Convenience at a Cost: The Security Risks of Template-based Development in the App-in-App Ecosystem

Yizhe Shi, Zhemin Yang, Yifan Yang, Yunteng Yang, and Min Yang
Fudan University
{*yzshi23@m., yangzhemin@, yifanyang23@m., 22210240104@m., m_yang@*}*fudan.edu.cn*

Abstract—**Recently, many popular mobile applications, such as WeChat and Alipay, have evolved into super-apps, with numerous merchants tending to develop their own mini-apps to enhance user experience. To overcome the limited development capabilities of individual merchants, super-apps allow third-party service providers to create mini-apps for them using templates, which enable the rapid development of mini-apps for various merchants. However, this template-based development mechanism also introduces significant security concerns, as many mini-apps created using templates exhibit malicious behaviors.**

In this work, we present the first systematic study focused on assessing the malicious behaviors associated with mini-app templates. We design and implement a novel tool, MATEMINER, which adopts a three-stage clustering analysis and differential analysis to extract mini-app templates from a large dataset of mini-apps. Furthermore, we propose a pattern-based method to detect malicious behaviors. Finally, we apply MATEMINER to 2,282,096 mini-apps and identify malicious behaviors in 79,758 mini-apps and 4,642 mini-app templates. Our results reveal that the templates have been abused to facilitate the development of malicious mini-apps, with most malicious behaviors targeting user privacy. This study highlights the security risks associated with template-based development and provides a foundation for detecting and mitigating such threats. Furthermore, we have reported all our findings to the super-app platform.

1. Introduction

Recently, many popular mobile applications, such as WeChat and Alipay, have evolved into super-apps, providing platforms where mini-apps can efficiently serve users. Due to the extensive user base of super-apps (e.g., 1.1 billion monthly active users in WeChat [1]), an increasing number of merchants are developing their own mini-apps to attract customers and boost profits. However, many merchants face challenges due to limited development capabilities. To address this, super-app platforms enable third-party service providers to assist massive merchants in developing mini-apps with templates, which can significantly accelerate the development process. By calculating the usage statistics from the WeChat service market [2], we find that over 480 thousand mini-apps have been developed using templates.

Security Risk of Mini-app Templates. However, this novel development paradigm, while facilitating legitimate development, also supports malicious activities. First, mini-app templates may be intentionally designed to enable harmful activities, such as stealing user data. The malicious behaviors embedded in these templates can propagate across numerous mini-apps developed with them. Second, mini-app templates can be abused by developers to create malicious mini-apps. For example, we found that a mini-app template is abused to develop 496 malicious mini-apps that impersonated official mini-apps for selling movie tickets. This abuse can significantly accelerate the development process of malicious mini-apps.

Previous studies have found that many malicious applications are developed using app generators with pre-designed templates [3], [4], [5]. Chen et al. [3] found that the online app generators have been abused to facilitate development of scam applications. Unlike online app generators, which have distinct features and are mostly free for users, mini-app templates lack clear features and are often paid services. This makes it difficult to identify both the mini-app templates and the mini-apps developed using them, thereby hindering further analysis. To the best of our knowledge, no study has investigated the malicious behaviors in template-based mini-apps and their impacts on the app-in-app ecosystem.

Our Work. In this paper, we perform the first in-depth analysis to assess the **m**alicious beh**a**viors in **te**mplate-based mini-apps (denoted as MATE). Our research aims to answer two key research questions: the status quo of malicious mini-app templates and the abuse of templates in malicious mini-app development. However, this is not a trivial task, and there are several challenges. First, it is hard to identify mini-app templates as they are typically closed-source and paid services. Besides, there are no obvious features to distinguish mini-apps developed with the same template. Second, it is difficult to determine whether the malicious behaviors are introduced by mini-app templates, as the template code and user-defined code are tightly coupled in mini-apps.

To this end, this paper designs and implements a novel tool, called MATEMINER, for analyzing MATE risks at a large scale. First, MATEMINER proposes a three-stage clustering analysis to identify mini-apps developed using the same templates, based on the observation that mini-apps developed from the same templates exhibit a high degree of code similarity. Specifically, the analysis is conducted at three levels of granularity: page structure, function, and call chain, to effectively cluster the mini-apps. Second, MATEMINER performs a fine-grained differential analysis to extract mini-app templates from template-based mini-apps. MATEMINER employs a majority-based decision algorithm to separate template code and user-defined code. Finally, MATEMINER conducts an in-depth behavior analysis for detecting malicious behaviors. MATEMINER models various malicious behaviors and their associated patterns, enabling the detection of different malicious types (Section 3.3).

Our Findings. We implemented the prototype of MATEMINER and applied it to 2,282,096 real-world mini-apps. As a result, MATEMINER identifies 15,143 unique mini-app templates, which collectively contain 268,363 mini-apps. Our results reveal that malicious mini-app templates are widespread, with some templates being specifically designed for malicious mini-app development. Specifically, MATEMINER identifies 4,642 malicious mini-app templates, which are used to develop a total of 52,926 mini-apps. Besides, many templates are abused to facilitate the development of malicious mini-apps. Among 268,363 template-based mini-apps, 79,758 are detected with malicious behaviors, associated with 7,219 mini-app templates. Additionally, malicious mini-app developers leverage mini-app templates to rapidly develop malicious mini-apps. Specifically, 286 mini-app developers create malicious mini-apps with the same templates within one week once the previously developed one is removed by super-app. Upon further analysis of these malicious mini-apps, we find that many mini-apps employ evasion strategies, such as changing their names, to bypass super-app's vetting mechanisms. These findings reveal the pervasive security risks associated with the template-based development and we have reported all our findings to the super-app platform.

In short, we make the following contributions.

- To the best of our knowledge, our work is the first to construct the dataset of template-based mini-apps and systematically study the malicious behaviors associated with them.
- We propose a novel approach, called MATEMINER, to automatically assess the MATE risks at a large scale. Besides, we will open-source our code for future researches.
- We have evaluated MATEMINER with 268,363 template-based mini-apps and identified malicious behaviors in 4,642 mini-app templates and 79,758 template-based mini-apps. Our analysis highlights the security risks associated with mini-app templates.

2. Study of Mini-app Template

2.1. Template-based Mini-app Development

Due to the limited development capabilities of merchants, super-app platforms enable third-party service providers to assist massive merchants in developing mini-apps with templates, which offer similar user interfaces and functionalities. Unlike traditional development process, mini-app templates can be reused to develop different mini-apps, which significantly accelerate the development process. To understand the template-based development within the app-in-app ecosystem, we conducted an in-depth analysis of the developer documentation provided by super-apps. This development process consists of two key steps:

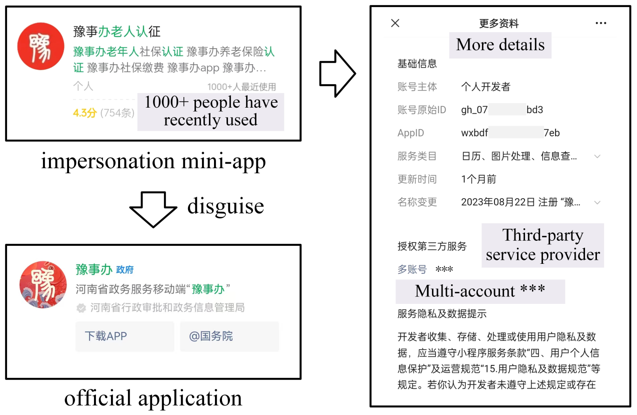
Authorization. To manage the delegation of development tasks and restrict the actions third-party service providers can perform, super-app platforms introduce a permission-based mechanism. Specifically, service providers must obtain the authorization of specific permissions from merchants before proceeding with the development process (e.g., permission 18 in WeChat [6]). The authorization process is based on OAuth 2.0 protocol [7], where the service provider generates an authorization link through the super-app platform to request development permissions from the merchant. Once the merchant grants consent, the service provider gains the permissions to proceed with mini-app development. Notably, these permissions can be revoked by the merchant at any time.

Template-based Development. During the development process, service providers can deliver similar development services to multiple merchants with mini-app templates. A mini-app template is functionally equivalent to a regular mini-app, containing necessary user interfaces and code logic. Additionally, to meet the merchant's unique needs, service providers can tailor both the front-end and back-end components of mini-apps. For the front-end, this typically involves updating the mini-app's name, description and customizing the user interface to align with the merchant's preferences. For the back-end, customization focuses on more complex requirements, such as session management and user data handling, ensuring that the mini-app's functionality supports the merchant's business needs.

2.2. A Motivating Example

While the template-based development significantly shortens the development cycle and brings convenience for mini-app development, it also introduces security risks. In particular, mini-app templates can be abused by malicious developers to develop malicious mini-apps. Additionally, although official marketplaces (e.g., WeChat service market [2]) offer vetted mini-app templates for merchants to purchase, many third-party service providers host their own templates on independent websites, such as [8], [9], [10]. Mini-app templates provided by third-party marketplaces can bypass the super-app platform's review process, allowing malicious service providers to distribute malicious mini-app templates. This exposes merchants and mini-app users to significant risks, as unvetted templates may introduce malicious behaviors.

To demonstrate the security risks, we highlight an example of a template-based malicious mini-app (wxbdf***7eb)



豫事办老人认征
豫事办老年人社保认证 豫事办养老保险认证 豫事办社保缴费 豫事办app 豫事办…
个人　1000+人最近使用
4.3分 (754条)

豫事办 政府
河南省政务服务移动端“豫事办”
河南省行政审批和政务信息管理局
下载APP　@国务院

更多资料

基础信息

账号主体	个人开发者
账号原始ID	gh_07[redacted]bd3
AppID	wxbdf[redacted]7eb
服务类目	日历、图片处理、信息查…
更新时间	1个月前
名称变更	2023年08月22日 注册 “豫…

授权第三方服务

多账号 ***

服务隐私及数据提示

开发者收集、存储、处理或使用用户隐私及数据，应当遵守小程序服务条款“四、用户个人信息保护”及运营规范“15.用户隐私及数据规范”等规定。若你认为开发者未遵守上述规定或存在

(a) a malicious mini-app example　(b) detailed information

Figure 1: A motivating example of malicious mini-app.

in Figure 1. This mini-app disguises itself as an official government service, using a misleading name and icon to appear legitimate. Its description claims to provide services such as social insurance and old-age pension, deceiving users into using the mini-app. Notably, more than 1,000 people have recently used this mini-app. Once users interact with the mini-app, they are prompted to enter personal information and click on malicious links. As a result, the mini-app can steal sensitive personal data and commit fraud.

Upon further analysis of the mini-app's detailed information (Figure 1-b), we find that it was developed by an individual developer who granted development permission to a third-party service provider, i.e., “Multi-account ***”. Additionally, we identified 12 other mini-apps developed by this service provider from our dataset, all of which are created using the same mini-app template. This case highlights that, while the mini-app template facilitates mini-app creation, it also enables malicious mini-app developers to exploit mini-app templates for harmful behaviors.

2.3. Threat Model

During template-based mini-app development, multiple parties are involved. The super-app platform serves as a trusted entity, while mini-app developers (merchants) and third-party service providers potentially act as malicious actors. These malicious parties aim to develop and release malicious mini-app templates or mini-apps. As a result, two threat models exist in the template-based development.

Malicious Service Provider. In the first scenario, malicious behaviors are inherent in the mini-app templates. A malicious service provider can embed crafted code (“backdoors”) into the mini-app templates, facilitating malicious activities. These malicious behaviors can propagate to the mini-apps developed using these templates. For example, we identified that a mini-app template designed for express delivery services shared users' express information with third parties without informing users about the data collection and sharing practices. This affects 195 mini-apps developed using the template.

Malicious Merchant. In the second scenario, some merchants abuse templates to rapidly create malicious mini-apps, such as disguising the mini-apps as official ones to mislead users or broadcasting inappropriate content, which can result in data theft or economic loss. We have identified more than 70 thousand malicious mini-apps that are developed with templates. Worse still, in such cases, service providers may also act as mini-app developers. For example, we found that in 402 mini-app templates, the companies of the service providers and some mini-app developers are the same. This exacerbates the issues, forming a supply chain for malicious mini-apps, which poses a significant threat to the security of the app-in-app ecosystem.

2.4. Generalizing Malicious Behaviors

Before conducting our assessment, we extracted and categorized the types of malicious behaviors outlined in the official documentation of super-apps (e.g., [11], [12]) , supplemented by definitions from prior research [13], [14], [15], [16], [17]. Additionally, we conducted detailed case studies on the malicious mini-apps detected by super-apps [18] and previous work [17], [19], [20]. As a result, we identified and categorized four key types of malicious behaviors within the app-in-app ecosystem. Below, we present detailed descriptions of these behaviors and their associated security impacts.

Impersonation. Certain mini-apps disguise themselves as official applications by manipulating their names, descriptions, or icons, thereby misleading users into interacting with these fraudulent mini-apps. This deception can trick users into sharing personal information to malicious mini-apps or engaging in transactions with malicious entities, potentially leading to scams, data breaches, or financial loss.

Inducement. Some mini-apps use incentivized sentences, such as promises of monetary rewards, to entice users into taking specific actions, such as inviting others to use the mini-app or participating in certain activities. However, mini-app users may not receive the promised rewards after completing the required actions. Such behaviors are categorized as incentivized sharing or rogue malware in [17], which includes actions such as inducing users to share the mini-app with friends. Given that super-app platforms host billions of users [1], inducement mini-apps can leverage mini-app users to spread the mini-app across their social networks and potentially be used to facilitate other malicious behaviors, such as data theft.

Inappropriate Content. Some mini-apps disseminate illegal or explicit content, such as pornography, hate speech, or racial discrimination. This exposes users, especially teenagers, to harmful materials that can negatively impact their well-being and safety. Moreover, this type of malicious behavior is particularly concerning in mini-apps due to their widespread accessibility and the lack of effective age verification mechanisms.

Data Theft. Some mini-apps collect personal information without disclosing the data practices. This compromises user's right to transparency and control over their personal data, as outlined in privacy regulations such as the GDPR [21] and CCPA [22]. Data theft is prevalent in mobile applications and mini-apps, and has been widely studied in prior work [17], [23], [24], [25]. It is also referred to as "stealth collection" in some studies [17].

3. Our Design

In this work, we propose a novel approach, named MATEMINER, to detect the malicious behaviors associated with mini-app templates. As discussed in Section 1, there are several challenges. First, mini-app templates are typically closed-source and paid services, making it difficult to identify both mini-app templates and mini-apps developed using the same template. Second, it is hard to differentiate between the behaviors inherent to the templates and those customized by mini-apps, as the template code is tightly coupled with user-defined code in template-based mini-apps. To address these challenges, MATEMINER introduces a mini-app behavior analysis technique with three phases as illustrated in Figure 2.

First, MATEMINER conducts a three-stage clustering analysis to cluster mini-apps developed with the same templates, leveraging code features such as page structures and function call chains. Second, MATEMINER separates the template code from the user-defined code for the analysis of template behaviors. Specifically, MATEMINER employs a differential analysis to identify the customized and template parts of the mini-apps. During this process, MATEMINER performs an iterative analysis based on the inter-procedural control flow graph (ICFG) of mini-apps to reconstruct both inter- and intra-procedural dependencies within the mini-app templates. Third, to detect malicious behaviors, MATEMINER proposes a pattern-based behavior analysis approach to identify malicious behaviors in both templates and mini-apps. Specifically, MATEMINER models each type of malicious behavior, and tracks the mini-app behaviors to extract relevant features for the malicious behavior detection.

3.1. Template-based Mini-app Clustering

Before the clustering analysis, we first construct the dataset of template-based mini-apps. As discussed in Section 2.1, we can identify service providers based on the authorization information, which specifies the permissions granted to service providers, including the development permission. However, service providers may employ multiple templates to develop mini-apps. To identify mini-apps developed from the same template, which exhibit a high degree of code similarity, we perform a similarity analysis to cluster them. Specifically, we propose a three-stage clustering algorithm to efficiently and accurately group mini-apps based on three levels of granularity: page structure, function, and call chain.

Structure-based Clustering. In the first step, MATEMINER applies a hash algorithm to generate the signature for each

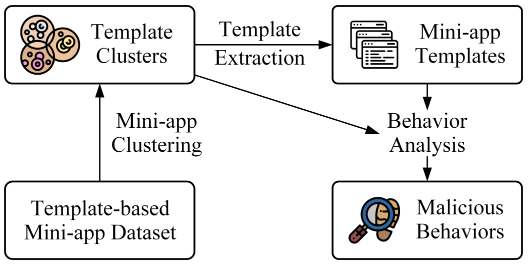
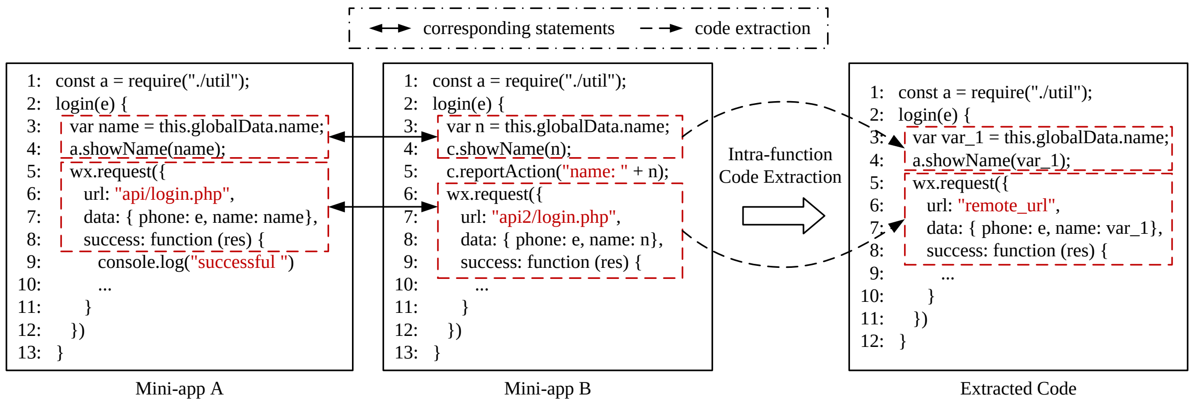


Figure 2: Workflow of MATEMINER.

mini-app based on the page structure, which encompasses all the paths of page files within the mini-app. The page information can be extracted from the `app.json` file, a global configuration file that defines page paths, bottom tabs, etc. [26], [27]. Since the page structures often remain unchanged during customization, MATEMINER calculates a similarity score among mini-apps and clusters mini-apps with identical or similar page structures. Specifically, the similarity score is calculated using the Levenshtein Distance, which efficiently measures the difference between mini-app structures. To select an appropriate algorithm, we compared Levenshtein Distance with Jaccard and cosine similarity algorithm, and found that Levenshtein Distance demonstrated better performance (as detailed in Appendix A). If the similar score is above a predetermined threshold, θ_1, MATEMINER decides that the two mini-apps are structure-wise similar. The threshold is configurable to suit different requirements, and its selection will be discussed in the evaluation section. As a result, this step establishes over-approximated relationships among mini-apps. Fine-grained analysis is then performed in the subsequent steps.

Function-based Clustering. In the second step, MATEMINER constructs the ICFG of mini-apps and analyzes the function-level similarities among mini-apps. Since the order of function definitions within different mini-apps may not be consistent, we analyze the similarity for each pair of functions between mini-apps. If a function is similar with multiple other functions that exceed the similarity threshold of θ_2, we select the function with the highest similarity as the final match. We then represent the similarity among mini-apps based on the proportion of matched functions relative to the total number of functions. Additionally, we construct a function-level similarity matrix, denoted as `functionMatrix`, to record these similarities. This matrix is of size $n \times n$, where n is the total number of mini-apps. Subsequently, we perform clustering analysis using the hierarchical clustering algorithm (HCA) to group mini-apps with high function-level similarity (θ_3), resulting in several clusters that represent mini-apps potentially developed from the same templates. HCA is an unsupervised machine learning algorithm that groups data into a tree-like structure of nested clusters and a widely used method for clustering datasets [28].

Call Chain-based Clustering. In the third step, MATEMINER further investigates the similarity of function call



```
const a = require("./util");
login(e) {
  var name = this.globalData.name;
  a.showName(name);
  wx.request({
    url: "api/login.php",
    data: { phone: e, name: name},
    success: function (res) {
      console.log("successful ")
      ...
    }
  })
}
```

```
const a = require("./util");
login(e) {
  var n = this.globalData.name;
  c.showName(n);
  c.reportAction("name: " + n);
  wx.request({
    url: "api2/login.php",
    data: { phone: e, name: n},
    success: function (res) {
      ...
    }
  })
}
```

```
const a = require("./util");
login(e) {
  var var_1 = this.globalData.name;
  a.showName(var_1);
  wx.request({
    url: "remote_url",
    data: { phone: e, name: var_1},
    success: function (res) {
      ...
    }
  })
}
```

Figure 3: Intra-function analysis during mini-app template extraction.

relationships among mini-apps. Specifically, MATEMINER analyzes whether similar functions share the same call relationships based on the function-level similarity matrix (`functionMatrix`) constructed in the second step. Next, MATEMINER analyzes mini-app similarity based on the call relations by counting the number of matched call edges, and we define the similarity threshold θ_4 to represent high similarity. Similarly, MATEMINER constructs an $n \times n$ similarity matrix based on call chain-level similarity and applies hierarchical clustering to refine the mini-app clusters.

After completing these phases of similarity analysis, MATEMINER clusters mini-apps potentially developed from the same template into a single group, enabling detailed analysis of each mini-app template.

3.2. Mini-app Template Extraction

After identifying mini-apps developed using the same template, MATEMINER performs template mining to extract mini-app templates from clustered mini-apps. The key insight is that mini-apps developed using the same template share identical template code. To achieve this, MATEMINER introduces a differential algorithm to identify the template code. Template extraction is conducted from two dimensions: inter-function and intra-function analysis, based on the ICFG of each mini-app. First, MATEMINER analyzes the call graphs of mini-apps to reconstruct the function call relations within the mini-app template. Second, MATEMINER analyzes the statements within each function based on the control flow graph to reconstruct the function bodies of mini-app templates. This approach preserves the code structures of mini-app templates and assists in further behavior analysis. Additionally, to facilitate template extraction, we select the central mini-app as the basis for extracting the template code. The central mini-app is the one with the highest similarity to other mini-apps within the cluster, making it a strong representative of the shared characteristics of the cluster.

Inter-function Analysis. During the inter-function analysis, MATEMINER constructs the function call relations of the mini-app template before recovering the function bodies. This process aims to extract the functions and their call relationships within the template. MATEMINER first selects an edge from the call graph of the central mini-app. For the two function nodes of this edge, i.e., $func_1$ and $func_2$, we utilize the function-level similarity matrix (`functionMatrix`), generated during the clustering process, to retrieve the corresponding functions $func_1'$ and $func_2'$ from other mini-apps. We then determine whether an equivalent edge exists between these functions in other mini-apps. If more than half of the mini-apps contain this edge, we consider that $func_1$ and $func_2$ are part of the mini-app template and share a call relation. The fine-grained analysis of the functions (e.g., $func_1$ and $func_2$) is conducted in the next step, and this process continues until the entire call graph is traversed. Furthermore, not all matched functions have identical names due to adjustments or customizations in the mini-app templates, as well as variations introduced by mini-app decompilation. In such cases, the function names from the central mini-app are used as the reference.

Intra-function Analysis. During the intra-function analysis, MATEMINER performs a fine-grained analysis to construct the function bodies within the mini-app templates. Our analysis begins with the control flow graph of each function within the same cluster. For each statement in the function, MATEMINER analyzes whether corresponding functions in other mini-apps contain similar statements. Specifically, MATEMINER first analyzes the types of these statements (e.g., `AssignmentExpression`, `VariableDeclaration`). If the types match, MATEMINER proceeds to analyze whether the statements convey the same semantics. The focus is on the variables within the statements. MATEMINER performs fine-grained data flow analysis to investigate data dependencies of these variables. Variables of different functions are considered related if they share common sources or dependencies, such as identical function parameters or values generated by the same functions. Additionally, some variables may exhibit implicit dependencies, particularly through getter/setter pairs, such as `setStorage` or `getStorage`. In these cases, MATEMINER models these implicit patterns to complement the variables' data dependencies.

We illustrate the process of extracting similar statements from two mini-apps in Figure 3. First, lines 3–4 in mini-app A correspond to lines 3–4 in mini-app B. Despite differences in variable names, both `name` and `n` refer to the same underlying content `this.globalData.name`. As a result, they are identified as similar, and an alias array is created, with `var_1` pointing to both `name` and `n`. Line 5 of mini-app B contains customized content without a corresponding statement in mini-app A, so it is excluded from extraction. In contrast, lines 5–8 in mini-app A align with lines 6–9 in mini-app B. In this case, the URL parameter is treated as customized content, and the constant value is normalized (e.g., as `remote_url`), with the original values recorded for reference. Additionally, through data dependency analysis, we determine that `name` in line 7 of mini-app A and `n` in line 8 of mini-app B reference the same content, ensuring semantic consistency.

When applied to multiple mini-apps, a majority-based decision strategy is used for template extraction. Specifically, if a statement appears in more than half of the mini-apps, it is included in the extracted template.

3.3. Malicious Behaviors Detection

In this section, our objective is to detect various types of malicious behaviors associated with mini-app templates. Previous work [17] adopts keyword-based methods for malicious behavior detection, which will introduce many false positives and false negatives, making them ineffective for accurate detection. Given the diverse nature of malicious behaviors, we design a modular approach tailored to each specific type. Since the mini-app template is functionally equivalent to a regular mini-app, we can apply similar methods to detect malicious behaviors in both the extracted mini-app templates and mini-apps. However, the impersonation behaviors are tailored to each mini-app, typically involving the disguise of the mini-app's name, description, or icon, which are not present in the mini-app template. Therefore, we do not detect such behaviors in mini-app templates. For the remaining three types of malicious behaviors, we use 'mini-app' as a general term to refer to both template-based mini-apps and mini-app templates for simplicity. Furthermore, this approach is scalable, allowing for the integration of additional malicious behaviors as detection needs expand.

3.3.1. Impersonation Detection. Impersonation mini-apps are designed to deceive users into believing that they are official mini-apps by employing misleading names or descriptions, as illustrated in Figure 1. To detect such mini-apps, we first construct a base dataset by collecting mini-apps developed by top companies [29], [30] that are likely to be impersonated. Then we propose an appearance-based similarity analysis to detect impersonation mini-apps, which focuses on the names, descriptions, and associated companies of mini-apps. Previous studies often calculate similarity over the entire content using classical methods, such as cosine similarity and edit distance [31], [32]. However, these methods do not focus on the principal content and are easily influenced by noise data. For example, in the description, "This mini-app is a Tencent mini-app that is used to sell goods", the most critical element is 'Tencent', leading users to assume the mini-app might be developed by 'Tencent'. Therefore, our approach involves extracting the principal content and dynamically weighting it to mitigate the influence of noise data.

Our key insight is that frequently used words across different mini-apps do not convey significant information about the mini-app. Instead, users tend to focus on distinctive words, which occur less frequently and are more representative of the mini-app. To compute word frequencies, we construct a database containing metadata from over 2 million mini-apps and use jieba [33] to perform text segmentation. Subsequently, MATEMINER dynamically assigns importance scores to different content based on the reciprocal of word frequency, with a total score of 1. Thus, the higher the word frequency, the lower the score. Then we perform similarity analysis between the principal content of base mini-apps and the mini-apps under test. The similarity score is calculated based on the importance scores derived from word frequency. Due to the page limit, we place the detailed algorithm in Appendix B.

TABLE 1: Common inducement patterns.

Category	Pattern	Example expression
share	share [prep] [noun]	share for gift
	share [noun] [verb] [noun]	share friends, earn points
	[noun] [verb] [prep] sharing	points earned after sharing
	sharing [aux] [verb] [noun]	sharing can earn points
invite	invite [noun] [prep] [verb] [noun]	invite your friends to gain points
	[verb] invitation [noun]	earn invitation points

3.3.2. Inducement Detection. Mini-apps often use incentive phrases, such as "share for gifts", to encourage user propagation, which primarily involves sharing and inviting actions (further details are provided in Appendix C). However, if a mini-app fails to deliver users rewards as promised, it is termed an inducement mini-app. While some may consider that a mini-app can be classified as an inducement mini-app based on the presence of inducement phrases, we adopt a more conservative definition in this paper.

To detect the inducement behaviors, we first identify components that contain inducement phrases, which often share similar patterns, such as 'share for gifts' or 'share for points', typically following the 'share [prep] [noun]' pattern. Therefore, we analyze and summarize the key patterns of inducement behaviors as illustrated in Table 1. Given the diversity of sentence structures and the limitations of simple regular expressions for accurately identifying such patterns, we employ part-of-speech matching techniques to detect different patterns. MATEMINER then analyzes the call sequences related to these inducement phrases to identify program behaviors. Since many program behaviors are linked to page navigation operations, such as `<navigator>`, we model and extract the navigation path and parameters to complement the analysis. Additionally, since the invitation process involves cross-user interaction, MateMiner analyzes mini-app behaviors that occur after a user receives an invitation. This analysis includes identifying the page to be launched and the associated parameters.

Finally, we aim to verify whether mini-apps distribute rewards as promised. Given that reward distribution occurs in the mini-app server side, it is unknown whether mini-app developers fulfill their promises. We employ a heuristic approach to determine whether the mini-app issues rewards. Our main insight is that, to issue a reward, the mini-app must transmit the user's information to the server and subsequently issue the reward to the user. To verify this, MateMiner analyzes the interaction between the mini-app client side and the server side, specifically looking for the transmission of user identifiers, such as user ID or phone number. If a mini-app does not interact with the server side or does not transmit a user identifier, we consider that it does not issue the reward.

3.3.3. Inappropriate Content Detection. Inappropriate content, including pornography and gambling, is strictly prohibited in super-app platforms. Extensive research has been conducted to detect such content [34], [35], [36]. Inspired by previous work, we employ advanced natural language processing techniques to analyze the text-based content for inappropriate content. A predefined list of terms related to inappropriate content is continuously updated to align with guidelines in super-app platforms. However, not all "inappropriate content" in mini-app code is malicious, and some are used for blacklist checks. Our key insight is that irrelevant content will not be rendered on the user interface. Therefore, we perform a renderability analysis of the text information. Specifically, we first analyze the mini-app code to locate the inappropriate content, and then analyze the corresponding layout file to determine whether the content can be rendered, which typically use the same variables for data binding.

3.3.4. Data Theft Detection. MateMiner achieves this module based on TaintMini [25] to track the collected personal data by mini-apps. To determine whether data collection complies with law requirements, previous work mainly focused on privacy polices [23], [24] or Apple's Privacy Labels and Google's Data Safety section [37], [38]. However, there is a lack of such guidance for mini-app templates. Therefore, we adopt a conservative strategy for mini-app templates. Specifically, MateMiner extracts and analyzes all the privacy policies of mini-apps within the same cluster, and determines whether the data collection practices of mini-app templates comply with the informed practices. Subsequently, MateMiner parses the privacy policy based on PolicyLint [39] to obtain the informed privacy practices of mini-apps. If the collected data is not explicitly disclosed, we consider it to be data theft.

4. Large-Scale Assessment

In this section, we aim to understand the whole picture of malicious behaviors in template-based mini-apps and the impact of template-based development on the app-in-app ecosystem. First, we introduce the experiment setup. Next, we evaluate the performance of MateMiner in mini-app template extraction and malicious behavior detection. Last, we present the overall results and discuss our findings.

4.1. Experiment Setup and Datasets

Our experiments were conducted on an Ubuntu 18.04 LTS 64-bit server with 64 CPU cores (2.3GHz) and 206GB of memory. In this paper, we focus on WeChat, the largest and most popular super-app platform, to analyze the current state of malicious behaviors associated with template-based mini-app development. WeChat hosts over 3.5 million mini-apps [40] and owns a mature mini-app template service market, which provides a rich dataset. Additionally, we discuss the generality of MateMiner in Section 5.

To evaluate our approach, we constructed the dataset from two sources. First, we collected data from WeChat service market, which includes 78 third-party service providers and 765 case mini-apps developed using on-sale templates. Second, we crawled the metadata of 2,281,953 mini-apps from the WeChat mini-app market. In total, the dataset contains 2,282,096 mini-apps. Then we filtered the mini-apps developed by third-party service providers based on the authorization information. Additionally, we excluded the service providers with fewer than three mini-apps, as such cases are insufficient for template analysis. Finally, we obtained 268,363 mini-apps developed by 5,203 third-party service providers. We then used MiniCrawler [41] to download these mini-apps.

We aim to answer the following research questions:

- RQ1: How effective is MateMiner in understanding and assessing the Mate risks?
- RQ2: What is the landscape of malicious mini-app templates in the wild?
- RQ3: How are mini-app templates abused by mini-app developers, and what are the security implications?

4.2. RQ1: Performance of MateMiner

MateMiner aims to assess the Mate risks in real-world mini-apps. Specifically, it aims to understand the status quo of malicious templates and the abuse of templates in malicious mini-app development. Therefore, we measure the effectiveness of MateMiner from two perspectives: the performance of template extraction and the performance of malicious behavior detection.

4.2.1. Performance in Mini-app Template Extraction. MateMiner extracts 15,143 templates from our dataset, which contain 268,363 mini-apps. There are two key components in template extraction: mini-app clustering and

TABLE 2: Performance of MATEMINER in template extraction.

Sample	CG-level						CFG-level					
	# TP	# FP	# FN	Precision	Recall	F1-score	# TP	# FP	# FN	Precision	Recall	F1-score
$Template_1$	96	4	2	96%	97.96%	96.97%	168	10	3	94.38%	98.25%	96.28%
$Template_2$	96	4	1	96%	98.97%	97.46%	135	21	0	86.54%	100%	92.78%
$Template_3$	100	0	3	100%	97.09%	98.52%	143	5	11	96.62%	92.86%	94.70%
$Template_4$	97	3	0	97%	100%	98.48%	170	26	23	86.73%	88.08%	87.40%
$Template_5$	96	4	3	96%	96.97%	96.48%	120	3	16	97.56%	88.24%	92.66%
$Template_6$	100	0	1	100%	99.01%	99.50%	180	32	3	84.91%	98.36%	91.14%
$Template_7$	98	2	1	98%	98.99%	98.49%	142	8	13	94.67%	91.61%	93.11%
$Template_8$	95	5	4	95%	95.96%	95.48%	68	1	5	98.55%	93.15%	95.77%
$Template_9$	99	1	3	99%	97.06%	98.02%	146	10	17	93.59%	89.57%	91.54%
$Template_{10}$	100	0	0	100%	100%	100%	198	11	24	94.74%	89.19%	91.88%
Total	977	23	18	97.70%	98.19%	97.94%	1470	127	115	92.05%	92.74%	92.39%

TABLE 3: Performance of MATEMINER across different phases in mini-app clustering.

Tool	# TP	# FP	# FN	Precision	Recall	F1-score
MATEMINER_s	331	181	5	64.65%	98.51%	78.07%
MATEMINER_{sf}	319	51	17	86.22%	94.94%	90.37%
MATEMINER	312	4	24	98.73%	92.86%	95.71%

template extraction. Therefore, the performance analysis is twofold. First, we demonstrate the necessity of each step of MATEMINER by investigating the metrics of different step combinations. Then, we measure the effectiveness of MATEMINER in extracting mini-app templates based on mini-app clusters.

Stepwise Contribution in Mini-app Clustering. In this section, we aim to evaluate the effectiveness of different phases in mini-app clustering. To build the ground truth, we randomly sample 10 service providers from our dataset and manually analyze their developed mini-apps to identify those developed with the same templates.

For the similarity threshold in different phases, the value of different thresholds have a coefficient effect on the metrics of the evaluation. Therefore, we conduct a grid search over different choices of them to establish the optimal values based on the F1-score, inspired by [42]. The thresholds are tested from 85% to 95%, with a step size of 1%. The false positive (FP) means that a mini-app is not developed with the same template as the central mini-app of the cluster. The false negative (FN) means that a mini-app in a different cluster is developed with the same template as the central mini-app. The final thresholds are as follows: θ_1 = 86%, θ_2 = 94%, θ_3 = 90%, and θ_4 = 92%.

The results are illustrated in Table 3. MATEMINER_s is a setup of MATEMINER using only structure-based analysis. MATEMINER_{sf} is a setup that incorporates both structure-based and function-based analysis. Adding function-based analysis leads to a significant decrease in false positives, although there is a minor increase in the false negatives. The false negatives are caused by mini-app developers customizing or replacing template functions, which leads to incorrect clustering of the corresponding mini-apps. Furthermore, applying callchain-based analysis results in better performance. The F1-score of MATEMINER is 95.71%, which means that our proposed method can effectively detect the mini-apps developed with the same mini-app templates.

Performance of Template Extraction. It is hard to evaluate the performance of template extraction due to the lack of original mini-app templates. To address this, we purchased 10 mini-app templates from the WeChat service market. Since the market lists some case mini-apps developed using each template, we are able to identify the mini-app clusters corresponding to the purchased templates. We manually analyze both the template code and the associated mini-app clusters to evaluate performance from two perspectives: the call graph level and the control flow graph level.

At the call graph level, we randomly select 100 call edges from the mini-app template to analyze false positives, and 100 call edges which are discarded during the extraction process to analyze false negatives. At the control flow graph level, we perform a fine-grained analysis of the detailed instructions within the functions and analyze the template code to establish the ground truth. Performance is then evaluated at the statement level.

As illustrated in Table 2, the average F1-score at the CG level is 97.94%, and at the CFG level, the average F1-score is 92.39%. False positives primarily arise from situations where unrelated statements with similar structures are identified as part of the template. Additionally, code obfuscation in mini-apps obscures the semantic meaning, making accurate extraction difficult. We further analyze the impact of code obfuscation and find that 13 out of 150 false positives are caused by it. Moreover, the average precision remains above 97% at the CG level and above 92% at the CFG level, indicating that code obfuscation does not significantly affect the overall performance. The static analysis tool struggles to handle such obfuscation. False negatives mainly result from the absence of some call edges during the construction of the ICFG. Furthermore, some mini-apps employ third-party development frameworks, such as Taro [43] and UniApp [44], which complicate the client-side code and hinder complete analysis.

4.2.2. Performance of Malicious Behavior Detection.

Before presenting the overall results of the malicious behavior detection, it is essential to evaluate the accuracy

TABLE 4: Performance of MATEMINER in malicious behavior detection in mini-app templates.

Malicious Behavior	# TP	# FP	# FN	Precision	Recall	F1-score
Impersonation*	-	-	-	-	-	-
Inducement	90	8	0	91.84%	100%	95.74%
Inappropriate Content	93	7	3	93%	96.88%	94.90%
Data Theft	98	2	10	98%	90.74%	94.23%
Total	281	17	13	94.30%	95.58%	94.93%

* Impersonation is a behavior of mini-apps, which is not in templates.

TABLE 5: Performance of MATEMINER in malicious behavior detection in template-based mini-apps.

Malicious Behavior	# TP	# FP	# FN	Precision	Recall	F1-score
Impersonation	91	9	4	91%	95.79%	93.33%
Inducement	90	10	0	90%	100%	94.74%
Inappropriate Content	91	9	8	91%	91.92%	91.46%
Data Theft	94	6	15	94%	86.24%	89.95%
Total	366	34	27	91.50%	93.13%	92.31%

of MATEMINER, which are shown in Table 4 and Table 5. MATEMINER achieves good performance, with an F1-score of 94.93% for mini-app templates and 92.31% for template-based mini-apps. Given that there is no ground truth, we evaluate false positives and false negatives by randomly selecting 100 mini-app templates and 100 mini-apps from those identified as malicious and non-malicious, respectively. If the count is less than 100, we used the entire dataset for evaluation. These samples are then manually reviewed by three domain experts, who are familiar with mini-apps and malware analysis, to identify various types of malicious behaviors. To ensure the accuracy and reliability of the labeling, a result was considered a true positive only if all experts reached a consensus. If there is any discrepancy in the labeling results, the three experts will have a discussion to reach their final agreement.

Evaluation with Manually Labeled Dataset. We first conduct an in-depth analysis of the detection results for different types of malicious behaviors with manually labeled datasets.

For the impersonation behaviors, we manually verify whether the detected mini-app impersonates the base mini-app. For mini-apps marked as non-malicious, we analyze them to determine whether they exhibit tendency to impersonate the base mini-app. Finally, the precision is 91%, and the recall is 95.79%. The false positives primarily arise because some mini-apps contain the principal content of base mini-apps but serve different meanings. For example, some mini-apps include terms like "Wanda" or "McDonald" to indicate locations, and in some cases, the principal content is short, e.g., less than three words, which can easily be included. As for the false negatives, they mainly arise because we pre-collected mini-apps from top companies, given that malicious mini-apps often target popular ones. However, some mini-apps impersonate official mini-apps that are not included in our list, resulting in four false negatives.

For the inducement behaviors, we manually examine the inducement patterns exhibited in the layout files, track the behaviors and monitor the network traffic. For mini-apps where inducement patterns are not detected, we manually analyze the mini-app code and analyze the false negatives of MATEMINER. There are no false negatives, and false positives primarily arise because static analysis does not fully recover the behaviors of mini-apps, e.g., reward distribution, which are mistakenly classified as malicious.

For the inappropriate content, we manually checked the samples and obtained an F1-score of 94.90% and 91.46%, respectively. False positives primarily arise because the meaning of content varies across different contexts. False negatives primarily result from two factors. First, some mini-apps contain dynamically loaded content, including illicit information which cannot be detected by our static analysis tool, leading to 5 false negatives. Second, the predefined list of inappropriate content is incomplete, resulting in 3 additional false negatives.

For the data theft, the assessment focuses on whether data collection occurs with user consent. False positives mainly arise from issues with the parsing of privacy policies. False negatives primarily result from incomplete data flows in mini-app templates or mini-apps, which may omit certain data collections.

TABLE 6: Performance of [17] in malicious behavior detection in template-based mini-apps.

Malicious Behavior	# TP	# FP	# FN	Precision	Recall	F1-score
Impersonation	-	-	-	-	-	-
Inducement	65	35	10	65.00%	86.67%	74.29%
Inappropriate Content	82	18	17	82.00%	82.83%	82.41%
Data Theft	70	30	20	70.00%	77.78%	73.68%
Total	217	83	47	72.33%	82.20%	76.95%

Comparison with Related Work. Then we compare MATEMINER with the state-of-the-art tool proposed by Yang et al. [17], which presents a systematic study on malicious mini-apps. Although the target of MATEMINER and [17] are different, a meaningful comparison can be made in terms of malicious behavior detection. We conduct the experiment on the same dataset as in Table 5 and the detection results of [17] are shown in Table 6. The average precision and recall of [17] are 72.33% and 82.20%, respectively, both of which are lower than those achieved by MATEMINER. This is primarily because [17] adopts keyword-based detection techniques, which are ineffective in identifying malicious behaviors that lack explicit textual features. Besides, the absence of program behavior analysis results in many false positives, such as that the inducement detection fails to analyze the reward behaviors of mini-apps, resulting in 30 false positives. Furthermore, [17] is incapable of detecting impersonation mini-apps, as the detection requires comparing the malicious mini-apps with official mini-apps to identify the impersonation behaviors.

Evaluation with Delisted Mini-apps. As mentioned in [17], [45], super-apps employ vetting mechanism to detect and delist malicious mini-apps. Besides, when accessing the delisted mini-apps, a notification would indicate the reason for removal, such as "Suspected inducement of sharing". All malicious mini-apps detected by MATEMINER have passed the super-app platform's vetting process prior to their release on the mini-app market. To further evaluate MATEMINER, we revisited the mini-apps in our dataset. As of now, 23,043 mini-apps have been removed from the market. We recorded these removed mini-apps along with their removal reasons to compile a list of mini-apps flagged as malicious. We then use them to evaluate MATEMINER. Since the reasons for removal do not fully align with the definitions of malicious behaviors, three experts manually analyze the reasons and map them to the corresponding malicious behaviors. If MATEMINER fails to detect a flagged mini-app, it is classified as a false negative.

However, we find that most of the reasons are vague, such as "No real service content" and "Suspected of violating relevant laws, regulations, and policies", which have also been mentioned by many mini-app developers [46]. Consequently, we collect 50 malicious mini-apps for evaluation, while the remaining ones could not be classified as specific type of malicious behaviors. MATEMINER successfully detected 42 (84%) of them. The undetected ones involve inappropriate content, which MATEMINER fails to identify due to dynamically loaded content and absence of evident features in the mini-app code. Additionally, 2 impersonation mini-apps target official mini-apps that are not present in our pre-collected dataset.

TABLE 7: Overview of malicious behavior detection result in mini-app templates.

Malicious Behavior	**Malicious Template**		**Associated Mini-app**	
	# template	% total	# mini-app	% total
Impersonation	-	-	-	-
Inducement	98	0.65%	1,282	0.48%
Inappropriate Content	797	5.26%	9,173	3.42%
Data Theft	4,245	28.03%	47,270	17.61%
Total	4,642	30.65%	52,926	19.72%

4.3. RQ2: Landscape of Malicious Templates

As shown in Table 7, we present the detection results for malicious mini-app templates, revealing a total of 4,642 malicious mini-app templates, with 52,926 mini-apps developed using them. Since impersonation is primarily a behavior of mini-apps, we do not detect such behaviors in templates. Among the malicious behaviors, data theft is the most prevalent. Specifically, 91.45% of malicious mini-app templates engage in data theft, primarily involving users' location and contact information. For example, we find a mini-app template stealthily sends users' location data and browsing history to third parties. Although fewer templates involve inducement behaviors, they are used to develop 1,282 mini-apps, which also pose significant risks.

Upon further analysis of the malicious behaviors in mini-app templates, we find most of the mini-app templates (4,532) are associated with the specific type of malicious behavior, such as data theft or inducement. Besides, our analysis reveals that several third-party service providers use mini-app templates specifically designed to generate particular types of malicious mini-apps. For example, we find that a template provided by the service provider "yi***Tech" includes an inducement phrase "share to earn money" in the template code. However, there is no code logic to reward users after sharing.

Finding I: *A significant number of mini-app templates exhibit malicious behaviors.*

Furthermore, we find that service providers may collaborate with malicious merchants. We conduct an analysis of the companies behind both third-party service providers and associated mini-app developers to uncover their relationships. Our analysis reveals that, in 402 mini-app templates, the companies of the service providers and some mini-app developers are the same. Besides, 28 service providers belong to related companies, which share the same official website or company legal representative as the mini-app developers. This suggests a tight relationship and potential collaboration between service providers and malicious merchants. Mini-app developers may exploit the template-based development mechanism and register as third-party service providers to facilitate the malicious mini-app development. This 'collusion' creates a 'supply chain' of malicious mini-apps. For example, a template developed by "hui***Tech" is used by the service provider to create malicious mini-apps, which exhibit inducement behaviors. Similarly, a template provided by "yixuan***" is used to develop malicious mini-apps, which involve data theft behaviors.

Finding II: *Many malicious third-party service providers double as developers of malicious mini-apps.*

TABLE 8: Overview of malicious behavior detection result in template-based mini-apps.

Malicious Behavior	**Malicious Mini-app**		**Associated Template**	
	# mini-app	% total	# template	% total
Impersonation	1,113	0.41%	346	2.28%
Inducement	1,872	0.70%	130	0.86%
Inappropriate Content	13,113	4.89%	970	6.41%
Data Theft	71,889	26.79%	6,725	44.41%
Total	79,758	29.72%	7,219	47.67%

4.4. RQ3: The Abuse of Mini-app Templates

Additionally, Table 8 presents the detection results for malicious mini-apps. Out of 268,363 successfully analyzed template-based mini-apps, 79,758 (29.72%) are found to exhibit malicious behaviors, originating from 7,219 mini-app templates. This indicates that templates are being abused to facilitate the development of malicious mini-apps. For example, we find that the mini-app developer “wenchi***” develops 18 malicious mini-apps with the same mini-app template, which is provided by the third-party service provider “jutui***”. These mini-apps disguise official mini-apps for selling movie tickets, such as Wanda Film and Maoyan Movie, which belong to Wanda Group and Meituan, respectively. Mini-app users may be misled into clicking these mini-apps and purchasing movie tickets, potentially leading to data leakage and financial loss. Besides, we find that 127 mini-app developers use more than one template to develop malicious mini-apps. Moreover, we observe that the number of malicious mini-app templates is less than the number of templates used to develop malicious mini-apps. This gap arises because mini-app developers also use benign templates to create malicious mini-apps.

Finding III: *Mini-app templates are significantly abused to develop malicious mini-apps.*

Among these malicious behaviors, data theft is the most prevalent, with many template-based mini-apps attempting to steal sensitive user data, such as phone number and user location. This is mainly due to that most mini-apps do not disclose the data collection practices or fail to specify the purpose for which user data is collected, thereby violating the law requirements. A significant proportion of mini-apps (83.7%) do not disclose to users that they collect privacy-related data, and many others fail to specify the purpose of data collection. This data highlights the serious prevalence of data theft in current mini-apps, revealing significant gaps in transparency and user consent practices. Impersonation and inducement behaviors are also common, leading to user confusion and potential exploitation, such as misleading mini-app users to input sensitive information. Additionally, many mini-apps contain inappropriate content, which poses significant risks, particularly for vulnerable user groups such as teenagers.

Moreover, to understand the security risks of data theft, we conduct a detailed analysis of the sensitive data collected by template-based mini-apps, as illustrated in Table 9. The most commonly collected privacy items are location data and user information. Additionally, to investigate data theft issues in template-based and non-template-based mini-apps, we randomly sampled 1,000 mini-apps from each category to compare their associated privacy risks. Among these, 259 template-based mini-apps and 82 non-template-based mini-apps were found to contain data theft.

Finding IV: *Malicious template-based mini-apps put sensitive user data at serious risk.*

Although super-apps implement measures to detect malicious behaviors [17], [45], the template-based development allows malicious mini-app developers to rapidly develop malicious mini-apps. To understand the development speed of malicious mini-apps using templates and the malicious behavior detection, we analyzed the registration times of malicious mini-apps developed using the same templates and found that, although some were delisted by the super-app, others, developed more recently, were not. Specifically, we discovered that 1,643 malicious mini-apps were registered later than those that were delisted. For example, a mini-app template provided by a third-party service provider (xincao***) was used to develop 12 malicious mini-apps. Of these, 2 mini-apps were delisted, while the remaining 10 were not. Furthermore, when analyzing malicious mini-apps developed by the same developers, we find that 286 mini-app developers create malicious mini-apps with the same templates within one week and the previously developed one has been removed. Besides, one mini-app developer quickly creates a similar mini-app (wx3e5***e61) just one day later, and the previously developed one (wxf36***fef) has been delisted by the super-app.

TABLE 9: Data theft detected in malicious mini-apps.

Item	# mini-app	**Item**	# mini-app
User info	62,828	Location	56,526
Media	44,279	Album	43,642
Address	26,011	Contact info	11,960
Microphone	5,038	Camera	3,490
File	2,554	Receipt	1,008
Bluetooth	872	Run data	658

Finding V: *Template-based development allows malicious mini-apps to evolve faster than the vetting process of WeChat.*

Moreover, we identify several evasion strategies employed by malicious mini-apps to bypass the vetting of super-apps. Yang et al. [17] found evasion strategies related to dynamic code execution and content rendering, which deliver malicious content after the malicious mini-apps pass vetting. In addition to these, we uncover several other evasion strategies. For example, some mini-apps initially register with legitimate names but later change them to misleading ones intended to deceive users, or frequently alter their names to evade detection. Besides, impersonation mini-apps often use homograph words, that share similar shapes or spellings (e.g., “0” and “O”), in their names or descriptions. This tactic can bypass similarity analysis that relies on original text. Additionally, many mini-apps use seemingly complete or legitimate workflows to carry out malicious behaviors, rather than relying solely on dark UI patterns. For example, in cases of inducement behavior, a malicious mini-app appears to carry out legitimate operations and send network requests. However, without in-depth code analysis, it is difficult to detect the underlying malicious behaviors.

To demonstrate the robustness of MATEMINER against evasion strategies employed by malicious mini-apps, we conducted an in-depth analysis. Inspired by [17], we collected 2,659 malicious mini-apps that adopt evasion strategies. MATEMINER successfully detected 2,335 of them. The undetected cases are primarily due to two reasons. Some mini-apps dynamically hide malicious content without distributing inappropriate content to the front-end, making it difficult for MATEMINER to detect such malicious behaviors through static analysis. Besides, due to the limitations of [25], certain privacy-violating data collection behaviors could not be accurately traced, resulting in these mini-apps to be misclassified as benign.

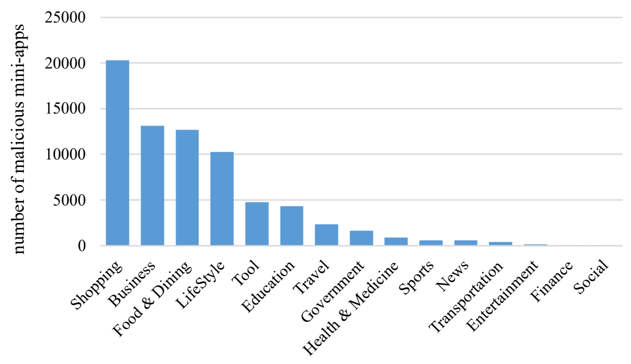


Figure 4: Distribution of malicious mini-app categories.

Distribution. We then analyze the distribution of malicious mini-apps across different categories to gain insights into the category information for each mini-app. Ultimately, we obtain category information for 72,083 malicious mini-apps, as illustrated in Figure 4. Our analysis reveals that categories frequently used by users, such as Shopping and Food, tend to have a higher number of malicious mini-apps. Interestingly, we also observe many malicious behaviors in categories like Government and Education. Upon manually reviewing these mini-apps and their associated categories, we find that some mini-apps are misclassified. For instance, 4,305 mini-apps are categorized under Education, even though the actual services of many mini-apps are unrelated to this category. For example, the category information for a template-based mini-app "earnMoney***" (wx7d8***5a1), which is an online shopping mini-app, is listed as "Infant and Young Child Education". This suggests that mini-apps may register under unrelated categories to bypass vetting, a practice commonly observed in impersonation-related malicious mini-apps.

5. Discussion

Ethical Considerations. Throughout the course of our research, ethical considerations are at the forefront. We carefully manage our research activities to ensure that they stay within legal and ethical boundaries. The research design carefully mitigates potential risks, ensuring that no harm is caused to users, mini-apps, super-app platforms, or any other involved parties. This whole research has been approved by our institution's IRB, and this study is considered "minimal risk" based on consultations with IRB staff. Furthermore, we have followed the best practices established by prior related work [3], [17], [47].

Given the widespread and serious nature of the malicious template-based mini-apps, we reached out to the super-app platform and have reported all our detected malicious mini-apps to the platform, which has acknowledged our findings. Till now, 23,043 malicious mini-apps have been removed. The super-app platform has understood the security hazards and taken some measures to detect malicious mini-apps. We believe it is feasible to integrate our detection methods into the workflow of the super-app platform, and we intend to broach this topic in future discussions with them.

In accordance with the Menlo Report, our study adhered to the ethical principles and no human-related data was used. Furthermore, all participants involved were thoroughly informed about the purpose and scope of the research, and explicit consent was obtained from each individual. Moreover, for the mini-app dataset collection, we followed the practices established in previous studies [17], [48], [49]. To avoid placing a burden on the super-app's servers, we adhered to the rate limits imposed by the super-app platform to ensure that we did not exceed these limits throughout the course of our study.

Generality of MATEMINER. While our research focuses on WeChat, the approach proposed in this paper is scalable. First of all, popular super-app platforms, such as Baidu, TikTok, and Alipay, follow similar development frameworks and adopt similar mini-app structures based on JavaScript. Besides, due to the structure similarity, mini-apps across different platforms can be transformed using code conversion tools [50], [51], [52]. Consequently, MATEMINER could be applicable across these platforms. Furthermore, third-party service providers offer mini-app development services for various super-app platforms [8], [9]. Within our test set of malicious template-based mini-apps, we identified that 47 mini-apps also appear on Alipay, 27 on Baidu, and 10 on TikTok, sharing identical names and similar appearances. This indicates that they were developed from the same templates. All these mini-apps exhibited similar malicious behaviors. Therefore, the concern of MATE risks exists in other super-app platforms. Furthermore, we evaluated MATEMINER on Alipay. Specifically, we analyzed 37,821 Alipay mini-apps and identified 601 unique templates used by 8,562 mini-apps. Among these templates, 126 contained malicious behaviors, demonstrating that MATEMINER can be applied to other super-app platforms.

Limitations and Future Work. The assessment demonstrates the effectiveness of MATEMINER in template extraction and malicious behavior detection within the app-in-app ecosystem. However, there are several limitations.

For the mini-app dataset, we cannot obtain the complete dataset of template-based mini-apps from the super-app platform. Nevertheless, our results can reveal the security risks associated with mini-app templates and provide valuable insights. For malicious behavior detection, our approach relies on static analysis to analyze downloaded mini-app packages and identify malicious behaviors. However, this method may produce false positives and false negatives. For instance, dynamically loaded content in mini-apps cannot be detected by MATEMINER, which causes several false negatives. We plan to incorporate dynamic features to enhance our tool in the future. Additionally, our study focuses on malicious behaviors in template-based mini-apps. Malicious developers may create malicious mini-apps without using templates, which is out of our focus. Moreover, our work focuses on WeChat mini-apps. Since other popular super-apps adopt similar mini-app frameworks, we plan to extend MATEMINER to these platforms in the future.

Mitigations. To mitigate the malicious behaviors associated with mini-app templates, super-app platforms must implement thorough vetting mechanisms for both the mini-app templates and third-party service providers. When a malicious mini-app is detected, if the malicious behavior is related its underlying mini-app template, super-app platforms can detect other mini-apps developed from the same template. This enables a more proactive approach to uncover hidden threats across multiple mini-apps. Additionally, super-app platforms should carefully review the actual services and registration category information of mini-apps to ensure consistency. For mini-app users, being aware of potential risks, such as data privacy concerns or the possibility of encountering inducement mini-apps, can significantly reduce the impacts of malicious behaviors. Super-app platforms can provide clear guidelines and incorporate educational materials about malicious behaviors to better help users enhance their awareness. Moreover, we believe that MATEMINER provides a valuable tool for detecting and mitigating malicious behaviors associated with mini-app templates in the app-in-app ecosystem, and it is feasible to integrate our checks into super-app's workflow.

6. Related Work

Third-party Library Detection. Third-party libraries (TPLs) are widely used in app development, and their security vulnerabilities are increasingly gaining attention due to their potential impact on the numerous applications that integrate them. Therefore, existing research primarily focuses on detecting TPLs and analyzing their security implications [42], [53], [54], [55], [56]. LibScout [53] profiles TPL features using class hierarchy, which is resilient to common code obfuscations. LibPecker [55] introduces an adaptive class similarity threshold and weighted similarity score to improve both precision and recall. LibScan [42] adopts a three-step TPL detection, which utilizes method-opcode similarity and call-chain-opcode similarity to improve the accuracy. Unlike similarity-based methods, LibD [54] leverages app code dependencies to detect and classify library candidates. In this paper, we focus on detecting the templates used in mini-apps. Unlike TPLs, which are typically independent components, mini-app templates are closed-source and often tightly integrated with the mini-app code. As a result, detection methods designed for TPLs are not applicable to our task.

Malware Detection. There have been several studies focused on malware detection [13], [14], [15], [16], [57], [58], [59], [60]. Researchers have employed both static and dynamic analysis methods to extract malware features, such as permissions, API calls, and behavior patterns, and then used machine learning techniques to enable efficient detection and classification. For instance, DroidAPIMiner [58] and DREBIN [59] utilize the frequency of API calls as features. MaMaDroid [60] models app behaviors using Markov chains, based on sequences of abstracted API calls. Additionally, as malware evolves to evade detection, many studies focus on improving the robustness of existing classifiers. APIGraph [14] enhances classifier resilience by capturing API semantics, while Chen et al. [15] tackles the issue of concept drift by adopting contrastive learning and continuously retraining Android malware classifiers.

Besides, some study found the online app generators pose serious security concerns [3], [4], [5] and many developers use app generators to create malicious apps. Chen et al. [3] finds that app generators have been abused to facilitate development of scam Android apps. Tu et al. [4] observes that ransomware generators are used to create ransomware. The templates provided by app generators follow fixed structures and place user-provided assets in predefined locations, such as 'assets/data/dcloud_control.xml' for DCloud [61]. As a result, it is easy to identify these templates and separate user-provided assets from template code based on structural patterns. However, unlike online app generators, mini-app templates are typically paid services, and the template code is tightly coupled with the user-defined code. Therefore, it is impossible to separate the template code simply by locating the configuration file. To date, there has been no systematic study to assessing malicious behaviors in template-based mini-apps and mini-app templates.

App-in-App Security. In recent years, many studies focus on the security of the app-in-app ecosystem, including the security of mini-apps [48], [62], [63] and super-apps [45], [64], [65], [66], [67]. For mini-apps, several studies have found that many mini-app developers hard-code sensitive credentials on the client side [48], [63]. Yang et al. [62] discovered that many mini-apps fail to verify the sender mini-app's identity, which can lead to data theft and shopping for free. For super-apps, Wang et al. [67] studied hidden APIs in super-apps that are undocumented and can expose sensitive resources.

Additionally, several works have explored malicious behaviors in mini-apps [17], [19], [20], [24], [68]. For example, Long et al. [19] analyzes the dark UI patterns in mini-apps and the implications of these deceptive design practices. Deng et al. [20] proposed a CNN-based image recognition to identify potential counterfeits. Yang et al. [17] conducted a systematic study on malicious mini-apps, which have been removed by super-apps, using keyword-based heuristics to detect specific types of malicious payloads. However, this method cannot be used to perform an accurate detection of malicious behaviors. In this paper, we highlight new security risks associated with template-based mini-app development and present the first systematic study of the MATE risks.

7. Conclusion

In this work, we conduct the first systematic study focused on identifying and analyzing the malicious behaviors in template-based mini-apps. We propose a novel detection tool MATEMINER and applied it to 2,282,096 real-world mini-apps. Our results demonstrate that MATEMINER can accurately extract mini-app templates and identify 15,143 unique mini-app templates, which collectively contain 268,363 mini-apps. Furthermore, MATEMINER identifies malicious behaviors in 79,758 mini-apps and 4,642 templates, revealing the widespread presence of malicious mini-app templates within the app-in-app ecosystem. This study highlights the security risks associated with template-based development and uncovers several noteworthy findings. We believe that our study can offer valuable insights for detecting and mitigating such new threats.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Natural Science Foundation of China (62172104, 62472096). Zhemin Yang and Min Yang are the corresponding authors. Min Yang is a faculty of Shanghai Institute of Intelligent Electronics & Systems, and Engineering Research Center of Cyber Security Auditing and Monitoring, Ministry of Education, China.

References

[1] statista. Monthly active users of the leading apps in China in September 2024. https://www.statista.com/statistics/1032630/china-leading-apps-by-monthly-active-users/ (visited on 06/06/2025).

[2] Wechat. WeChat Service Market. https://fuwu.weixin.qq.com/search?tab=1&sorter=7&page=1/ (visited on 06/06/2025).

[3] Z. Chen, L. Wu, Y. Hu, J. Cheng, Y. Hu, Y. Zhou, Z. Tang, Y. Chen, J. Li, and K. Ren, "Lifting the grey curtain: Analyzing the ecosystem of android scam apps," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023.

[4] C. Tu, L. Wang, Y. Xu, Y. Zhao, H. Xu, and H. Wang, "Ransomware as a service: Demystifying android ransomware generators," in *International Conference on Security and Privacy in Communication Systems (SecureComm)*, 2023.

[5] M. Oltrogge, E. Derr, C. Stransky, Y. Acar, S. Fahl, C. Rossow, G. Pellegrino, S. Bugiel, and M. Backes, "The rise of the citizen developer: Assessing the security impact of online app generators," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018.

[6] WeChat. WeChat Mini-App Permission Set Description. https://developers.weixin.qq.com/doc/oplatform/en/Third-party_Platforms/2.0/product/miniprogram_authority.html (visited on 06/06/2025).

[7] OAuth 2.0. https://oauth.net/2/ (visited on 06/06/2025).

[8] Youzan. https://www.youzan.com/ (visited on 06/06/2025).

[9] Weimob. https://weimob.com/ (visited on 06/06/2025).

[10] SXL. https://www.sxl.cn/ (visited on 06/06/2025).

[11] WeChat. WeChat Mini-App Operation Standards. https://developers.weixin.qq.com/miniprogram/product/ (visited on 06/06/2025).

[12] TikTok. Mini-App Operation Guidelines. https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/operation/miniapp-creation/basic-info/basic-info-audit-standard (visited on 06/06/2025).

[13] A. Bianchi, J. Corbetta, L. Invernizzi, Y. Fratantonio, C. Kruegel, and G. Vigna, "What the app is that? deception and countermeasures in the android user interface," in *2015 IEEE Symposium on Security and Privacy (S&P)*, 2015.

[14] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security (CCS)*, 2020.

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

[16] H. Zhou, S. Wu, C. Qian, X. Luo, H. Cai, and C. Zhang, "Beyond the surface: Uncovering the unprotected components of android against overlay attack," in *The 31st Network and Distributed System Security Symposium (NDSS)*, 2024.

[17] Y. Yang, Y. Zhang, and Z. Lin, "Understanding miniapp malware: Identification, dissection, and characterization," in *The 32nd Network and Distributed System Security Symposium (NDSS)*, 2025.

[18] WeChat. Dissecting the violative miniapp cases. https://developers.weixin.qq.com/community/business/course/00080470cb41c0c6094ab3b785b00d (visited on 06/06/2025).

[19] M. Long, Y. Xu, J. Wu, Q. Ou, and Y. Nan, "Understanding dark ui patterns in the mobile ecosystem: A case study of apps in china," in *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS)*, 2023.

[20] X. Deng, M. Zhang, X. Dong, and X. Hu, "Detect counterfeit mini-apps: A case study on wechat," in *Proceedings of the ACM Workshop on Secure and Trustworthy Superapps (SaTS)*, 2023.

[21] General Data Protection Regulation (GDPR). https://gdpr-info.eu/ (visited on 06/06/2025).

[22] California consumer privacy act (ccpa). https://oag.ca.gov/privacy/ccpa (visited on 06/06/2025).

[23] D. Bui, Y. Yao, K. G. Shin, J.-M. Choi, and J. Shin, "Consistency analysis of data-usage purposes in mobile apps," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[24] W. Li, B. Yang, H. Ye, L. Xiang, Q. Tao, X. Wang, and C. Zhou, "Minitracker: Large-scale sensitive information tracking in mini apps," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2023.

[25] C. Wang, R. Ko, Y. Zhang, Y. Yang, and Z. Lin, "Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis," in *Proceedings of the 45th International Conference on Software Engineering (ICSE)*, 2023.

[26] WeChat. Mini Program configuration. https://developers.weixin.qq.com/miniprogram/en/dev/framework/config.html (visited on 06/06/2025).

[27] Alipay. App.json configuration. https://opendocs.alipay.com/mini/framework/app-json (visited on 06/06/2025).

[28] F. Nielsen and F. Nielsen, "Hierarchical clustering," *Introduction to HPC with MPI for Data Science*, 2016.

[29] CompaniesMarketCap. Largest Chinese companies by market capitalization. https://companiesmarketcap.com/china/largest-companies-in-china-by-market-cap/ (visited on 06/06/2025).

[30] statista. The 100 largest companies in the world by market capitalization in 2023. https://www.statista.com/statistics/263264/ (visited on 06/06/2025).

[31] S. Shimmi, A. Rahman, M. Gadde, H. Okhravi, and M. Rahimi, "{VulSim}: Leveraging similarity of {Multi-Dimensional} neighbor embeddings for vulnerability detection," in *33rd USENIX Security Symposium (USENIX Security)*, 2024.

[32] F. Rahutomo, T. Kitasuka, M. Aritsugi *et al.*, "Semantic cosine similarity," in *The 7th international student conference on advanced science and technology ICAST*, 2012.

[33] Jieba Text Segmentation. https://github.com/fxsjy/jieba (visited on 06/06/2025).

[34] W. Wang, J. Huang, C. Chen, J. Gu, J. Zhang, W. Wu, P. He, and M. Lyu, "Validating multimedia content moderation software via semantic fusion," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2023.

[35] A. Arunasalam, H. Farrukh, E. Tekcan, and Z. B. Celik, "Understanding the security and privacy implications of online toxic content on refugees," in *33rd USENIX Security Symposium (USENIX Security)*, 2024.

[36] X. Li, Y. Yang, J. Deng, C. Yan, Y. Chen, X. Ji, and W. Xu, "Safegen: Mitigating sexually explicit content generation in text-to-image models," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[37] Y. Xiao, Z. Li, Y. Qin, X. Bai, J. Guan, X. Liao, and L. Xing, "Lalaine: Measuring and characterizing {Non-Compliance} of apple privacy labels," in *32nd USENIX Security Symposium (USENIX Security)*, 2023.

[38] I. Arkalakis, M. Diamantaris, S. Moustakas, S. Ioannidis, J. Polakis, and P. Ilia, "Abandon all hope ye who enter here: A dynamic, longitudinal investigation of android's data safety section," in *33rd USENIX Security Symposium (USENIX Security)*, 2024.

[39] B. Andow, S. Y. Mahmud, W. Wang, J. Whitaker, W. Enck, B. Reaves, K. Singh, and T. Xie, "PolicyLint: Investigating internal privacy policy contradictions on google play," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019.

[40] QPSoftware. Wechat mini program - all you need to know. https://qpsoftware.net/blog/wechat-mini-program-all-you-need-know (visited on 06/06/2025).

[41] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin, "A measurement study of wechat mini-apps," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, vol. 5, no. 2, 2021.

[42] Y. Wu, C. Sun, D. Zeng, G. Tan, S. Ma, and P. Wang, "LibScan: Towards more precise Third-Party library identification for android applications," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[43] JD. Taro. https://docs.taro.zone/en/docs (visited on 06/06/2025).

[44] DCloud. Uni-app. https://en.uniapp.dcloud.io/ (visited on 06/06/2025).

[45] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang, "Demystifying resource management risks in emerging mobile app-in-app ecosystems," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security (CCS)*, 2020.

[46] Mini-app was removed without specific reasons. https://developers.weixin.qq.com/community/develop/doc/00004c7586c6586ed78ff87d95bc00 (visited on 06/06/2025).

[47] N. Scaife, C. Peeters, and P. Traynor, "Fear the reaper: Characterization and fast detection of card skimmers," in *27th USENIX Security Symposium (USENIX Security)*, 2018.

[48] Y. Zhang, Y. Yang, and Z. Lin, "Don't leak your keys: Understanding, measuring, and exploiting the appsecret leaks in mini-programs." in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[49] Y. Shi, Z. Yang, K. Zhong, G. Yang, Y. Yang, X. Zhang, and M. Yang, "The skeleton keys: A large scale analysis of credential leakage in mini-apps," in *The 32nd Network and Distributed System Security Symposium (NDSS)*, 2025.

[50] Antmove. https://github.com/ant-move/Antmove (visited on 06/06/2025).

[51] Wx2my. https://www.npmjs.com/package/wx2my (visited on 06/06/2025).

[52] Baidu. Wx2. https://smartprogram.baidu.com/docs/develop/tutorial/move/ (visited on 06/06/2025).

[53] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (CCS)*, 2016.

[54] M. Li, W. Wang, P. Wang, S. Wang, D. Wu, J. Liu, R. Xue, and W. Huo, "Libd: Scalable and precise third-party library detection in android markets," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017.

[55] Y. Zhang, J. Dai, X. Zhang, S. Huang, Z. Yang, M. Yang, and H. Chen, "Detecting third-party libraries in android applications with high precision and recall," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018.

[56] X. Zhan, L. Fan, S. Chen, F. We, T. Liu, X. Luo, and Y. Liu, "Atvhunter: Reliable version detection of third-party libraries for vulnerability identification in android applications," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021.

[57] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE symposium on security and privacy (SP)*, 2012.

[58] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *Security and Privacy in Communication Networks: 9th International ICST Conference (SecureComm)*, 2013.

[59] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Network and Distributed Systems Security (NDSS) Symposium*, 2014.

[60] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "Mamadroid: Detecting android malware by building markov chains of behavioral models," in *Network and Distributed Systems Security (NDSS) Symposium*, 2017.

[61] DCloud. https://dcloud.io/ (visited on 06/06/2025).

[62] Y. Yang, Y. Zhang, and Z. Lin, "Cross miniapp request forgery: Root causes, attacks, and vulnerability detection," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[63] S. Baskaran, L. Zhao, M. Mannan, and A. Youssef, "Measuring the leakage and exploitability of authentication secrets in super-apps: The wechat case," in *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2023.

[64] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang, "Identity confusion in {WebView-based} mobile app-in-app ecosystems," in *31st USENIX Security Symposium (USENIX Security)*, 2022.

[65] C. Wang, Y. Zhang, and Z. Lin, "One size does not fit all: Uncovering and exploiting cross platform discrepant {APIs} in {WeChat}," in *32nd USENIX Security Symposium (USENIX Security)*, 2023.

[66] ——, "Rootfree attacks: Exploiting mobile platform's super apps from desktop," 2024.

[67] ——, "Uncovering and exploiting hidden apis in mobile super apps," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[68] X. Zhang, Y. Wang, X. Zhang, Z. Huang, L. Zhang, and M. Yang, "Understanding privacy over-collection in wechat sub-app ecosystem," 2023. [Online]. Available: https://arxiv.org/abs/2306.08391

[69] Wikipedia. Traditional Chinese characters. https://en.wikipedia.org/wiki/Traditional_Chinese_characters (visited on 06/06/2025).

TABLE 10: Performance of different similarity algorithms.

Tool	# TP	# FP	# FN	Precision	Recall	F1-score
MATEMINER$_{jaccard}$	280	46	56	85.89%	83.33%	84.59%
MATEMINER$_{cosine}$	221	52	115	80.95%	65.77%	72.58%
MATEMINER	312	4	24	98.73%	92.86%	95.71%

Appendix A. Similarity Algorithm Selection

To evaluate the performance of various similarity algorithms, we first select several candidate algorithms, i.e., Levenshtein Distance, Jaccard similarity, and cosine similarity. We then replace the initially chosen Levenshtein Distance algorithm with the Jaccard and cosine similarity algorithm for comparison. The results are presented in Table 10, and the dataset used is the same as that in the stepwise contribution experiment described in Section 4.2.1. MATEMINER$_{jaccard}$ represents the configuration of MATEMINER using Jaccard similarity for mini-app clustering, while MATEMINER$_{cosine}$ refers to the setup incorporating cosine similarity for mini-app clustering. The results indicate that the Levenshtein Distance algorithm outperforms both alternatives in terms of clustering effectiveness.

Appendix B. Details of Impersonation Mini-app Detection

We present the detailed algorithm for detecting impersonation mini-apps, as illustrated in Algorithm 1. Through an in-depth analysis of impersonation behaviors, they can be categorized into two types: impersonating existing mini-apps or impersonating non-existent mini-apps but use information from well-known mobile applications or companies to mislead users. Therefore, we first construct the base datasets that may be impersonated by malicious mini-apps. Specifically, we collect mini-apps developed by top companies [29], [30], as well as those from sensitive categories such as government, health, and education (B_1). For the other scenario, we extract company information from the collected mini-apps and search for mobile applications developed by these entities to expand the dataset (B_2).

Next, we traverse each mini-app m_i in our dataset to detect whether it impersonates any mini-app in the base datasets.

'Homograph' Detection. The process begins with detecting homographs in the name of m_i. To evade the detection of malicious behaviors, many mini-apps do not directly imitate the content of legitimate mini-apps, such as the mini-app name, to disguise themselves. Instead, they use words with similar shapes or spellings, such as "0" and "O", to deceive users. Therefore, we construct a dictionary that records pairs of original words and their corresponding homographs. MATEMINER replaces these words to recalculate the similarity score and choose a higher one. Additionally, some homographs are rarely used, especially Traditional Chinese characters [69]. If such words appear in the name of a mini-app, the likelihood that the mini-app is a disguised one increases significantly. In such cases, we assign a high probability of impersonation and increase the total score. We then convert the homographs into their regular form for subsequent similarity analysis.

Principal Content Analysis and Impersonation Detection. For detecting impersonation of existing mini-apps, we begin by considering each mini-app m_j in base dataset B_1. First, we collect the principal content of m_j using the $PCAanalyzer$ function (line 11) and dynamically assign importance scores to different content based on word frequency. We then calculate the weighted similarity score between m_i and m_j based on importance score. If the similarity score exceeds the threshold T_{sim_1}, we consider that m_i impersonates m_j. In the alternative scenario, for each entity e_j in base dataset B_2, the process is similar. We first collect the principal content of e_j, then calculate the weighted similarity score. If the score exceeds the threshold T_{sim_2}, we consider that m_i impersonates e_j.

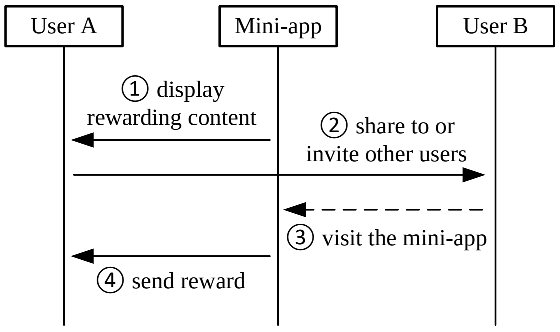


Figure 5: Process of sharing or inviting with rewards, where the dotted lines indicate actions that are not required in the sharing process.

Appendix C. Incentivized Sharing/Inviting Process

To promote mini-apps and earn more profits, many mini-apps employ incentive phrases to encourage users to share the mini-app within their social networks, particularly those aiming to attract user traffic for advertising income. This inducement primarily involves sharing and inviting behaviors. The typical processes for sharing and inviting are illustrated in Figure 5. Mini-apps display inducement phrases to users (step ①), such as "share for gifts" or "invite your friends to earn rewards". During the sharing process, when user A forwards the mini-app to user B (step ②), user A should receive the promised reward. The inviting process requires

Algorithm 1: Algorithm of Impersonation Mini-app Detection

```
Input: Mini-app dataset $D$, Base dataset $B_1$ and $B_2$,
       similarity threshold $T_{sim_1}$ and $T_{sim_2}$
Output: List of impersonation mini-apps $M$
Initialize an empty list $M$;
for each mini-app $m_i \in D$ do
    $score_{total} = 0$;
    // Check for the presence of
       homographs in the mini-app name
    foreach word $w \in m_i.name$ do
        if $w$ is Homograph then
            $score_{total} = increase(score_{total})$;
            convert $w$ to its regular form;
        end
    end
    // Type-1 Detection
    for each mini-app $m_j \in B_1$, $j \neq i$ do
        // extract principal content
        $content_{base} = PCAanalyzer(m_j)$;
        // Calculate the similarity score
           for base mini-app and mini-app
           under test
        $score_{total} = WeightedSum(content_{base}, m_i) \times 100$
        if $score_{total} >= T_{sim_1}$ then
            Mark $m_i$ as potential impersonation
              mini-apps;
            Add $m_i$ to $M$;
            break;
        end
    end
    // Type-2 Detection
    for each entity $e_j \in B_2$ do
        $content_{base} = PCAanalyzer(e_j)$;
        $score_{total} = WeightedSum(content_{base}, m_i) \times 100$;
        if $score_{total} >= T_{sim_2}$ then
            Mark $m_i$ as potential impersonation
              mini-apps;
            Add $m_i$ to $M$;
            break;
        end
    end
end
return $M$;
```

additional actions from the invited users. For example, when user A sends the mini-app to user B, user B needs to visit or register on the mini-app (step ③), after which user A earns a reward.

Appendix D. Meta-Review

D.1. Summary

This paper proposes MATEMINER, a tool for extracting mini-app templates from mini-apps and a pattern-based method to detect malicious behavior amongst both apps and templates. The work reveals widespread abuse of templates to facilitate development of malicious apps.

D.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field
- Creates a New Tool to Enable Future Science
- Provides a New Data Set For Public Use

D.3. Reasons for Acceptance

1) This paper tackles a previously understudied topic and the empirical findings uncover a concerning amount of abuse in malicious template-based development in super-app ecosystems. The actionable insights and practical implications are notable: ethical reporting of the issues to the platform owners resulted in abusive app removals and policy changes.
2) The technical approach represents a meaningful advance in this domain. The multi-layer clustering and differential extraction design is sound and rigorously evaluated. The systematic evaluation includes assessment at both control- and data-flow levels, manual labeling with expert agreement, and comparison to prior work.
3) The work provides a large, novel dataset for public use. The dataset is practically relevant, focusing on a major app platform, WeChat, and including both active and delisted apps.

D.4. Noteworthy Concerns

1) The widest concern was with generalizability to future examples: unknown templates, unseen apps, or novel malicious behaviors. The approach that MATEMINER adopts for detecting malicious behavior is highly tailored to previously seen apps, relying on static pattern matching with limited heuristics. For example, impersonation detection depends on presence of the original base apps and other categories are detected using a predefined set of keyword patterns. The work also acknowledges obfuscation as an obstacle; while only 8.6% of false positives are attributed to obfuscation, the impact on the false negative rate is unknown.
2) Similarly, the results may overfit to the WeChat app population, however the authors do include a smaller analysis on Alipay mini-apps which demonstrates that

MateMiner can be applied to other super-app platforms.

3) One reviewer noted that the detection mechanism for data theft focuses on data collection only and it cannot be determined whether the provider will share the collected data or not.