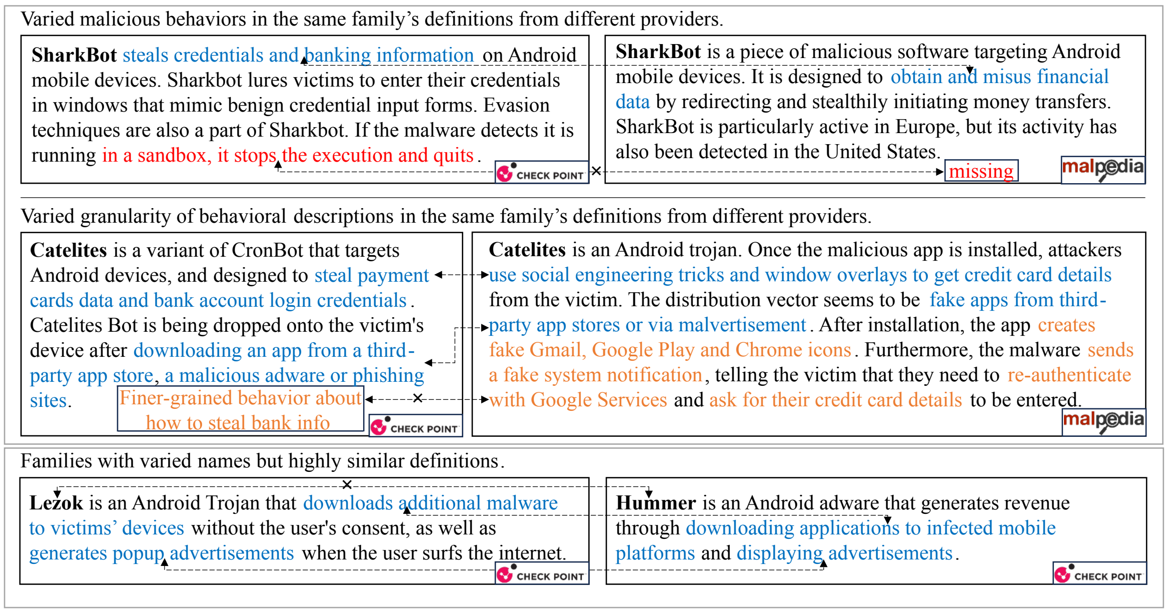
TABLE I: Definition Comparison of Same Families

Family	Same definition	Different behaviors	Different granularity
Catelites			✓
FurBall			✓
Joker	✓		
Gustuff			✓
AndroRat		✓	
Fakecalls	✓		
Eventbot			✓
LokiBot			✓
Anubis			✓
SharkBot		✓	
FluBot			✓
Medusa			✓
FlyTrap		✓	
Cerberus			✓
FaceStealer	✓		
AhMyth		✓	

Conclusion #1: *The variation in how security vendors define malicious families highlights the difficulty in standardization, emphasizing the necessity for more research. Inconsistencies exist among providers in both the depth of descriptions and the identified malicious behaviors.*

2) Overlap in Definitions of Malware Families: The analysis of family definitions provided by the CheckPoint security vendor revealed significant overlap in definitions across multiple families. Notably, families such as Lezok and Hummer exhibited identical malicious behaviors, involving the download of other malicious software onto infected devices and subsequent display of advertisements. To investigate the uniqueness of family definitions further, we conducted a comparative analysis between definitions from two major AV companies, Malpedia and CheckPoint.

Manual identification of semantically similar pairs of families from numerous definitions proved to be highly time-consuming and impractical, requiring comparisons across


Varied malicious behaviors in the same family's definitions from different providers.

SharkBot steals credentials and banking information on Android mobile devices. Sharkbot lures victims to enter their credentials in windows that mimic benign credential input forms. Evasion techniques are also a part of Sharkbot. If the malware detects it is running in a sandbox, it stops the execution and quits.

CHECK POINT

SharkBot is a piece of malicious software targeting Android mobile devices. It is designed to obtain and misus financial data by redirecting and stealthily initiating money transfers. SharkBot is particularly active in Europe, but its activity has also been detected in the United States.

missing

malpedia

Varied granularity of behavioral descriptions in the same family's definitions from different providers.

Catelites is a variant of CronBot that targets Android devices, and designed to steal payment cards data and bank account login credentials. Catelites Bot is being dropped onto the victim's device after downloading an app from a third-party app store, a malicious adware or phishing sites.

Finer-grained behavior about how to steal bank info

CHECK POINT

Catelites is an Android trojan. Once the malicious app is installed, attackers use social engineering tricks and window overlays to get credit card details from the victim. The distribution vector seems to be fake apps from third-party app stores or via malvertisement. After installation, the app creates fake Gmail, Google Play and Chrome icons. Furthermore, the malware sends a fake system notification, telling the victim that they need to re-authenticate with Google Services and ask for their credit card details to be entered.

malpedia

Families with varied names but highly similar definitions.

Lezok is an Android Trojan that downloads additional malware to victims' devices without the user's consent, as well as generates popup advertisements when the user surfs the internet.

CHECK POINT

Hummer is an Android adware that generates revenue through downloading applications to infected mobile platforms and displaying advertisements.

CHECK POINT

Fig. 2: Three Examples of Family Definition Comparison

4,392 pairs of family definitions. To address this challenge, we leveraged the semantic understanding capabilities of modern large language models (LLMs), specifically utilizing three open-source models: Llama 3 [29], Gemma [30] and Llava [31]. By constructing and using appropriate prompts, these models assisted in identifying semantically similar pairs of family definitions (Technical implementation details in Appendix A.1 [32]).

The results, as depicted in Figure 3, revealed substantial overlap in definitions, with Llama 3 identifying 837 similar pairs, Gemma identifying 1,095 pairs, and Llava identifying 637 pairs. Notably, 42 pairs were identified as similar by all three models. To ensure the accuracy of the findings, we conducted manual verification of family pairs identified as similar by all three models, ultimately confirming 22 pairs of similar family definitions. The root cause of this overlap and redundancy in definitions lies in the absence of a unified protocol to regulate naming and definitions among security vendors. Consequently, each security provider strives to establish a comprehensive family classification system, leading to the proliferation of families and redundancy in definitions.

***Conclusion #2:** The observed overlap in family definitions underscores the need for standardization. The lack of a unified protocol has led to the repetition of definitions for similar malicious behaviors.*

B. Comparison of Malicious Behaviors

The inconsistency in family definitions ultimately leads to insufficient descriptions of malicious behaviors exhibited by the malware. To further investigate the performance of malicious families in describing malicious behaviors of malware, we conducted a comparative study between the behaviors included in family definitions and those possessed by the malware itself.

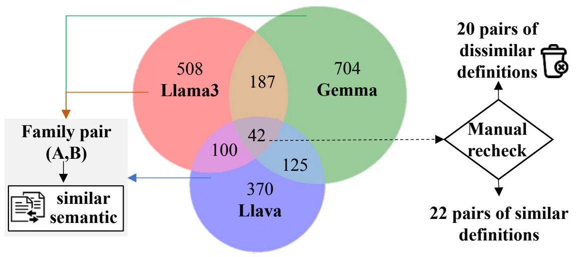
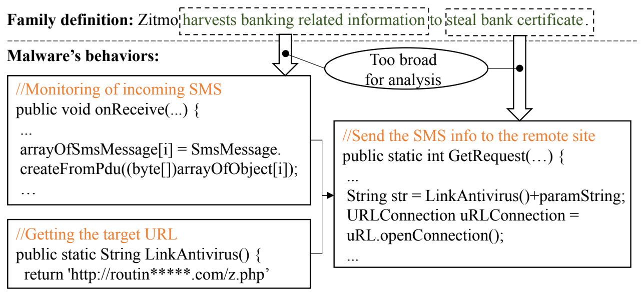


Fig. 3: Families with Varied Names but Similar Definitions

1) Data Preparation: We employed the Drebin dataset, a widely used resource for Android malware analysis, containing 5,560 malicious applications from 179 malware families. By analyzing the number of malware in Drebin's malicious families, we found that the number of malware in each family in Drebin is very unbalanced; for example, the malware number of one family, fakeinst, reaches 805, while there are 39 families in Drebin that have a malware number of 1, and 25 families with a malware number of 2.

To focus our analysis on relevant families and malware, we filtered out 51 malicious families from the Drebin dataset that had more than 10 malware. From this subset, we started with the family names provided by Drebin and looked up common AV companies to see if there were definitions for the corresponding family. However, since there is no way to know which AV company originally defined the family, it is non-trivial to get a large number of authoritative family definitions


Family definition: Zitmo harvests banking related information to steal bank certificate.
Malware's behaviors:
Too broad for analysis

```
//Monitoring of incoming SMS
public void onReceive(...) {
 ...
 arrayOfSmsMessage[i] = SmsMessage.
 createFromPdu((byte[])arrayOfObject[i]);
 ...
```

```
//Getting the target URL
public static String LinkAntivirus() {
  return 'http://routin*****.com/z.php'
```

```
//Send the SMS info to the remote site
public static int GetRequest(…) {
 ...
 String str = LinkAntivirus()+paramString;
 URLConnection uRLConnection =
 uRL.openConnection();
 ...
```

Fig. 4: A Malware of Broader Definition. Leaking bank credentials by sending incoming SMS to a remote URL is much finer-grained than the definition of its family.

based on the family names.

We obtained definitions for 34 families from F-Secure [33] (We have explained the reasons for choosing F-secure in Appendix A.2 of [32]), which served as the basis for our analysis.

Based on the detailed malicious behaviors outlined in these family definitions, we summarized behavior labels to provide a concise representation. For example, for the "Batterydoctor" malware family, described as *"This trojan masquerades as Android Battery Doctor. Upon execution, it silently forwards information about the device to a remote location. At the same time, it tries to advertisements to display on the device"*, we assigned behavior labels such as "Disguise-Itself," "Ad," and "Info-Steal." These behavior labels allow for easy comparison with the behaviors exhibited by malicious applications while retaining essential information from the family definitions.

Furthermore, we conducted an automated analysis of the 4,536 malicious applications belonging to the 34 selected malware families leveraging *AMBL* (details in § IV-B). The analysis process, as illustrated in Figure 6, comprises three main parts:

- **Acquisition of Malicious Behaviors Mapped APIs**: Establishing the correspondence between malicious behaviors obtained from VirusTotal [34] and APIs.
- **Acquisition of Label-related Sensitive APIs**: Identifying the association between each behavior label defined by us and the related APIs.
- **Acquisition of Malicious Labels**: Generating malware labels for each malware using the obtained *Malicious Behaviors Mapped APIs* and *Label-related Sensitive APIs*.

Consequently, we conducted a thorough examination of the behaviors manifested by the malicious applications in contrast to the behavior labels attributed to their corresponding families. This examination culminated in the identification of two primary issues, which will be elucidated in the subsequent sections.

2) Broader Definition: Upon comparing the behaviors outlined in family definitions with those observed in malicious applications, we identified instances where certain definitions of malicious families were overly generalized. While the purpose of the malware was implied in the family definition, specific malicious behaviors were not explicitly articulated. In such cases, we categorized the malware as broadly defined by its family. For example, the definition of the Zitmo family states, "Zitmo harvests banking related information sent from bank to the user's device." While this conveys the intent of the malware family to steal banking-related information, it lacks specificity regarding the exact malicious behaviors involved in executing this attack. As illustrated in Figure 4, the attack chain of the malware "Criptomovil", belonging to the Zitmo, involves more detailed behaviors such as retrieving received SMS messages and transmitting them to a remote server.

Given the impracticality of manually analyzing a large volume of malicious applications and their families, we sought to automate the process of determining whether malware

TABLE II: Comparison of Definitions and Malware's Behaviors

Family name	Total number	Beyond definition	Broader definition	Same as behavior
Nandrobox	13	13 **(100%)**	13 **(100%)**	0 (0%)
Kmin	145	145 **(100%)**	145 **(100%)**	0 (0%)
Fakeapp	42	42 **(100%)**	42 **(100%)**	0 (0%)
Mobilespy	14	14 **(100%)**	14 **(100%)**	0 (0%)
Pjapps	54	54 **(100%)**	27 (50%)	0 (0%)
Fakenotify	63	63 **(100%)**	0 (0%)	0 (0%)
Faketimer	12	12 **(100%)**	0 (0%)	0 (0%)
Droidsheep	11	11 **(100%)**	0 (0%)	0 (0%)
Penetho	17	17 **(100%)**	0 (0%)	0 (0%)
Ginmaster	344	344 **(100%)**	0 (0%)	0 (0%)
Faketimer	12	12 **(100%)**	0 (0%)	0 (0%)
Yzhc	37	37 **(100%)**	0 (0%)	0 (0%)
Batterydoctor	130	126 (96.9%)	130 **(100%)**	0 (0%)
Plankton	632	600 (94.9%)	632 **(100%)**	0 (0%)
Vdloader	16	15 (93.8%)	16 **(100%)**	0 (0%)
Fakeangry	12	4 (33.3%)	12 **(100%)**	0 (0%)
Zitmo	14	3 (21.4%)	14 **(100%)**	0 (0%)
Spirmo	11	0 (0%)	11 **(100%)**	0 (0%)
Spyhasb	13	0 (0%)	13 **(100%)**	0 (0%)
Mobiletx	68	0 (0%)	0 (0%)	68 **(100%)**
Fakeplayer	17	0 (0%)	0 (0%)	17 **(100%)**
Droidkungfu	659	491 (74.5%)	655 (99.4%)	2 (0.3%)
Golddream	68	64 (94.1%)	64 (94.1%)	3 (4.4%)
Hipposms	16	15 (93.8%)	0 (0%)	1 (6.2%)
Lotoor	59	58 (98.3%)	0 (0%)	1 (1.7%)
Basebridge	326	82 (25.2%)	322 (98.8%)	2 (0.6%)
Boxer	150	36 (24%)	0 (0%)	114 (76%)
Fakeinst	805	191 (23.7%)	0 (0%)	614 (76.3%)
Lovetrap	11	8 (72.7%)	1 (9.1%)	3 (27.3%)
Iconosys	152	111 (73%)	15 (9.9%)	41 (27%)
Smsreg	20	18 (90%)	0 (0%)	2 (10%)
opfake	543	147 (27.1%)	0 (0%)	396 (72.9%)
Fakelogo	20	1 (5%)	0 (0%)	19 (95%)
Jifake	26	11 (42.3%)	0 (0%)	15 (57.7%)
Geinimi	16	14 (87.5%)	1 (6.3%)	2 (12.5%)
Total	4536	2747**(60.6%)**	2127**(46.9%)**	1300(28.7%)

TABLE III: Correspondence Table for Fine Granularity Behavior

Family definition	Fine granularity behavior
Bank [35]	SMS, Phone, Network
Advertisement [36]	Network
Steal user info [37]	File, Network
Control of device [38]	Overstep
Execute remote commands [39]	Overstep, Network

is broadly defined by its family. Initially, through extensive examination of numerous malicious family definitions, we identified five primary types of broad malicious behaviors: *Bank, Advertisement, Steal user info, Control of device, and Execute remote commands*. Subsequently, by thoroughly investigating the specific fine-grained malicious behaviors required to achieve these broad objectives [35]–[39] (The details are introduced in Appendix A.3 [32]), we established rules for Broader Definition, as outlined in Table III. These rules offer a more nuanced perspective on the malicious behaviors necessary to fulfill common broad attack objectives specified in family definitions.

If a family definition only provides broad descriptions of behaviors without detailed granularity, yet the associated malware exhibits corresponding fine-grained behaviors, we classify the malware as broadly defined. Leveraging the rules delineated in Table III, we conducted an automated analysis of malware in the Drebin dataset. As depicted in Table II, behaviors of 2,127 malicious applications (comprising 46.9% of the dataset) were broadly defined by their respective families. Notably, all malicious applications from 11 families were broadly defined, while none from 16 families exhibited Broader Definition. This underscores the importance of well-defined and detailed family definitions in mitigating the occurrence of Broader Definition.

Conclusion #3: *The study reveals inconsistencies in the granularity of malicious family definitions, particularly in their tendency towards Broader Definition. This lack of specificity impedes researchers' ability to dissect precise behavioral nuances within malware.*

3) Beyond Definition: Family definitions, in addition to Broader Definition, sometimes fail to fully encompass the malicious behaviors exhibited by individual malware, namely Beyond Definition. For example, as shown in Listing 1, the BatteryDoctor family definition includes behaviors like collecting user and device information and displaying advertisements. However, the malware "Cache Remover" within this family also engages in additional behaviors, such as sending SMS and accessing location information, which are not covered by the family definition. These behaviors, highlighted in red, indicate potential blind spots in detection and analysis.

To systematically investigate whether the malicious behaviors of malware extend beyond their family definitions, we utilized an automated method. If a malicious application's behaviors were neither explicitly stated nor implied by the family definition (as discussed in § III-B2), we classified them as exceeding the family definition. The findings, summarized in Table II, reveal that 2,747 malware samples (60.6% of the dataset) exhibited behaviors beyond those defined by their families. Notably, all malicious applications from 11 families displayed behaviors exceeding their respective definitions. These results underscore the limitations of current family definitions in capturing the full range of malicious behaviors, highlighting the need for more comprehensive and detailed definitions to improve detection and analysis.

```
//Relevant permissions extracted from AndroidManifest.XML
android.permission.INTERNET
android.permission.READ_CONTACTS
android.permission.READ_SMS
android.permission.WRITE_SMS
android.permission.SEND_SMS
android.permission.ACCESS_COARSE_LOCATION
android.permission.ACCESS_FINE_LOCATION

//Collect user and device information
private String getDeviceId(Context context) ...
public String getSale_amount() ...
private String getMyPhoneNumber(Context context) ...
public String getBrand_id() ...
private String getContactNumbers() ...
//Fetching advertisements and displaying them via notifications
private boolean is_online() ...
private String buildLink() {
    ...
    temp = ``http://" + this.brand_id + ``." + getDomain()
     + ``/serve?sdk=android&action=" + this.action +
     "&advertiser_id=" + this.brand_id + ``&key=" + ... + this.
     URLEnc.bytesToHex(this.URLEnc.encrypt(data));
    ...
}
public static void showUpgradeNotification(String paramString)
     ...

//Retrieving the latest location information
public boolean update_variables(long updatetime) {
    ...
    LocationManager lm = (LocationManager)this.context.
     getSystemService(``location");
    Location location = lm.getLastKnownLocation("gps");
    ...
}
//Behavior of sending SMS
public void onReceive(String m) {
    ...
    SmsManager smsManager = SmsManager.getDefault();
    smsManager.sendTextMessage(this.phone_number, null, message, null, null);
    ...
}
```

Listing 1: A Malware of Beyond Definition. In addition to possessing the malicious behaviors described by its family (in blue), the malware also acquires the user's latest location and sends SMS (in red), which is beyond the definition.

Conclusion #4: *As malware evolves, it often exhibits behaviors beyond those specified in existing family definitions, complicating the task of security analysts to fully understand and mitigate the attack chain.*

IV. Methodology

To address the identified issues, we propose a novel methodology of directly indicating malicious behaviors with behavior labels. By reviewing real-world malware implementation and family definitions, we summarized six types of general program behaviors (labels) associated with malicious behaviors. To mitigate the high cost of manual annotation, we implemented an **A**utomated **M**alicious **B**ehavior **L**abeling framework (*AMBL*). Considering the emerging malicious behaviors, we designed a novel prompting method leveraging LLM to keep the Label Related Sensitive APIs updated.

A. Malicious Behavior Labels

1) Malicious Label Summary: The lack of a universal standard for classifying specific malware behaviors poses

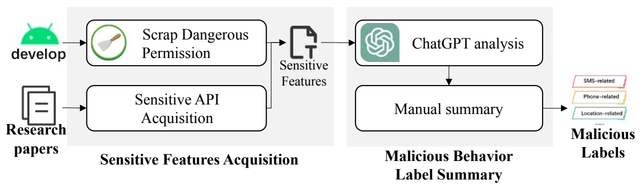


Fig. 5: The Workflow of Malicious Label Summary

significant challenges in malware analysis. Manually summarizing the vast range of malicious behaviors from numerous malware features is particularly arduous. To address this, we utilized large language models (LLMs) known for their robust summarization capabilities [40]. For the initial summarization of labels, our approach, illustrated in Figure 5, involves a method leveraging ChatGPT [41], which consists of two steps: sensitive features acquisition and malicious behavior label summarization.

*a) **Sensitive Feature Acquisition**:* The official Android development documentation provides the protected level for each permission. Permissions classified as dangerous often represent significant risks to user privacy and the operation of other applications. So, the 60 dangerous permissions defined by Android development documentation are used as part of sensitive features. Additionally, we compiled 193 sensitive APIs identified from existing research [10], [42], [43]. These permissions and APIs formed the basis of our sensitive feature set.

*b) **Malicious Behavior Label Summarization**:* Utilizing ChatGPT, we analyzed sensitive features to generate malicious behavior labels covering diverse malicious behaviors (refer to Appendix A.4 [32]). ChatGPT suggested ten labels: SMS Manipulation, Call Manipulation, Location Tracking, Device Data Retrieval, File System Access, Network Communication, User Data Access, System Modification, Security Bypass, and Application Control. Despite overlaps in some behaviors, ensuring label uniqueness (as required by Rule #1 in § IV-A3) led us to refine and summarize them. Behaviors like Device Data Retrieval, File System Access, and User Data Access aimed at accessing sensitive information were grouped as File-related. Additionally, System Modification, Security Bypass, and Application Control, Requiring Robust Permissions, were classified as Overstep-related. To achieve uniformity, we renamed the remaining labels without altering their meanings. Thus, the final six labels are: SMS-related, Phone-related, Location-related, File-related, Network-related, and Overstep-related.

2) Definition of Malicious Label: We have introduced here which specific malicious behaviors are associated with the 6 obtained labels respectively.

- **SMS-related** behaviors exploit SMS services for various attacks, such as privacy breaches and fraudulent subscriptions. Examples include transmitting personal data to remote servers, intercepting verification codes for authentication bypass, and secretly subscribing to premium services. Additionally, they can carry out control commands from remote servers and delete unread messages.
- **Phone-related** behaviors target telephone services, leading to privacy breaches and financial losses. They involve stealing user phone-related data like call logs and contacts and monitoring incoming and outgoing calls. For instance, ransomware may restrict victims to answering calls only to the infected device.
- **Location-related** behaviors leak user location information to remote servers, often for location-based attacks. Malware can also manipulate user location information through remote commands to facilitate further attacks.
- **File-related** behaviors involve accessing the device's file system, compromising user privacy. Malware can unauthorizedly access private files like photos and videos, transferring them to remote servers for extortion. Additionally, it may automatically download files, potentially creating backdoors for attacks.
- **Network-related** behaviors transmit user data from infected devices to remote servers through network connections. They can also receive control commands from remote servers to execute further malicious actions.
- **Overstep-related** behaviors bypass permission verification mechanisms, elevate privileges, and gain root access to the device. This enables the execution of commands that further compromise the device or user privacy.

3) Common Rules for Malicious Label: It is important to note that the labels defined in our study are derived from existing knowledge of dangerous permissions and sensitive APIs. As the malware continues to develop, additional malicious labels may be necessary to encompass newly identified malicious behaviors in the future. To address this, we have established the following three rules that need to be satisfied when defining malicious labels.

- **Rule #1:** The malicious behaviors represented by each label should be distinctive and that there should be no overlap between different labels
- **Rule #2:** All malicious behaviors exhibited by current malware can be captured by the defined labels.
- **Rule #3:** Each malicious label assigned to malware accurately reflects its malicious behavior and is not redundant.

B. AMBL Framework Construction

To address the high cost associated with manual annotation, we designed and open-sourced AMBL, which can automate the annotation of malicious labels and present a report containing comprehensive information adopted to determine the labels, which could be used in Android malware detection, such as training an ML-based detector.

As shown in Figure 6, with the help of the static analysis tool AndroGuard [44], we can obtain the static information (permissions, APIs, intents and CFG) of the malware (the implementation details can be referenced from the source code in the AMBL repository). Then the process of AMBL can be mainly divided into four steps: (1) Acquisition of

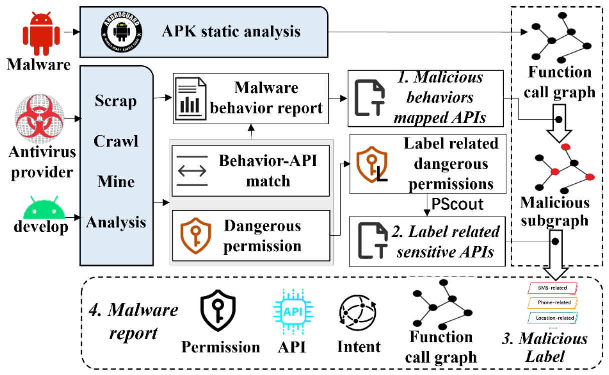


Fig. 6: An Overview of **A**utomated **M**alicious **B**ehavior **L**abeling Framework

Malicious behaviors mapped APIs, (2) Acquisition of Label related sensitive APIs, (3) Acquisition of Malicious Label and the output part, (4) Acquisition of Malware Report.

1) Acquisition of Malicious Behaviors Mapped APIs: In order to determine which API nodes in the malware's FCG are related to the malicious behaviors, we start with the official Android development documentation and malware behavior analysis reports to obtain the correspondence between the malicious behaviors and the used sensitive features, such as APIs. We first crawled behavior analysis reports of 1,000 malware from VirusTotal and discovered a total of 87 malicious behaviors. For each malicious behavior, we searched for keywords related to the behavior in the official Android development documentation and, upon manual confirmation, identified one or more APIs corresponding to the malicious behavior. Finally, the APIs mapped to each of the 87 identified malicious behaviors are compiled as the Malicious Behaviors Mapped APIs. With the assistance of Malicious Behaviors Mapped APIs, we can swiftly pinpoint crucial API nodes associated with malicious behaviors in the FCG of the malware based on the behavior analysis report.

2) Acquisition of Label Related Sensitive APIs: Label Related Sensitive APIs accounts for which sensitive APIs are associated with the pre-define malicious behavior labels. We obtain all dangerous level permissions from the official Android development documentation and manually match each permission with our defined malicious behavior label to obtain *Label related dangerous permissions*. Subsequently, based on the mapping of permissions to APIs provided by PScout [45] and other reseacher [46], the initial Label Related Sensitive APIs can be obtained. Considering that the matching between the permissions and the APIs provided by PScout may be outdated, We collect the APIs and utilize ChatGPT to assist in identifying APIs related to each of our malicious labels, thus getting the Malicious Label. The details will be introduced in § IV-C.

3) Acquisition of Malicious Label: Based on the *Malicious behaviors mapped APIs* and Label Related Sensitive APIs, the following operations are performed to obtain the Malicious

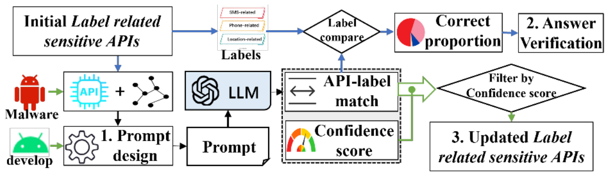


Fig. 7: Label Related Sensitive APIs Updating

Label of malware:

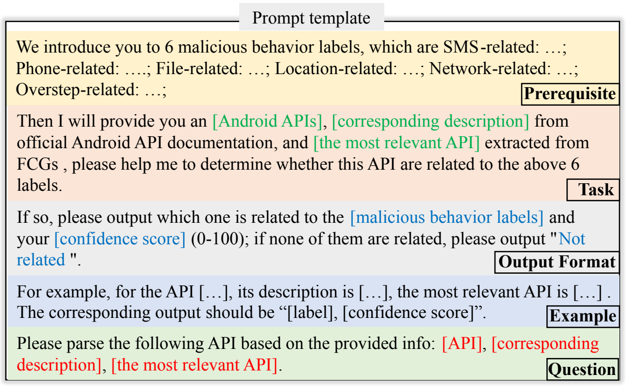
- **Malicious Subgraph**: We match the API nodes in the FCG with the APIs in the Malicious behaviors mapped APIs. Nodes that match are considered as central nodes, and we retrieve the central node and its first-degree adjacent nodes to form the Malicious subgraph.
- **Malicious Label of Malware**: We match the API nodes in the Malicious subgraph with the APIs in the Label Related Sensitive APIs. The corresponding labels are then assigned as the Malicious Label of the malware.

4) Acquisition of Malware Report: As shown in Figure 6, in the process of generating Malicious Label for malware in AMBL, the Malware Report we finally obtained includes various information such as function call graphs, permissions, APIs, Intents and Malicious Label. This information can serve as commonly used input features for ML-based Android malware detection methods and can be directly utilized by other classification tasks.

- **Permission** is a security mechanism primarily used to restrict the usage of certain functions with restrictive capabilities within applications and access between application components.
- **API** is the pre-defined function in the Android system, providing the ability for applications and developers to access a set of routines without needing to access the source code.
- **Intent** provides a mechanism to assist in the interaction and communication between activities, acting as an intermediary.
- **Function call graph** consists of nodes representing functions and edges indicating the call relationships between functions.

C. Label Related Sensitive APIs Updating

As described in § IV-B2, the initial Label Related Sensitive APIs was mapped through PScout based on dangerous permissions. However, PScout, developed 12 years ago, offers outdated mappings that fail to cover the latest sensitive APIs. To ensure that the Label Related Sensitive APIs encompasses the most recent sensitive APIs, we implemented an automated process using Large Language Models (LLMs). Our novel feedback-based prompting method enables *AMBL* to effectively update the Label Related Sensitive APIs, mitigating the impact of malware feature drifts [17]. The overall process, illustrated in Figure 7, involves three main steps: (1) Prompt Design, (2) Answer Verification, and (3) Acquisition of the Updated Label Related Sensitive APIs.



Prompt template

We introduce you to 6 malicious behavior labels, which are SMS-related: …; Phone-related: ….; File-related: …; Location-related: …; Network-related: …; Overstep-related: …;

Prerequisite

Then I will provide you an [Android APIs], [corresponding description] from official Android API documentation, and [the most relevant API] extracted from FCGs , please help me to determine whether this API are related to the above 6 labels.

Task

If so, please output which one is related to the [malicious behavior labels] and your [confidence score] (0-100); if none of them are related, please output "Not related ".

Output Format

For example, for the API […], its description is […], the most relevant API is […] . The corresponding output should be "[label], [confidence score]".

Example

Please parse the following API based on the provided info: [API], [corresponding description], [the most relevant API].

Question

Fig. 8: Prompt Template for API Updating

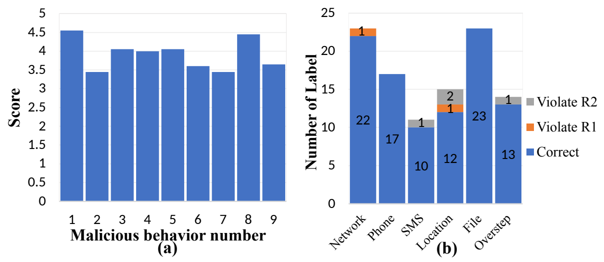


Fig. 9: Evaluation Results of (a) Correctness of Malicious Label and (b) Validation of Dataset Quality

1) Prompt Design: To leverage LLMs to identify whether an input API relates to defined malicious behaviors, we designed a detailed prompt template. This template, shown in Figure 8, aggregates various information aspects to enhance LLM understanding and ensure accurate labeling. The prompt template includes:

- **Prerequisite**: Provides foundational information, including malicious behavior labels and their definitions.
- **Task**: Instructs the LLM to determine if the given sensitive API relates to the defined malicious behavior labels. Information includes: API description from Android development documentation. The most relevant API in the Function Call Graphs (FCGs) of 1000 malicious applications.
- **Output Format**: Specifies that the LLM must output the corresponding malicious behavior label and a confidence score for its response.
- **Example**: Offers examples to guide the LLM in understanding the requirements.
- **Question**: Inputs the API for the LLM to evaluate and provide the corresponding malicious behavior label.

2) Answer Verification: Given that LLMs can sometimes provide unreliable responses for difficult tasks [47], we implemented a verification step. This process, shown by blue arrows in Figure 7, involves: (1) Randomly selecting 100 APIs with known labels from the initial Label Related Sensitive APIs. (2) Constructing prompts to query the LLM. (3) Comparing the LLM's generated labels with the ground truth from the initial Label Related Sensitive APIs to calculate the correct proportion. An 81% match rate indicated that the LLM's results were reliable. Additionally, the average confidence score for correctly matched APIs is 89.7, indicating that the LLM has a high level of confidence in its judgments.

3) Acquisition of the Updated Label Related Sensitive APIs: After confirming the credibility of LLM responses, we used the LLM to identify malicious behavior labels for APIs, as shown by the green arrows in Figure 7. The process includes: (1) Conducting a frequency analysis of sensitive APIs used by 1,000 malware applications. (2) Selecting the top 500 APIs for label identification. (3) Constructing and submitting prompts to the LLM for each API. (4) Choosing the average score of 89.7 that matches correctly in the § IV-C2 as the critical value. Filtering outputs by confidence scores, retaining only those with a score not less than 89.7 to ensure reliability. (5) This process resulted in the updated Label Related Sensitive APIs, encompassing the latest sensitive APIs and enhancing the detection and analysis of evolving malware behaviors.

D. Dataset

By leveraging the AMBL framework, we generated a new dataset of malicious applications with minimal human effort required for quality validation. Considering that commonly used malware datasets, such as Drebin (2010-2012) and Genome (2010-2011), are potentially outdated, we constructed a contemporary, ready-to-use dataset of malicious applications. This dataset includes malicious behavior labels and popular features summarized within a Malware Report. To ensure that the dataset reflects the current landscape of malware in the application market, we collected the latest 1,000 malware samples from VirusShare, with samples dated from 2022 onwards. Our dataset is publicly available to the research community, facilitating direct utilization by machine learning-based Android Malware Detection (AMD) research with minimal preprocessing effort. This accessibility aims to save time for subsequent researchers and streamline the research process. For more details on the dataset and to access the data, please refer to the dataset repository.

V. Evaluation

In this section, we validate the correctness of the Malicious Label in describing malicious behaviors and assess the quality of the dataset.

A. Validation #1: Malicious Label Correctness

To evaluate how effectively Malicious Label characterize behaviors in real-world malicious code snippets, we conducted an online survey[2].

1) Survey Preparation:

Survey data: We randomly selected 10 malicious applications from the dataset and generated malicious behavior labels using AMBL.

[2]See https://forms.gle/txdH5pfn4XjveyDJ8 for full survey questions.

Participant Recruitment: We recruited 25 participants from computer science-related disciplines at our university, with the majority being graduate students (24), including 7 individuals possessing prior experience in malware analysis. The participants' relevant academic backgrounds and specialized expertise contribute to the credibility and reliability of the questionnaire data.

2) Experiment Procedures: Participants received a brief introduction explaining the survey's objective: to assess the quality of the behavior labels generated by AMBL. They completed two tasks: (1) provided their educational background and experience in malware detection and program analysis, and (2) rated the match between the provided malicious behaviors, corresponding Malicious Label, and related code snippets on a scale from 1 to 5.

3) Results & Analysis: The average scores for each malicious behavior are shown in Figure 9(a). The average score for malicious behavior 1 (Downloading files from third-party websites to the local device) was 4.55, indicating a high accuracy of our behavior labels. Malicious behavior 2 and 6 has the lowest average score of 3.45, which we will analyze in Appendix A.5 [32]. Except for malicious behaviors 2 and 6, all other behaviors scored above 3.5, suggesting that the labels generated by AMBL are generally reliable. The overall average score for all malicious behaviors was 3.92, suggesting that our Malicious Label system performs better than acceptable and approaches good reliability.

B. Validation #2: Dataset Quality

To verify the credibility of the dataset, we conducted the following validation.

1) Selection of Malicious Samples: Manually analyzing a large number of malware samples to identify their malicious behaviors is impractical due to the immense effort involved. Therefore, we selected a representative subset of malware samples. To ensure comprehensive validation, each label must be verified by at least 10 samples. Based on this criterion, we randomly selected 103 malicious labels of 23 malware for detailed analysis.

2) Verification of Malicious Label for Selected Malware: We used Dex2Jar to decompile the selected malware and analyze the source code. We start from sensitive APIs and analyze the complete attack chain of malware based on function call relationships. After extracting the malicious behaviors, we match them with the behavioral labels of the malware to determine whether the assigned labels for the sample are appropriate. We provide detailed analysis examples in Appendix A.6 [32]. For incomplete matches, we conducted additional analysis to determine the reasons, focusing on two reasons: (1) The malware exhibited malicious behaviors not specified by the labels. (2) AMBL provided extra malicious labels without corresponding malicious behaviors.

3) Result & Analysis: Our analysis, shown in Figure 9(b), indicates that 97 out of 103 malicious labels (94.2%) had corresponding malicious code snippets, suggesting high reliability. However, two issues were identified: **Missing Labels (Reason 1)**: Two malware samples exhibited additional malicious behavior not captured by AMBL due to incomplete malware analysis reports from AV providers. We plan to source more comprehensive reports to enhance label accuracy. **Extra Labels (Reason 2)**: Four malware samples were assigned extra labels despite not exhibiting corresponding behaviors, often due to benign use of sensitive information by normal package whthin repackaged malware. To address this, we will incorporate techniques for detecting repackaged malware (such as [48]) to reduce false positives in future work.

VI. Discussion

This work reveals the inadequacies of family classification as a basis for further analysis of malware and also proposes its own solution starting from malicious behavior labels. however, there exists some threats that need to be discussed. Firstly, while our work does not resolve all problems related to family classification, we hope it encourages both academic and industrial researchers to reassess its utility. Such a reassessment could drive the development of more robust methodologies for Android malware analysis. Another threat to our work lies in validating the completeness of the summarized labels. Our work predominantly included limited researchers and data, which may only provide insightful directions but not a comprehensive solution. In future studies, we hope to attract more experts in Android malware analysis to participate in this study. Besides, our defined labels may become obsolete with the emergence of new malicious behaviors that have not been identified. To address this, we plan to design an open-access platform that allows more people to regularly contribute unlisted malicious behavior labels to maintain their relevance and comprehensiveness.

VII. Conclusion

This paper presents a study that summarizes the current issues with family definitions and introduces six malicious behavior labels to directly indicate the malicious behavior of malware. Then we designed the AMBL framework to automatically generate malicious labels for malware, and constructed an open-source dataset.

Acknowledgment

This research was funded by the National Natural Science Foundation of China (No.62332005).

References

[1] (2024) Android and Google Play statistics. [Online]. Available: https://www.appbrain.com/stats

[2] (2024) MALWARE AND PUA statistics. [Online]. Available: https://portal.av-atlas.org/malware

[3] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A Performance-Sensitive Malware Detection System Using Deep Learning on Mobile Devices," *IEEE Transactions on Information Forensics and Security*, 2020.

[4] R. Feng, S. Chen, X. Xie, L. Ma, G. Meng, Y. Liu, and S.-W. Lin, "MobiDroid: A Performance-Sensitive Malware Detection System on Mobile Platform," in *2019 24th International Conference on Engineering of Complex Computer Systems*, 2019.

[5] R. Feng, J. Q. Lim, S. Chen, S.-W. Lin, and Y. Liu, "SeqMobile: An Efficient Sequence-Based Malware Detection System Using RNN on Mobile Devices," in *2020 25th International Conference on Engineering of Complex Computer Systems*, 2020.

[6] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, 2018.

[7] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu, "Android malware familial classification and representative sample selection via frequent subgraph analysis," *IEEE Transactions on Information Forensics and Security*, 2018.

[8] Y. Bai, Z. Xing, X. Li, Z. Feng, and D. Ma, "Unsuccessful story about few shot malware family classification and siamese network to the rescue," in *International Conference on Software Engineering*, 2020.

[9] Y. Li, D. Yuan, T. Zhang, H. Cai, D. Lo, C. Gao, X. Luo, and H. Jiang, "Meta-learning for multi-family android malware classification," *ACM Transactions on Software Engineering and Methodology*, 2024.

[10] Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, "Multi-label classification for android malware based on active learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[11] S. Li, X. Xie, Y. Lin, Y. Li, R. Feng, X. Li, W. Ge, and J. S. Dong, "Deep learning for coverage-guided fuzzing: How far are we?" *IEEE Transactions on Dependable and Secure Computing*, pp. 1–13, 2022.

[12] B. Cheng, S. Zhao, K. Wang, M. Wang, G. Bai, R. Feng, Y. Guo, L. Ma, and H. Wang, "Beyond fidelity: Explaining vulnerability localization of learning-based detectors," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 5, Jun. 2024.

[13] H. Dongqi, W. Zhiliang, F. Ruitao, J. Minghui, C. Wenqi, W. Kai, W. Su, Y. Jiahai, S. Xingang, Y. Xia, and L. Yang, "Rules refine the riddle: Global explanation for deep learning-based anomaly detection in security applications," in *Proceedings of the 31st ACM conference on Computer and communications security*, 2024, p. 95–106.

[14] S. Li, M. Ge, R. Feng, X. Li, and K. Y. Lam, " Automatic Detection and Analysis towards Malicious Behavior in IoT Malware ," in *2023 IEEE International Conference on Data Mining Workshops (ICDMW)*, Los Alamitos, CA, USA, Dec. 2023, pp. 1332–1341.

[15] S. Y. Yerima, S. Sezer, G. McWilliams, and I. Muttik, "A new android malware detection approach using bayesian classification," in *International Conference on Advanced Information Networking and Applications*, 2013.

[16] Y. Li, R. Feng, S. Chen, Q. Guo, L. Fan, and X. Li, "Iconchecker: Anomaly detection of icon-behaviors for android apps," in *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*, 2021, pp. 202–212.

[17] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1127–1144.

[18] S. Wu, P. Wang, X. Li, and Y. Zhang, "Effective detection of android malware based on the usage of data flow apis and machine learning," *Information and software technology*, 2016.

[19] S. Liu, W. Ma, J. Wang, X. Xie, R. Feng, and Y. Liu, "Enhancing code vulnerability detection via vulnerability-preserving data augmentation," in *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, ser. LCTES 2024. Association for Computing Machinery, 2024, p. 166–177.

[20] X. Li, S. Liu, R. Feng, G. Meng, X. Xie, K. Chen, and Y. Liu, "Transrepair: Context-aware program repair for compilation errors," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22. Association for Computing Machinery, 2023.

[21] Q. Guo, X. Li, X. Xie, S. Liu, Z. Tang, R. Feng, J. Wang, J. Ge, and L. Bu, "Ft2ra: A fine-tuning-inspired approach to retrieval-augmented code completion," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2024. Association for Computing Machinery, 2024, p. 313–324.

[22] B. Wu, S. Liu, R. Feng, X. Xie, J. Siow, and S.-W. Lin, "Enhancing security patch identification by capturing structures in commits," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–15, 2022.

[23] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "Avclass: A tool for massive malware labeling," in *Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.

[24] R. Feng, S. Li, S. Chen, M. Ge, X. Li, and X. Li, "Unmasking the lurking: Malicious behavior detection for iot malware with multi-label classification," in *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2024, p. 95–106.

[25] G. Meng, R. Feng, G. Bai, K. Chen, and Y. Liu, "Droidecho: An in-depth dissection of malicious behaviors in Android applications," *Cybersecurity*, 2018.

[26] F. Maggi, A. Bellini, G. Salvaneschi, and S. Zanero, "Finding non-trivial malware naming inconsistencies," in *International Conference on Information Systems Security*, 2011.

[27] (2024) Introducing Malpedia. [Online]. Available: https://malpedia.caad.fkie.fraunhofer.de/

[28] (2024) Introducing CheckPoint. [Online]. Available: https://www.checkpoint.com/

[29] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[30] G. Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love *et al.*, "Gemma: Open models based on gemini research and technology," *arXiv preprint arXiv:2403.08295*, 2024.

[31] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, 2024.

[32] (2024) Appendixes of this work. [Online]. Available: https://sites.google.com/view/behavior-speaks-loader/home

[33] (2024) Introducing F-secure. [Online]. Available: https://www.f-secure.com/en

[34] "VirusTotal". [Online]. Available: https://www.virustotal.com

[35] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "An empirical analysis of android banking malware," in *Protecting Mobile Networks and Devices*. Auerbach Publications, 2016, pp. 223–246.

[36] P. Jyotiyana and S. Maheshwari, "A literature survey on malware and online advertisement hidden hazards," *Intelligent Systems Technologies and Applications 2016*, pp. 449–460, 2016.

[37] T.-E. Wei, A. B. Jeng, H.-M. Lee, C.-H. Chen, and C.-W. Tien, "Android privacy," in *2012 international conference on machine learning and cybernetics*, vol. 5. IEEE, 2012, pp. 1830–1837.

[38] C. Kotkar and P. Game, "Prevention mechanism for prohibiting sms malware attack on android smartphone," in *2015 annual IEEE India conference (INDICON)*. IEEE, 2015, pp. 1–5.

[39] K. Alfalqi, R. Alghamdi, and M. Waqdan, "Android platform malware analysis," *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2015.

[40] J. Kim, J. Nam, S. Mo, J. Park, S.-W. Lee, M. Seo, J.-W. Ha, and J. Shin, "Sure: Summarizing retrievals using answer candidates for open-domain qa of llms," *arXiv preprint arXiv:2404.13081*, 2024.

[41] (2022) Introducing ChatGPT. [Online]. Available: https://openai.com/index/chatgpt/

[42] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, "Quick and accurate android malware detection based on sensitive apis," in *2018 IEEE international conference on smart internet of things (SmartIoT)*. IEEE, 2018, pp. 143–148.

[43] M. Fan, J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu, "Dapasa: detecting android piggybacked apps through sensitive subgraph analysis," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1772–1785, 2017.

[44] A. Desnos and G. Gueguen, "Androguard documentation," *Obtenido de Androguard*, 2018.

[45] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 217–228.

[46] C. Li, X. Chen, R. Sun, M. Xue, S. Wen, M. E. Ahmed, S. Camtepe, and Y. Xiang, "Cross-language android permission specification," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022, pp. 772–783.

[47] Q. Lu, L. Zhu, X. Xu, Z. Xing, and J. Whittle, "A framework for designing foundation model based systems," *arXiv preprint arXiv:2305.05352*, 2023.

[48] K. Tian, D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged android malware with code-heterogeneity features," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 1, pp. 64–77, 2017.