Real or Rogue? Detecting Malicious Miniapps with Deceptive Reporting Interface

Yuqing Yang*
CISPA Helmholtz Center for Information Security
Saarbrücken, Saarland, Germany
yuqing.yang@cispa.de

Zhiqiang Lin
The Ohio State University
Columbus, Ohio, United States
zlin@cse.ohio-state.edu

Abstract

Today, mobile super apps such as WeChat offer a wide array of services through integrated miniapps. While the miniapps provide self-contained services via JavaScript and Web interfaces, the existence of a centralized authority, i.e., super app platform, enables strong protection against malware. Among the many mechanisms, the built-in report interface is an essential security countermeasure, allowing users to report any suspicious miniapp that is released to the market. Alarmingly, our study reveals that there are malicious miniapps implementing deceptive reporting interfaces to impersonate the official ones. If users are guided to these fake reporting interfaces that discard or rerouting the reports, the platforms will never be alarmed about the malware existence, thus enabling the malware to circumvent post-vetting regulation. In response to this imminent threat, this paper identifies, analyzes, and constructs a dataset consisting of 3,587 malware with detailed information among 135,274 official-alike reporting interfaces among over 4 million miniapps. Our findings further reveal abundant variations of behavior, including discarding or redirecting reports, applying obfuscation to escape vetting, and batch registration to lower the risk of platform removal. We have reported these malware to parties of interest, and we will release this dataset to facilitate further detection and analysis for the web community.

CCS Concepts

• **Security and privacy → Web application security**; **Malware and its mitigation**.

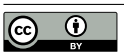
Keywords

Security; Security and Privacy; JavaScript Analysis; Program Analysis; Malware Detection

ACM Reference Format:
Yuqing Yang and Zhiqiang Lin. 2026. Real or Rogue? Detecting Malicious Miniapps with Deceptive Reporting Interface. In *Proceedings of the ACM Web Conference 2026 (WWW '26), April 13–17, 2026, Dubai, United Arab Emirates.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3774904.3792470

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.18360562.

*The work was conducted while the author was at The Ohio State University.



WWW '26, Dubai, United Arab Emirates

ACM ISBN 979-8-4007-2307-0/2026/04
https://doi.org/10.1145/3774904.3792470

1 Introduction

With the rise of mobile super apps such as WeChat, the miniapp framework has become increasingly popular, providing instant access to lightweight, yet comprehensive third-party services for hundreds of millions of users [9]. This is achieved by integrating a execution and rendering engine similar to the ones provided by web browser for extensions, but more closely dependent on the mobile super app's modules, such as payment, account information, and resource access. As such, each mobile super app becomes its own "browser" that supports "extensions" from third-parties to be downloaded to users' devices on demand.

While super app APIs enhance functionality with easy access to user and cloud resources, during the past decade, malicious developers have been integrating various evasive and camouflaging techniques against the mandatory vetting mechanisms enforced by mobile super apps. As such, it is vital to involve users to participate in the continuous monitoring of existence of malicious miniapps, which enables the platforms to keep monitoring new forms of malware, and to implement upgraded countermeasures to identify them. Thus, the built-in report mechanism is introduced. Whenever a user executes a miniapp, the mobile super app integrates a menu item redirecting to the report portal for suspicious miniapp disclosure. With a few clicks, a user can effortlessly submit concerns and type of suspicious activity the miniapp engages in, which contrasts with conventional app stores such as Google Play[11], that require users to exit the app and contact the app stores, such as by emailing concerns to the platforms.

Interestingly, while these report portal is supposed to be implemented by the platforms only, we were surprised to discover many miniapps implementing pages resembling the official reporting interfaces, built from scratch, but with almost pixel-to-pixel similarity. This certainly presents a major threat to the platform, as if users are lured to these fake reporting interfaces, the platform remains completely unaware of the existence as these reports are redirected or discarded. Although imitating app UI is a known tactic in phishing, the replication of authentic reporting interfaces from the platforms is much more devastating, as it prevents the reports submitted by the user from being received by the platforms. As the super apps lack continuous post-vetting malware overwatch mechanisms, this ensures an extended lifecycle of malware to affect more victims before it is finally delisted from the platform.

In response to this emerging threat, this paper pioneers the research monitoring the '**MI**niapp **R**eport **A**voidance **G**uise malwar**E**' (MIRAGE), a novel class of malware that ingeniously disguises itself as legitimate security interfaces to discard or redirect report of suspicious miniapp, thereby dodging detection. Unfortunately, due to the lack of proper datasets, it remains a effort-consuming and

challenging task to properly identify these malware. Therefore, in this research, we present the first research detecting such malware utilizing domain knowledge and similarity matching, and release the first fake reporting interface miniapp dataset to facilitate future research. Among over 4 millions miniapps, we identified 135,274 suspicious pages, which is further clustered based on the displayed content for de-duplication. A thorough and comprehensive verification of the dataset is performed to identify false-positive cases, resulting in a tagged dataset consisting of 3,692 miniapps, with 3,587 true malware and 105 false positives. We also tag the reasons for the false positives so as to facilitate future works to improve automatic detection. We further identify that MIRAGE extends beyond mere imitation of reporting interfaces to support malicious activities such as privacy infringement and deceptive gaming, with some miniapps even masquerading their fake interfaces as phishing sites by soliciting users' phone numbers under the guise of future contact, adopting a variety of monetization patterns that inflict financial risks and losses to the super app ecosystem.

In short, in this paper, we make the following contributions:

- **Unveiling** MIRAGE **Malware:** We present the first large-scale analysis on an evolved form of malware named MIRAGE, rooted in traditional phishing schemes, yet displaying novel behaviors by replacing official reporting interfaces with deceptive ones.
- **Comprehensive Measurement Insights:** Through our extensive measurement and analysis, we uncover the operational strategies of MIRAGE, yielding critical insights. These findings not only demystify the malware's complex behaviors but also offer actionable insights for developing effective defense mechanisms against this type of malware.
- **Dataset with comprehensive details:** We release the MIRAGE dataset[1] with significant details in how malware attempts to undermine user trust and reporting mechanisms to facilitate future research for the researchers of the security community.

2 Background

2.1 Authorities in Miniapp Paradigm

As shown in Table 1, the miniapp paradigm, featuring super apps like WeChat, involves a centralized authority that enforces stringent controls on miniapps. The table illustrates a trend towards centralization in authority regulations: web paradigms largely adopt decentralized vetting and reporting; mobile paradigms are semi-centralized, with regulation mainly enforced by app stores (including alternative ones) rather than runtime operating systems providers like Android; and super apps represent full centralization, where the super app, the very entity providing the runtime environment, actively monitors and regulates the behavior of miniapps.

Super app vs. Web. Compared with web platforms where there is no centralized authority (except the law enforcement) maintaining the security of websites and taking down malicious domains, the super apps actively vet all submitted miniapp codes along with their developers (e.g., developers have to submit identity ID or business certificate to be able to publish miniapps), to make sure that the miniapps released to the ecosystem are trusted. Also, the super apps even embed interfaces [30] for users to report any miniapps deemed malicious, whereas web platforms do not involve such centralized reporting mechanisms.

[1]The dataset will be hosted on https://minimalware.github.io/.

	Web App	Mobile App	Miniapp
Environment	Browser	Mobile Operating System	Suer App
Authority	Decentralized	Seperate App Store	Super App
Vetting	Decentralized	By Certain App Store	By Super App
Reporting	Decentralized	Write email to App Store	Via Built-in Interface

Table 1: Comparison of the authorities in different paradigms

Super app vs. Mobile. Even compared with mobile platforms, the vetting and reporting mechanisms enforced in miniapps are more powerful. First, side-loading is permitted in Android market, enabling malicious developers to release the malware directly to users, whereas it is prohibited in super apps. Second, vetting in mobile apps is performed by specific app stores (such as Samsung Galaxy Store [23] and Huawei App Gallery [13]), and may not necessarily be affiliated with the provider of Operating Systems, such as Google and Apple. Third, even though app stores allow users to report malware, users generally have to find and email the specific app stores by themselves, where malware may take longer time to be taken down.

2.2 The Miniapp Reporting Portal

While top-down vetting has been extensively discussed in various papers [19, 34, 36, 38], little is known about the bottom-up reporting. As a crucial mechanism for users to report malware to the platforms, the super apps provide miniapp reporting portals to allow users to select types of malware and submit the report conveniently via two main approaches, both independent from individual miniapps, i.e., cannot be manipulated by third-parties:

- **Through Supper App Interfaces [30]:** When users launch a miniapp, the super app will launch a separate interface to execute the miniapps with menus and widgets integrated in the container. Users can report the malware by clicking on the three-dot menu at the top-right corner of the miniapp UI. Then, the super app will launch the webpage for reporting the malware, such as https://mp.weixin.qq.com/mp/infringement for WeChat. In the interface as shown at top left of Figure 1, users may enter the reasons and submit screenshot of malicious activities to report the malicious miniapp to the platform.
- **Through In-Miniapp Component Page [32]:** In addition to allowing users to enter the report interface from the top-right button, the platform also embeds the reporting interface as miniapp components. As a convenient service for shop owners to easily create their own miniapps to sell products, WeChat provides an official miniapp called "WeChat Mini Shop" [32] to help shop owners to create miniapps automatically. Among the components integrated in the generated miniapp, there is a crucial component called `wxpay` which encapsulates payment procedure. As payment is a relative sensitive service, this official component includes malware report interfaces with the same appearance as the webpage version for users to submit report of malware or complaints about the payment process.

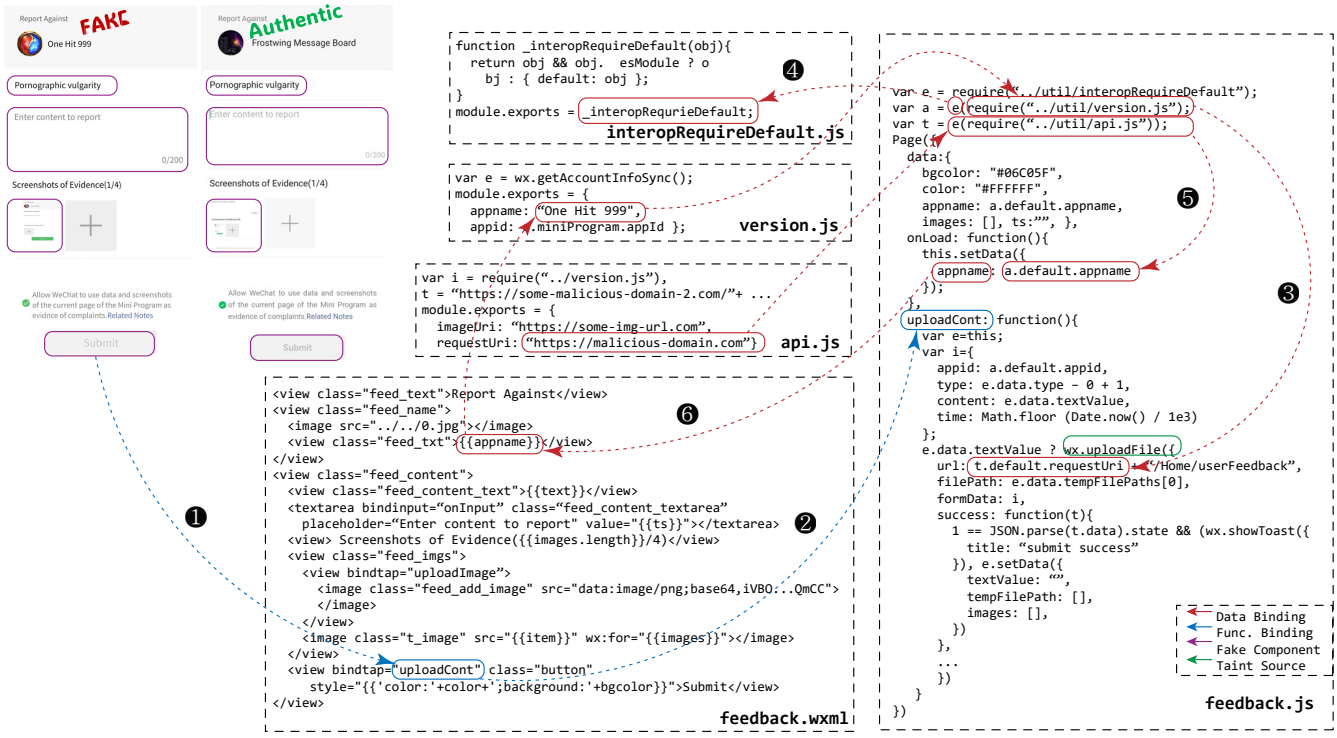


Figure 1: An example of the miniapp faking report submission interface

3 Motivating Example

To understand how a malicious miniapp implements deceptive reporting portals to mislead users, we present a real world example of an actual MIRAGE malware, as shown in Figure 1. In this example, if the user fills the form and clicks on the submit button, the miniapp will send the report to the attacker-controlled server instead of WeChat official server. To do so, the WXML page first displays an authentic-alike reporting interface, replicating the textual and visual elements in legitimate report pages. Subsequently, interactive elements like the `view` component enables users to interact with the page. Then, this page binds the submit button with `uploadCont`, which handles the user submission in `feedback.js`. Consequently, the data is sent to `requestUri`, instead of official URL. In practice, not all miniapps send the report out. For instance, the miniapp can simply ignore the submission by implementing a stub function that does nothing to effectively discard the user report. As such, the operation of the miniapp can be dissected into two distinct flows.

Page Content Flow: The miniapp utilizes various JS and WXML files, with WXML serving as a super app-specific variant of HTML. As shown in Figure 1, the miniapp features a page, `feedback.wxml`, designed to mirror the layout of the official miniapp reporting portal, as depicted in the purple boxes in the top left corner of the figure. This resemblance may lead users to erroneously conclude that the page is the authentic reporting interface for miniapps involved in inappropriate behaviors. Initially, `feedback.wxml` displays the miniapp's icon and name by referencing local icon files (`../../0.jpg`) and the miniapp name defined in `feedback.js` through `appname`. In contrast to genuine report interfaces that retrieve these details from the super app's cloud, this approach is static in that all the contents in the pages are pre-defined instead of captured during execution. The text area then presents `text`, which denotes the report reason, "Pornographic vulgarity", selected by the user from the previous step. To enhance the similarity, `feedback.wxml` also embeds a base64-encoded, pre-captured screenshot of the miniapp, diverging from official practices where a live screenshot is taken.

User Interaction Flow: Beyond replicating the appearance of legitimate interfaces, the malware also simulates interactive functionalities to falsely indicate successful report submissions. Within `feedback.wxml`, the `submit` button is linked to a `view` element with a `bindtap` event tied to `uploadCont` (❶). Upon user interaction with the `submit` button, the `uploadCont` function within `feedback.js` is triggered (❷), then the report details are assembled. These details, alongside screenshots, are dispatched via `uploadFile` to a URL specified in `t.default.requestUri` (❸). Here, `t` is essentially `api.js` (❹), although it is encapsulated by the script `interopRequireDefault.js`. By examining `api.js`, we understand that `t.default.requestUri` is ultimately set to a domain named `malicious-domain.com` in `api.js` as `requestUri`, which is under the control of the malware creators rather than the legitimate platform. This deceptive process ensures that the report never reaches the intended super app platform, thereby keeping the existence of the malware hidden and allowing its continued proliferation among users.

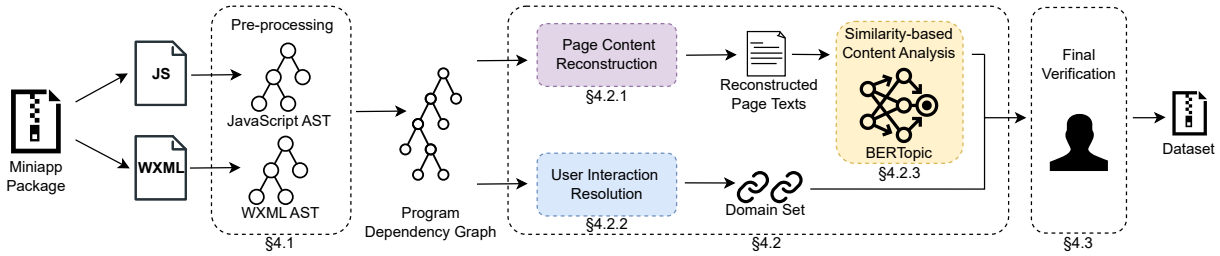


Figure 2: The MIRROR Malware Detection Workflow.

4 Identifying the MIRAGE Malware

Building on the example presented in §3, we enforce a mixture of semantic filtering, automatic identification, and manual verification to craft the MIRAGE dataset. As cross-checking the 4 million miniapps with hundreds of pages on average is, unfortunately, neither scalable nor feasible for dataset construction, we first perform pre-processing with a single string filter to reduce the less-relevant miniapps. Then, we perform automated identification with cross-module data flow reconstruction and static analysis to rebuild the semantic information displayed on the pages, based on a tool derived from DoubleX [10] and CMRFScanner [35], but largely extended to adapt to the complex, cross-page data flow of MIRAGE malware. After the reconstruction, the displayed contents are extracted, and passed to a similarity model, calculating the similarity of between the page and the authentic report interface's page content. Finally, we verify these candidate malware by generating MD5 hash of the malicious page WXML, and manually evaluate one miniapp of each group to verify the correctness and tag the malicious behavior.

4.1 Pre-processing

As the reporting interface is embedded by the super app itself, benign miniapps generally do not need to implement their standalone report interface, leaving a large portion of miniapps irrelevant of the contents such as "report miniapp" in their pages. As such, these can be filtered out to reduce the time and space required for analysis and clustering. However, we wish to leave as many suspicious miniapps as possible to maximize the inclusiveness of the dataset. By analyzing the authentic reporting interfaces, we reach to a conclusion that, users are ultimately guided to a page where screenshots and details are entered. In this page, the keyword "report" occurs multiple times (e.g., "Report Against", "Enter content to report" in Figure 1. Thus, we decide to use the keyword "report" to filter the miniapps. On top of that, AST trees are generated for the miniapp left for the automated identification.

4.2 Automated Identification

4.2.1 Page Content Reconstruction. As the authentic report interface is a built-in page whose code is not publicly available, MIRROR has to build them from scratch. To capture these contents, we first need to reconstruct the contents displayed in the pages, as we cannot assume that these data are always hard-coded as-is in the WXML pages. To do so, we perform cross-script data flow analysis to replace dynamically-bound variables in WXML with their corresponding values. We will use Figure 1 as an example to illustrate our methodology.

First, to find the values associated with variables in WXML pages, we need to capture all variable declarations and import for a complete data flow. As such, we capture 1) the declaration and assignment of variables and 2) cross-file import and export statements for each file, and connect them based on the file names. As such, for `a`, we track through `version.js` and identify the `appname`. The declaration and assignment involve not only variables in declaration and assignment expressions, but also the data declared as `data` field of the `Page` class in Javascript files. For example, `a` is resolved to a require statement (⑤), and the `appname` is assigned to `a.default.appname` in `feedback.js` of Figure 1. Then, with the resolved variable declarations, the AST tree of the WXML files is parsed to the analyzer, and variables interpolated between `{{` and `}}` are replaced by the value based on the variable name, e.g., "*One Hit 999*" for `appname`.

4.2.2 User Interaction Resolution. After the data is reconstructed, we still need to analyze how the miniapp handles user interaction when the button of submit is clicked. The key rationale is, if the miniapp still sends the report to the official domain of the super apps, the immitation of report interface is not considered malicious as it does not prevent the report from reaching the platform's authentic back-end. To do so, we implement control flow analysis finding a path between button functions and network-related APIs, and then data flow analysis to resolve the domains.

Step-I: Resolving the control flow for network request. The control flow analysis is similar to a taint analysis in that it starts from the functions bound to clickable components in WXML (source) and tries to find a path leading to invocation of network-related APIs (sink). For example, in Figure 1, when a user clicks on the `submit` button, the `ontap` function `uploadCont` (①) is invoked. Therefore, this function becomes the starting point of the control analysis, from which the analyzer then traverses the control flow (②), eventually identifying an invocation to `wx.uploadFile` (③). To this end, the analyzer discovers that the miniapp is sending something via network after the user clicks on the submit button.

Step-II: Resolving the data flow for URL domains. To resolve the target domain of network request, we track the data flow to find the value of `requestUri`. However, this variable is configured in a separate script `api.js`, in which we resolve that the `requestUri` is malicious-domain.com. However, in other cases, such data can

be configured at app-level, e.g., in the `data` field of `app.js`, or in the `ext.json`, and accessed via `getApp()` and `getExtConfig()`, separately[2]. As such, all the scripts are processed, including their `data` field and `export` section, to ensure the completeness of data flow resolving.

4.2.3 Similarity-based Content Analysis. With the texts reconstructed and domains recognized, MIRROR feeds all these strings extracted from pages of all the miniapps to calculate the text similarity. In this paper, we adopt SentenceBert model [14] to embed the texts and calculate the cosine similarity between the text of a page and official report interfaces, as the model is semantic-aware, i.e., can capture similarities between synonyms. As there has not yet been a well-marked dataset of ground truth for determining similarity between miniapp layouts, in this paper, we involve three experienced experts in miniapp malware and set the thresholds empirically. After the texts of each pages of each miniapps are embedded and similarity is calculated, we group these pages based on their similarity scores by a step of 0.1, from 0.0 to 1.0. Then, we determine the threshold by sampling multiple pages out of each group and manually selecting the threshold that best captures pages similar to official report interfaces. To do so, 3 security researchers with expertise in miniapp security evaluated 100 miniapps sampled for determining the threshold. With the threshold being set, MIRROR further examines each page whose similarity is above threshold. If a page sends requests to other domains or discarded, the miniapp containing this page is identified as malware.

4.3 Final Verification

After the MIRROR determines the candidate malware list using text construction and similarity threshold, we group all the identified pages based on the hash of the malicious WXML pages and verify whether the identified cases are indeed malicious. On top of that, we also tag the behavior of the malware after the user submits the report. For the malware discarding the report, we further record whether and how the malware deceives the users that the report has been submitted. Additionally, we observed that these malware may even collect users' personal contact information, so we also record the types of sensitive information these malware collect. As a result, we crafted a dataset consisting of 8 dimensions of detailed attributes, which will be discussed in detail in the next section.

5 Evaluation

5.1 Dataset Collection

We acquired a miniapp dataset containing over 4 million miniapps from MiniCrawler [38]. To detect the MIRAGE, we deployed the analysis on a server with 16 Intel Xeon CPU, which take close to one month to finish, and we identified 135,274 miniapps to contain official-alike interfaces in total. Among these miniapps, MIRROR discovered 3,707 pages within 3,692 miniapps that implement official-alike interfaces but discarding or rerouting the user report to non-authentic domains, whereas the rest of them commonly involve pages of `wxpay`, as the official `wxpay` component creating mini shops also integrates report pages submitting user reports to the platform official as discussed in §2.2, and thus they are benign.

[2]More details are discussed in the appendix.

During manual verification, we group the miniapps based on the hash value of malicious page codes, which results in 260 families. For each of the families, we sample one miniapp to manually verify the maliciousness. Because the maliciousness of MIRAGE depends on the implemented interface and the corresponding JavaScript file, sampling one out of each family is representative enough to identify the malicious of the entire family, and thus sufficient for validation of the dataset. As a result, we found 105 miniapps that are mistakenly identified as malware but display contents that can be sufficiently differentiated from authentic report interfaces, making the false positive rate of the automatic detection to be 2.84%. Among these false positive cases, 42 involves additional text declaring the provider of the reporting interfaces (such as report the details "to our company" for better services, instead of "to the platform"), 57 miniapps explicitly declare that the report interface is for complaining about improper user-generated contents (e.g., comments or posts submitted by users), and 6 miniapps are significantly different from authentic report interface in terms of means to interact, such as requiring users to scan a QR code of a WeChat account to contact customer support. These 105 false positive cases are also included in the dataset for future research, but in the rest of the paper, we will focus on the 3,587 miniapps that are confirmed to be malicious.

Efficiency. We build our tool on top of CMRFScanner and TaintMini, with additional cross-module data flow analysis to resolve the missing texts due to dynamic binding, as well as the web domains. On average, each miniapp takes 10.37 seconds to generate data flow graph for all files with MIRROR, with an additional marginal overhead of 0.001 second and 0.03 second to resolve the data binding and domain name respectively, making the performance overhead of MIRROR over TaintMini to 0.33% .

Similarity Comparison. In total, MIRROR generated similarity score for a total of 17,415,931 pages for the miniapps with official-alike report interfaces. Then, we sampled 5 pages with similarity of 0.0 to 1.0 with a 0.1 step, totaling 50 pages to determine the threshold. The threshold is a configurable parameter that can be set according to the need of analyst who use this system, and for this research, the similarity of displayed texts and the similarity scores are examined by three researchers and empirically set to 0.8 for capturing the pages that are the most similar to the official report interfaces.

Ablation Study. To avoid missing data flow for resolving texts in WXML and domains associated with network APIs, we proposed to perform page content reconstruction with WXML data binding analysis (§4.2.1), as well as interaction data flow resolution with cross-script data dependency analysis (§4.2.2). To quantify the contribution of both approaches, we calculate the malicious miniapps identified with or without the WXML Data Binding and/or Cross-script Data Dependency analysis techniques. As shown in Table 2, the approach without both techniques only discovered 41.14% of the malware detected by MIRROR. Also, compared with Cross-script Data Dependency analysis that boosts the discovery of MIRAGE by 0.08%, WXML Data Binding analysis contributed significantly more to the final detection of MIRAGE (57.37% gain over 0.08% gain). Both these techniques contribute positively to the discovery of malware detection.

	Malware	% Total
No WXML Data Binding / Data Dependency Analysis	1,461	40.73
With Cross-script Data Dependency Analysis only	1,463	40.78
With WXML Data Binding analysis only	3,534	98.52
MIRROR (with both techniques)	3,587	100.00

Table 2: Contribution of detecting MIRAGE malware for each proposed techniques.

Category	Report Page		Reason Page	
	Discard	Redirect	Discard	Redirect
Business	5	3	15	9
E-learning	0	16	52	50
Education	35	22	250	142
Entertainment	3	8	35	64
Finance	0	0	1	1
Food	34	5	4	10
Games	6	10	257	445
Government	2	11	16	13
Health	4	0	4	0
Job	0	2	4	5
Lifestyle	79	6	34	23
Photo	0	2	2	2
Shopping	113	15	25	27
Social	2	10	56	46
Sports	3	0	5	8
Tool	24	22	154	116
Traffic	3	1	4	2
Travelling	0	0	13	0
Uncategorized	16	15	110	1,279
Total	329	148	1,041	2,242

Table 3: Distribution of MIRAGE malware by categories. (Note that while miniapps add up to 3,707, there are 15 miniapps containing fake pages for both)

Rating	Report Page		Reason Page	
	Discard	Redirect	Discard	Redirect
4.1-5.0	4	5	35	43
3.1-4.0	4	6	41	65
2.1-3.0	0	5	15	36
1.1-2.0	0	2	1	4
0.0-1.0	0	0	0	0
Unscored	321	130	949	2,094
Total	329	148	1,041	2,242

Table 4: Distribution of miniapps that fake official report interfaces by rating

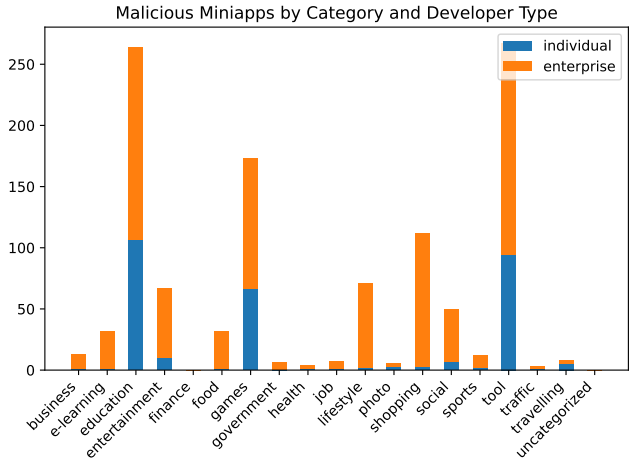


Figure 3: Malicious miniapp distribution on categories

5.2 Dataset Details

With the detection result, we discover that not all detected malware send the report to third-party domain, but instead directly discard the user report. To quantify the distribution of both types of MIRAGE, we categorize these collected malware based on the handling of user interaction, their meta information (category, rating, and developer), as well as the cross-miniapp relationships.

Categories. As shown in Table 3, we group the malware based on their pre-defined categories. Specifically, we find that there is a significant amount of Shopping, Gaming, and Tool miniapps that fake the report interfaces, and the majority of these malware discard the user input instead of sending the network request. This indicates that these malware developers actively attempt to bypass the user reporting mechanism to prevent their malware from being identified and taken down by the platform. Interestingly, the categories found to have more malware are generally more sensitive. For example, for miniapps faking Report pages, the top-3 categories are Shopping (128, 26.83%), Lifestyle (85, 17.82%), and Education (57, 11.95%), whereas for those faking Reason Page, the top-3 categories are Games (702, 21.38%), Education (392, 11.94%), and Tools (277, 8.44%). However, as gaming, lifestyle, and shopping miniapps frequently involve payment, and tool miniapps usually involve the collection of privacy data (such as phone-based user login and location info acquisition). Hence, these malware developers tend to have higher motivation to circumvent malware report mechanisms.

Ratings. Similarly, we group the malware based on the rating in Table 4. A lack of scoring for the miniapp suggests its relative lack of popularity. Surprisingly, we found that despite that more than 97% of the malware is unscored which indicate their un-popularity, there are still an amount of miniapps with higher ratings containing fake reason pages, and 98 of these miniapps are rated higher than 4.0, which indicates that even high-rating miniapps may still involve fake interfaces circumventing the reporting.

Developers. With MIRAGE identified, we further crawl the developer information to evaluate the relationships between developers and the malware they developed. As shown in Figure 3, the majority of the malware are registered as "tool", "education", and "games", in which category miniapps from individual developers are about as much as miniapps from enterprise developers. However, there are certain categories that involve significantly more enterprise developers, including shopping, social, and food. This is possibly because these categories generally require users to submit proof of licenses for providing services to sell products, food, or operate social-related services during registration, which is hard for individual developers to obtain one.

Developer	# T.	# M.	% M.	R.	% R.	Avg.	Top Cat.
Top 10 Developers by # Total Miniapp							
Tieli ***	34	1	3%	2	6%	4.8	tool
Beijing ***	33	1	3%	4	12%	4.65	shopping
Guangzhou ***	30	1	3%	9	30%	4.22	shopping
Beijing ***	28	7	25%	1	4%	3.0	tool
Beijing ***	28	1	4%	12	43%	4.3	entertainment
Fuzhou ***	27	1	4%	2	7%	3.3	lifestyle
Anhui ***	27	1	4%	0	-	-	tool
Shanxi ***	26	7	27%	16	62%	4.46	entertainment
Shanxi ***	21	5	24%	8	38%	3.48	tool
Yancheng ***	20	1	5%	16	80%	4.44	social
Top 10 Developers by # Malicious							
Beijing ***	28	7	25%	1	4%	3.0	tool
Shanxi ***	26	7	27%	16	62%	4.46	entertainment
Changsha ***	8	6	75%	6	75%	2.68	games
Shangqiu ***	16	6	38%	7	44%	3.91	entertainment
Shanghai ***	5	5	100%	0	-	-	education
Hangzhou ***	5	5	100%	2	40%	3.9	games
Shanxi ***	21	5	24%	8	38%	3.48	tool
Ganzhou ***	9	5	56%	2	22%	3.4	e-learning
Jiaxing ***	6	5	83%	0	-	-	education
Guangzhou ***	12	5	42%	5	42%	3.18	education
Top 10 Developers by % Malicious							
Hangzhou ***	5	5	100%	2	40%	3.9	games
Shanghai ***	5	5	100%	0	-	-	education
Xiamen ***	5	5	100%	0	-	-	e-learning
Xi'an ***	4	4	100%	0	-	-	games
Xi'an ***	4	4	100%	0	-	-	education
Zhengzhou ***	3	3	100%	0	-	-	education
Huiyang ***	3	3	100%	0	-	-	games
Xian ***	3	3	100%	0	-	-	tool
Shenzhen ***	3	3	100%	0	-	-	tool
Jinjiang ***	2	2	100%	1	50%	4.6	tool

Table 5: Top 10 developers, sorted by total miniapps, malicious miniapps, and portion of malicious miniapps. Names following company city redacted due to ethics consideration. T: Total miniapps submitted. M: Malicious Miniapps. R: Rated miniapps. Avg.: Average Rating. Top Cat: Top category

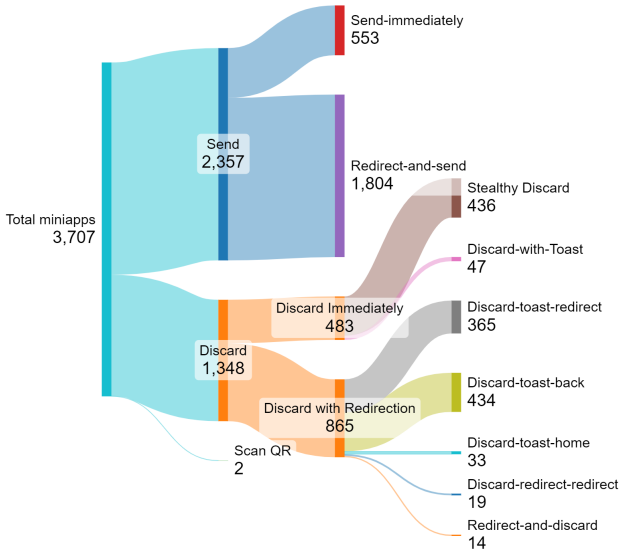


Figure 4: Sanky figures of malware behavior

Unfortunately, the platform does not display specific personal information for the individual developers. However, we are still able to characterize the enterprise developers and their association with the miniapps. As shown in Table 5, we discover that a single developer may submit a total of at most 34 miniapps, but the average rating of the submitted miniapps remain relatively high, with a small portion of malicious miniapps. On the contrary, for the enterprises involving the most malicious miniapps, the average rating of miniapps associated with the developers are significantly lower. The rating problem of the developers is even worse for the top-10 developers with maximum portion of malicious miniapps detected. All of the ten developers shown in the category involve 100% of associated miniapps being malicious miniapps, and except for two developers, none of the other developers have a single miniapp that is rated. As the platform only displays rating of a miniapp when enough user gives the rating, it indicates that either these malware are not popular, or the users using these malware are less inclined to give a rating.

Malicious behaviors. To understand what happened after users click the "submit" button of the fake report interface, we grouped and summarized the post-submit behavior in Figure 4. As illustrated in the figure, among the 3,707 malicious pages, 2,357 send the report to a third-party-maintained domain, whereas 1,348 pages simply discard the report. On top of that, there are 2 miniapps that implement a report page to allow users to scan the QR code after clicking the submit button. Among the pages that send the report, 553 pages send the report immediately at the page, whereas 1,804 pages redirect the users to an additional page and then send the report. For those discarding the report, 483 discard immediately, 436 pages show nothing to the users, and 47 pages discard the report with a message showing information such as "the report has been received" while the report is never sent. For the rest 865 pages that redirects users after discarding, 365 pages redirect the users to an additional page using `redirectTo()` or `<navigator>`, 434 pages redirect users to the previous page with `navigateBack`, 33 pages redirect users to a home page, and the rest 19 pages redirect users multiple times, first to the report details page and then redirect users to the main page. Besides, 14 pages redirect the users to a report result page showing that the report has been received (which has not). While post-submit behaviors may vary, the maliciousness of these miniapps are confirmed, because these reports are either discarded or redirected to malware developers' domains.

Obfuscation, encapsulation, and info collection. To further dissect the behavior of the malicious miniapps, we further analyzed the use of obfuscation, encapsulation of APIs redirect the report, the APIs used to redirect the report, and additional information collection if any. As shown in Table 6, there are over 30% miniapps that adopted various obfuscation and webpacking, indicating the challenge of static analysis due to lack of semantics. Additionally, 3.2% of the malware obfuscated the names of the Javascript files associated to the redirection of user report, 17 cases obfuscated the function name by replacing them with a few alphabets, and 20 miniapps obfuscated the front-end pages by using clickable images in the `<view>` instead of `<button>`, which avoid leaving the keyword

Item	All Cases	# Redirect	%Redirect	# Discard	% Discard
Info Collection					
ID	52	52	1.4%	0	0.0%
Phone	187	171	4.6%	16	0.4%
User info	120	119	3.2%	1	0.0%
WeChat	2	2	0.1%	0	0.0%
Reason of Encapsulation					
Appid	4	4	0.1%	0	0.0%
Code	12	12	0.3%	0	0.0%
Cookie	9	9	0.2%	0	0.0%
Domain	38	38	1.0%	0	0.0%
Login token	2	2	0.1%	0	0.0%
App secret	3	3	0.1%	0	0.0%
User Session	20	19	0.5%	1	0.0%
Signature	310	310	8.4%	0	0.0%
Auth token	45	45	1.2%	0	0.0%
Unknown	31	31	0.8%	0	0.0%
User info	36	36	1.0%	0	0.0%
No encap.	1,711	1,708	46.3%	3	0.1%
Obfuscation					
File name	118	118	3.2%	0	0.0%
Web-packed	1,205	1,204	32.6%	1	0.0%
Function name	17	17	0.5%	0	0.0%
Image as button	20	7	0.2%	13	0.4%
Framework API					
wx.cloud.callFunction	1	1	0.0%	0	0.0%
wx.cloud.database	1	1	0.0%	0	0.0%
wx.cloud.uploadFile	1	1	0.0%	0	0.0%
wx.fetch	1	1	0.0%	0	0.0%
wx.request	554	551	14.9%	3	0.1%
wx.scanCode	0	0	0.0%	0	0.0%
wx.uploadFile	1,153	1,153	31.2%	0	0.0%

Table 6: Detailed behaviors of the MIRAGE **Malware. Please note that a miniapp may have multiple sets of authentic-alike reporting interfaces, so there are cases containing both redirect and discard pages.**

"submit" in the button area and potentially attempting to evade the vetting regulation, but are still identified via the semantic-aware similarity analysis with reconstructed texts.

We also observe miniapps encapsulating the APIs to launch network request, mostly for user authentication. For example, besides the 31 miniapps whose motivation of encapsulation is unknown due to heavy obfuscation, 310 miniapps encapsulate the network request to attach signatures, 38 of them declare tens of various domains to be used across the miniapp (i.e., the URL for user report, user login, etc), 45 attach tokens, 9 attach cookies. In addition, these miniapps may even transmit sensitive information collected from the users, including user information and secret keys. On top of that, 4 miniapps send the AppID to the back-end, which indicates the existence of third-party platforms handling multiple miniapps' back-end functionalities, presumably the template providers.

Besides the miniapps encapsulating network request APIs, there are still 1,711 miniapps that use the network-related APIs directly without encapsulation. Despite the majority of the miniapps using standard `wx.request()` and `wx.uploadFile()` to send the reports, there are 3 cases where the miniapp invokes a so-called cloud function, which is an unique feature provided by super apps. To support developers without a back-end of their own, the super apps generally provide cloud databases for these developers to access, as well as allowing them to configure light-weight APIs called cloud APIs to interact with these back-ends. For instance, `uploadFile()` uploads files to the cloud database, `database()` allows developers to write and read data from the cloud database, and `callFunction()` allows developers to invoke functions implemented by themselves.

On top of interacting with users on reporting the miniapps, these malware may even collect sensitive information from users. For example, there are 187 malware collecting users' phones so that "we can contact you later on the issue", 120 of them collecting user information, 2 of them collecting users' WeChat account ID "so the customer services can contact you". Interestingly, there are 17 cases where miniapps collect sensitive information from the users but never sends these information, potentially trying to be sincere and authentic enough to gain trust from users.

6 Related Work

Miniapp Security. Recent works on the security of the super apps include framework-side security on super app functionalities and mechanisms [19, 24, 27, 28, 34], and miniapp-side security on vulnerabilities [35, 39, 41], malware detection [36, 37], as well as measurement studies on miniapp privacy policies [18, 29, 40]. Moreover, Want et al. proposed a static analysis tool TaintMini to capture dynamic data transmitted between pages and miniapps [26]. Compared with these existing works that mainly focuses on JS-side vulnerabilities, our paper reveals a new type of malware that closely involves both the WXML and the JS side, with a special focus on resolving the cross-file data flow dependency for dissecting the malicious behavior of the novel malware.

Phishing and Malware Detection. Detection of web-based and extension malware have been studied across the past years [1, 5, 8, 33]. For malicious webpage detection, there have been works for large scale detection and defense based on static analysis, similarity matching, and behavioral analysis [6, 7, 17, 21]. Meanwhile, as browsers begin to allow integration of web extensions, recent works have proposed various approaches to detect these extensions by analyzing the codes and required permissions of extensions [2, 12, 15, 16, 20, 22].

7 Conclusion

We have shown a novel type of malware that circumvents the malware reporting mechanisms by implementing interfaces imitating the official report portal. To detect such malware, we develop MIRROR to automatically identify these malicious miniapps with semantic-aware text embedding and cross-module data analysis. In total, we have identified 3,587 malware among 135,274 miniapps containing official-alike report interfaces, circumventing the reporting mechanism by discarding or redirecting user reports to attacker-owned domains. These malware involves various malicious activities, but remain undetected by the platforms.

Acknowledgment

We thank the anonymous reviewers for their insightful feedback during the review process of this paper. This research was supported in part by NSF award 2330264. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of NSF.

References

[1] Shubham Agarwal. 2022. Helping or Hindering? How Browser Extensions Undermine Security. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 23–37. doi:10.1145/3548606.3560685

[2] Shubham Agarwal, Aurore Fass, and Ben Stock. 2024. Peeking through the window: Fingerprinting Browser Extensions through Page-Visible Execution Traces and Interactions. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security* (Salt Lake City, UT, USA) *(CCS '24)*. Association for Computing Machinery, New York, NY, USA, 2117–2131. doi:10.1145/3658644.3670339

[3] Alipay Documentation Center. 2025. Operation Regulation of Alipay Miniapp. https://opendocs.alipay.com/b/03al2i.

[4] Baidu Smart Miniprogram. 2025. Detailed Operation Regulation. https://smartprogram.baidu.com/docs/operations/specification/.

[5] Duc Bui, Brian Tang, and Kang G. Shin. 2023. Detection of Inconsistencies in Privacy Practices of Browser Extensions. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2780–2798. doi:10.1109/SP46215.2023.10179338

[6] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. 2011. Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages. In *Proceedings of the 20th International Conference on World Wide Web* (Hyderabad, India) *(WWW '11)*. Association for Computing Machinery, New York, NY, USA, 197–206. doi:10.1145/1963405.1963436

[7] Jian Chang, Krishna K. Venkatasubramanian, Andrew G. West, and Insup Lee. 2013. Analyzing and Defending against Web-Based Malware. *ACM Comput. Surv.* 45, 4, Article 49 (aug 2013), 35 pages. doi:10.1145/2501654.2501663

[8] Rachna Dhamija, J. D. Tygar, and Marti Hearst. 2006. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada) *(CHI '06)*. Association for Computing Machinery, New York, NY, USA, 581–590. doi:10.1145/1124772.1124861

[9] Digital Creative Asia. 2025. WeChat Mini Programs - Everything You Need To Know To Succeed. https://digitalcreative.cn/blog/wechat-mini-programs-simple-guide.

[10] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. 2021. DoubleX: Statically Detecting Vulnerable Data Flows in Browser Extensions at Scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 1789–1804. doi:10.1145/3460120.3484745

[11] Google Play. 2025. Android Apps on Google Play. https://play.google.com/store/.

[12] Sheryl Hsu, Manda Tran, and Aurore Fass. 2024. What is in the Chrome Web Store?. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security* (Singapore, Singapore) *(ASIA CCS '24)*. Association for Computing Machinery, New York, NY, USA, 785–798. doi:10.1145/3634737.3637636

[13] HUAWEI Developers. 2024. App Gallery Review Guidelines. https://developer.huawei.com/consumer/en/doc/distribution/app/50104.

[14] Hugging Face. 2025. uer/sbert-base-chinese-nli. https://huggingface.co/uer/sbert-base-chinese-nli.

[15] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. 2015. Trends and lessons from three years fighting malicious extensions. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Washington, D.C.) *(SEC'15)*. USENIX Association, USA, 579–593.

[16] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. 2014. Hulk: eliciting malicious behavior in browser extensions. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 641–654.

[17] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. 2013. Revolver: An automated approach to the detection of evasive web-based malware. In *22nd USENIX Security Symposium (USENIX Security 13)*. 637–652.

[18] Shuai Li, Zhemin Yang, Yunteng Yang, Dingyi Liu, and Min Yang. 2024. Identifying Cross-User Privacy Leakage in Mobile Mini-Apps at a Large Scale. *Trans. Info. For. Sec.* 19 (Jan. 2024), 3135–3147. doi:10.1109/TIFS.2024.3356197

[19] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying Resource Management Risks in Emerging Mobile App-in-App Ecosystems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 569–585. doi:10.1145/3372297.3417255

[20] Marvin Moog, Markus Demmel, Michael Backes, and Aurore Fass. 2021. Statically Detecting JavaScript Obfuscation and Minification Techniques in the Wild. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 569–580. doi:10.1109/DSN48987.2021.00065

[21] Rui Ning, Cong Wang, ChunSheng Xin, Jiang Li, Liuwan Zhu, and Hongyi Wu. 2019. CapJack: Capture In-Browser Crypto-jacking by Deep Capsule Network through Behavioral Analysis. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. 1873–1881. doi:10.1109/INFOCOM.2019.8737381

[22] Nikolaos Pantelaios, Nick Nikiforakis, and Alexandros Kapravelos. 2020. You've Changed: Detecting Malicious Browser Extensions through their Update Deltas. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 477–491. doi:10.1145/3372297.3423343

[23] Samsung Developers. 2025. App Distribution Guide. https://developer.samsung.com/galaxy-store/distribution-guide.html.

[24] Yizhe Shi, Zhemin Yang, Kangwei Zhong, Guangliang Yang, Yifan Yang, Xiaohan Zhang, and Min Yang. 2025. The Skeleton Keys: A Large Scale Analysis of Credential Leakage in Mini-apps. In *32nd Network and Distributed Systems Security Symposium (NDSS)*.

[25] Tiktok Open Platform. 2025. Miniapp Operation Regulation. https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/operation/management/specification/standard.

[26] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Taintmini: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 932–944. doi:10.1109/ICSE48619.2023.00086

[27] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. Uncovering and Exploiting Hidden APIs in Mobile Super Apps. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 2471–2485. doi:10.1145/3576915.3616676

[28] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2024. Root Free Attacks: Exploiting Mobile Platform's Super Apps From Desktop. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security* (Singapore, Singapore) *(ASIA CCS '24)*. Association for Computing Machinery, New York, NY, USA, 830–842.

[29] Yin Wang, Ming Fan, Junfeng Liu, Junjie Tao, Wuxia Jin, Haijun Wang, Qi Xiong, and Ting Liu. 2024. Do as You Say: Consistency Detection of Data Practice in Program Code and Privacy Policy in Mini-App. 24 pages. doi:10.1109/TSE.2024.3479288

[30] WeChat Open Community. 2025. Official guide of obfuscating official report interfaces. https://developers.weixin.qq.com/community/develop/doc/0004c8bfe845009c466f050ea51009

[31] WeChat Open Community. 2025. Service Categories Opened for Miniapps. https://developers.weixin.qq.com/minigame/product/material/.

[32] WeChat Open Community. 2025. What is minishop? Who can apply for minishop? https://developers.weixin.qq.com/community/develop/doc/0008a6a52f8d182a91aacb32156c09

[33] Qinge Xie, Manoj Vignesh Kasi Murali, Paul Pearce, and Frank Li. 2024. Arcanum: detecting and evaluating the privacy risks of browser extensions on web pages and web content. In *Proceedings of the 33rd USENIX Conference on Security Symposium* (Philadelphia, PA, USA) *(SEC '24)*. USENIX Association, USA, Article 258, 18 pages.

[34] Yuqing Yang, Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. SoK: Decoding the Super App Enigma: The Security Mechanisms, Threats, and Trade-offs in OS-alike Apps. (2023). arXiv:2306.07495 [cs.CR] https://arxiv.org/abs/2306.07495

[35] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (Los Angeles, CA, USA) *(CCS '22)*. Association for Computing Machinery, New York, NY, USA, 3079–3092. doi:10.1145/3548606.3560597

[36] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2025. Understanding the Miniapp Malware: Identification, Dissection, and Characterization. In *32nd Network and Distributed Systems Security Symposium (NDSS)*.

[37] Yang, Yuqing and Zhiqiang Lin. 2025. Stealthy Trackers: Uncovering Permissionless Fingerprinting in WeChat Miniapps. In *Proceedings of the 2025 Workshop on Security and Privacy of AI-Empowered Mobile Super Apps (SaTS '25), October 13–17, 2025, Taipei, Taiwan*. doi:10.1145/3733824.3764871

[38] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. 2021. A Measurement Study of Wechat Mini-Apps. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 2, Article 14, 25 pages. doi:10.1145/3460081

[39] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. 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* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 2411–2425. doi:10.1145/3576915.3616591

[40] Zhibo Zhang, Lei Zhang, Guangliang Yang, Yanjun Chen, Jiahao Xu, and Min Yang. 2024. The Dark Forest: Understanding Security Risks of Cross-Party Delegated Resources in Mobile App-in-App Ecosystems. *IEEE Transactions on Information Forensics and Security* 19 (2024), 5434–5448. doi:10.1109/TIFS.2024.3390553

[41] Zhibo Zhang, Zhangyue Zhang, Keke Lian, Guangliang Yang, Lei Zhang, Yuan Zhang, and Min Yang. 2023. TrustedDomain Compromise Attack in App-in-app Ecosystems. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps* (Copenhagen, Denmark) *(SaTS '23)*. Association for Computing Machinery, New York, NY, USA, 51–57. doi:10.1145/3605762.3624430

Appendix

This appendix presents additional information that could be of interest to the readers to this paper. Such information includes 1) a detailed behavioral analysis on the identified malware to understand the motivation behind the bogus reporting interface and the techniques that accompany such malicious behavior, 2) a discussion on the generality of MIRROR malware on other super app platforms other than WeChat, 3) a comparative discussion between phishing malware and MIRROR, 4) additional information regarding the unique miniapp-specific data flow that the proposed detection technique handles, as well as 5) a discussion on the limitation of our research and mitigation mechanisms to shed light on future research.

A Behavioral Analysis and Observations

Upon identifying the malicious miniapps that implement reporting interface to evade regulation from the platforms, we want to understand why these miniapps of implement such interfaces. As a special type of evasive technique deployed at post-vetting phase, these malware may as well perform unwanted or even malicious activities necessitating the implementation of fraud reporting interface to confuse users. To do so, we performed manual analysis after clustering these malware based on the signature of the fake reporting pages. As a result, we identified various privacy-sensitive behaviors and additional regulation evasion techniques that are deployed in combination with the fake report interface technique. On top of that, by examining the code and UI of the miniapps, we further uncover multiple monetization schemes that may enable developers of these miniapps to benefit financially from the users.

A.1 Privacy Collection

We first performed analysis directly on the displayable contents on the fake report interfaces. Despite the majority of the identified interfaces are completely duplicating the official reporting interfaces, we were surprised to identify several clusters of bogus interfaces that place additional input box asking (luring) users to provide privacy-sensitive information. For example, a case miniapp requires phone numbers from users so that "the platform will contact them shortly regarding the report", as shown in Figure 5, where the submitted phone number `inputPhone` is eventually sent to `hostUrl` in `config.js`. We even found cases that collect credit card number along with their real names and phone numbers, as well as home address and social account ID. This is particularly suspicious when the WeChat already provides secure and convenient APIs to obtain user ID and their phone number protected by permission mechanisms.

A.2 Regulation Evasion

On top of the bogus interface, we are particularly interested on whether the miniapps implement additional mechanisms to circumvent vetting or confuse users. Consequently, we identified 3 interesting family of evasive techniques that are used together with the bogus reporting interfaces.

Categorical camouflaging. As shown in Figure 3, a majority of the identified malware is tool miniapps, which is a category that requires less verification documentations when registering, as listed

```
<form id="form">
  <view class="report_title"> Select reason for reporting
  </view>
  ...
  <view class="weui-cells  title">
    Please enter your phone number so we can contact you later
  </view>
  <view class="report_cell weui-flex">
    <view class="weui-cell  hd">
      <label class="weui-label">Phone number</label>
    </view>
    <view class="weui-cell  bd weui-flex  item">
      <input bindinput="phoneInput" maxlength="11" type="number">
      </input>
    </view>
  </view>
  <view class="button-area">
    <button bindtap="submitReport">Submit</button>
  </view>
</form>                                              report.wxml
```

```
var t = require("../../config"), c = this;          Data Binding
Page({                                              Func. Binding
  phoneInput: function(e) {
    c.data.inputPhone = e.detail.value;
  },
  submitReport: function(i) {
    e.request({
      url: t.service.hostUrl,                       https://dt.bleege.com
      data: {                                       (config.js)
        ver: t.service.version,
        ...
        phone: c.data.inputPhone,
      })
  },
})                                                  report.js
```

Figure 5: Case of privacy acquisition

in the official documentation for miniapp category registration [31]. Compared with other categories that require enterprise certificates, the tool category requires no such qualification document. However, the actual services may be camouflaged under the seemingly innocent category, incurring implicit financial risks and fraud risks. The identified malware claims that the users may play to earn money, but the difficulty to earn the money is significantly high. In the Q&A game where users answer questions to earn money, the users only have 10 seconds to answer the questions, and will only get the reward after 15 consecutive correct answers. Ironically, while the user may be able to redeem products after they finally made 15 consecutive correct answers, the button is not associated with processing logic, and no functionalities related to payment is identified in the case miniapp. Even if the users would like to report the miniapp, they may be redirected to the fake reporting interface that discard the report, merely showing that the report "is acknowledged".

Shell page. Reports of "spamming games" has been constantly received among the super app platforms, which commonly provide malicious activities such as fraud in-game purchasing. For instance, the game shown in Figure 6 implements a camouflage page showing benign pictures to bypass vetting, and after that switch to the malicious game service page. To further evade the vetting, the games are not implemented as miniapp code, but simply wrapped in a `<web-view>` pointing the URL to the malicious developer's website.

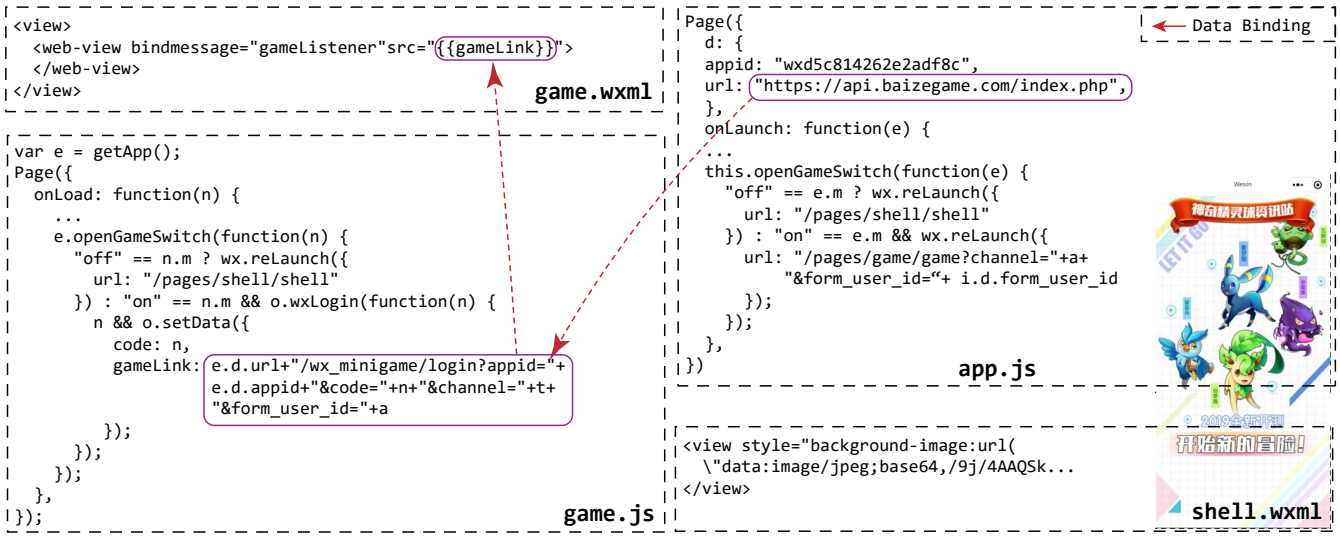


Figure 6: A case for fraud games circumventing report and vetting mechanisms

Request obfuscation. Obfuscation is enforced by many malicious cases to hide the back-end domain and encapsulate the request body construction. And even the name of the script can be obfuscated. For example, a script called `54C6FB07E19E9BCF32A09300776ECC11.js`, automatically constructs the domain back-end URL to handle communication with back-ends. Instead of directly declaring the functions, the script exports the functions as an object property, and this is imported by `app.js` as `util`. Then, to send network requests, it first invokes `var a=getApp()` to fetch the global data object declared in `app.js`, and then call `a.util.request()` directly. In the call to `a.util.request`, the miniapp does not need to specify the URL as the URL is declared in the encoded script. As a result, the data flow of the static analysis breaks, and the attacker successfully hides the communicated domain from automatic detector.

A.3 Monetization Schemes

Further, we proceed to understand the identified miniapps as a whole to evaluate the motivation of malicious developers to implement such malware. Our investigation uncovered that developers may be as well motivated to utilize such malware as a channel to monetize from the platform or external parties by engaging users to use their miniapps. Such a motivation leads to various monetization strategies, which can be broadly categorized into three types: play to earn, pay to win, and reciprocal miniapps.

Play to Earn. A notable tactic to monetize unqualified services of malware is the "play to earn" scheme. For instance, the Q&A game case promises monetary rewards and redemption of products. Another case entices users with gift lotteries, but only draws the winner after a certain amount of participants enter. To lure more users so as to boost revenue through advertisements, alternatives such as sharing the miniapp in group chats to gain direct rewards are deployed, consisting half of the 20 miniapps analyzed.

Pay to Win. Additionally, games may leverage the traditional "pay to win" scheme. For example, a game induces users to pay for enhancements and gadgets to defeat bosses. These miniapps may escalate game difficulty, compelling users to make additional purchases for additional chances to win, especially when intertwined with "play to earn" elements.

Reciprocal Miniapps. In addition to "play to earn" schemes, we identified miniapps that, while not directly extracting money from users, monetize through embedded advertisements, thus functioning as reciprocal miniapps. For example, a miniapp case provides little functionality other than a page with banner ads redirecting users to other miniapps upon clicking. This allows malware developers to profit from advertisement revenue, even if the miniapp itself offers minimal functionality, marking a new form of spamming within the ecosystem that is discouraged by the platform.

B Generality of MIRROR

While this paper mainly focuses on WeChat platform due to availability of miniapp information and dataset, the detection approaches and proposed malware is applicable to other super app platforms. First, the detection technique is applicable across platforms as major super app platforms such as Baidu and Alipay use similar architecture of execution environments, and miniapp scripts are written in JavaScript. Second, through implementing miniapps in these platforms, we discover that the APIs bear similarity with minimal naming differences, which indicates that MIRROR only needs to make marginal changes to adapt to different file postfix and API prefixes. Third, report interfaces are commonly integrated by super app platforms, and the behavior of imitating official report interface, i.e., "confounding with official activities", is forbidden by all of the mainstream platforms we evaluated [3, 4, 25]. Thus, such behavior is commonly identified as malicious across super app platforms other than WeChat.

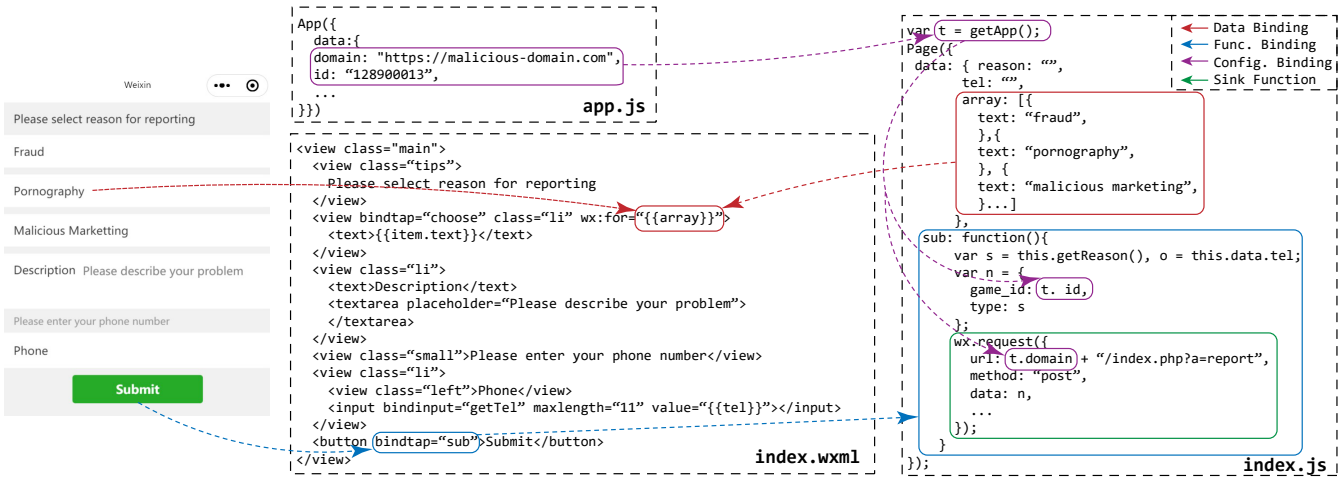


Figure 7: Example of miniapps using dynamic binding to display interfaces of "select reason" page

C Comparison with Phishing Malware

While both types of malware involve masquerading authentic interfaces, MIRAGE still exhibits novelty in the paradigm with the existence of a powerful authority (super app). First, the motivation of the attacker in phishing is to obtain information submitted by a user, whereas motivation of MIRAGE is to discard or redirect the user's information. Second, the web domains of "authentic" apps in phishing can be various (e.g., Facebook or Google), whereas for MIRAGE, only the domain of the super app is authentic. Third, MIRAGE is essentially a regulation evasion technique, whereas traditional paradigms generally do not have a unified authority that vets all apps.

D Miniapp-specific Data Flow

In subsection 4.2, we discussed three types of data flow: the data flow between WXML and JavaScript script to resolve interpolated dynamic binding variable, the data accessed via `data` field of the script, and the data accessed via app-level API such as `getApp()`. In the domain of miniapp, the latter two data flow is unique, and are accessed by platform-specific APIs, which needs to be resolved accordingly.

For the variables fetched via `data` field of the script, these variables are accessed by invoking `getData()`, and their scope is script-specific, i.e., can only be accessed within the script. For the variables fetched via `getApp()` and `getExtConfig()`, the data is global, i.e., can be accessed by any script in the miniapp. The `getApp()` accesses the variables declared in `app.js`, which is the main script of a miniapp, and the `getExtConfig()` returns a JSONObject containing all items in `ext.json`. As such, during the initial construction of AST for all scripts, we record all the data defined in each script, as well as the JSON files. Thus, if such API is invoked, we look up the specific value of these variables. To make the workflow easier to understand, we use the code in Figure 7 as an example. The domain of `wx.request` is set to `t.domain`, where `t` is fetched via `getApp()`, which returns the `data` object declared in `app.js`. Therefore, we lookup the `data` object in `app.js`, and finally the domain is resolved to `malicious-domain.com`.

E Limitation and Mitigation

Limitation. This paper prioritizes crafting a workable dataset over comprehensiveness of the detection, and thus compromises are made when designing the detection framework. First, semantic based filtering is performed because the static analysis incur deep traversal of AST across multiple scripts, which is time-consuming. Second, the similarity threshold is applied based on sampling. To mitigate this issue, we verify the results to prevent false positives. On top of that, in this paper, we only focus on fake report interfaces in miniapp pages and analyzed the JS and WXML code in miniapp packages. However, the malware may circumvent this detection by implementing their fake report interface on third-party domain and use `<web-view>` in WXML instead of implementing them in miniapp source code. However, detecting such case is a unknown challenge, as code hosted on third-party domain is not available to non-developers. Also, our extensive sampling did not find such case, and thus this case is out-of-scope.

Mitigation. Given the dynamic and evasive nature of malware, it has been challenging to identify and defend against such malware. In the case of MIRROR, attackers could further circumvent detection by utilizing dynamic capabilities of JavaScript, such as by loading the malicious contents completely at the cloud end, which further reduces the visibility of the malicious behavior. However, such behavior may be mitigated by performing side-channel observation on the behavior of users and runtime monitoring of page rendering. For instance, if a user is lured to the bogus reporting interfaces, the user may enter the information without doubt or hesitation, but the rendering monitor may still detect that the UI is similar to the official interface. Meanwhile, as reporting miniapps is generally a less popular feature to use, when a user is redirected to a less frequently-used page and stay for enough time entering text contents, the super app may be able to identify such anomaly and take actions to notify the users.