Understanding Miniapp Malware: Identification, Dissection, and Characterization

Yuqing Yang
The Ohio State University
yang.5656@osu.edu

Yue Zhang
Drexel University
yz899@drexel.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

***Abstract*—Super apps, serving as centralized platforms that manage user information and integrate third-party miniapps, have revolutionized mobile computing but also introduced significant security risks from malicious miniapps. Despite the mandatory miniapp vetting enforced to the built-in miniapp store, the threat of evolving miniapp malware persists, engaging in a continual cat-and-mouse game with platform security measures. However, compared with traditional paradigms such as mobile and web computing, there has been a lack of miniapp malware dataset available for the community to explore, hindering the generation of crucial insights and the development of robust detection techniques. In response to this, this paper addresses the scarcely explored territory of malicious miniapp analysis, dedicating over three year to identifying, dissecting, and examining the risks posed by these miniapps, resulting in the first miniapp malware dataset now available to aid future studies to enhance the security of super app ecosystems.**

To build the dataset, our primary focus has been on the WeChat platform, the largest super app, hosting millions of miniapps and serving a billion users. Over an extensive period, we collected over 4.5 million miniapps, identifying a subset $(19,905)$ as malicious through a rigorous cross-check process: 1) applying static signatures derived from real-world cases, and 2) confirming that the miniapps were delisted and removed from the market by the platform. With these identified samples, we proceed to characterize them, focusing on their lifecycle including propagation, activation, as well as payload execution. Additionally, we analyzed the collected malware samples using real-world cases to demonstrate their practical security impact. Our findings reveal that these malware frequently target user privacy, leverage social network sharing capabilities to disseminate unauthorized services, and manipulate the advertisement-based revenue model to illicitly generate profits. These actions result in significant privacy and financial harm to both users and the platform.

I. Introduction

In recent years, miniapps have emerged as a transformative computing model, gaining widespread adoption across leading mobile super apps, well beyond the traditional IT vendors. This novel framework permits third-party developers to integrate their services into a host mobile app, utilizing its data and functionalities. Embedded within super app platforms, miniapps have become fundamental to their ecosystems, enriching super apps with a diverse array of services for a vast user base. For instance, WeChat, a prominent super app platform providing services to over a billion users [1], has experienced rapid growth of its miniapp market. Reaching more than four million miniapps, the miniapps hosted by WeChat's miniapp store now surpasses the number of mobile apps available on Google Play, which currently stands at about three million [6].

To enhance user stickiness and increase the dominance of super apps, these platforms have introduced three key features for miniapp integration. (1) They provide miniapps with access to user data such as account details, phone numbers, real-time location, and home addresses. These accesses facilitate a diverse range of services, from e-commerce to transportation. (2) Miniapps can generate revenue through pay-per-click advertisements within the super app's ecosystem, providing a financial incentive for developers. (3) The ability to share miniapps within the super app's social networks, such as group chats, promotes their distribution and attracts new users. Thus, leveraging vast user data, ad-based monetization models, and social-network-based distribution channels, miniapps have emerged as a distinct and crucial component of the digital ecosystem, standing alongside traditional mobile and desktop environments.

Unfortunately, with the versatile user data delegation and miniapp distribution, the miniapp ecosystem have also been targeted by malicious parties to inflict losses to the ecosystem, including the super app platform and the end users. Despite that robust app vetting and strict control over miniapp distribution have been enforced by super apps, the hunt for malware remains a constant cat-and-mouse game, as malicious miniapps have been seeking approaches to evade the vetting and infiltrate the ecosystems. For instance, Lu et al. [48] have demonstrated the capability of malicious miniapps to bypass vetting processes and illicitly harvest private data. This ongoing struggle has also led to the development of various evasive and obfuscation techniques. What makes the situation even worse is that, to this date, there is an absence of an available malware dataset for research, leaving crucial insights for developing effective defense strategies still undiscovered.

To address this gap, this paper sets forth to collect the first miniapp malware dataset. Our preliminary investigation

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230082
www.ndss-symposium.org

reveals that despite being versatile and constantly evolving, miniapp malware inevitably leave traces in the ecosystem. Since miniapp malware must be released to the app store to interact with victim users, they often conceal their malicious content (payload) to bypass the mandatory vetting, only executing their harmful payloads after the approval. Further, passing vetting does not guarantee the malware's survival, as end users can still report suspicious miniapps to the platform, leading to their removal from the ecosystem.

Based on these observations, we conducted the first collection and analysis of miniapp malware over a three-year period, targeting samples that meet two key criteria. (1) The malware must contain the ability to evade vetting processes. To identify such evasive signatures, we analyzed the dataset using the APIs from prohibited evasion-capable libraries and examples as reported by WECHAT [9], covering both code and content vetting evasion techniques. (2) The malware must have been removed from the ecosystem, indicating that the malicious activities were acknowledged by the platform. To collect such information, we iterated through WECHAT miniapp store twice from March 2020 to December 2022. This effort revealed that 360,467 miniapps had been removed by the time when finishing our second collection. By applying these malware signatures to the miniapps removed from the platform, we ultimately compiled a dataset of $19,905$ miniapps, forming our final malware dataset.

To demonstrate the real-world impact and gain a deeper understanding of the lifecycle and motivations behind these malware, we conducted detailed case studies on the collected samples. In particular, we systematically investigated the lifecycle of miniapp malware, including infection channels, activation methods, and the types of malicious payloads involved. Additionally, we classified the malware into 6 categories with detailed case studies to showcase their security impacts. These findings reveal a broad spectrum of security threats, ranging from privacy violations to financial fraud, posed by these malicious entities.

Contributions. In short, this paper makes the following contributions:

- **Publicized miniapp malware dataset (§III).** We have identified, organized, and are now releasing the first comprehensive dataset of miniapp malware[1]. This collection, sourced from the official WECHAT miniapp market, comprises miniapps that demonstrated evasive behavior and had been removed from the store, indicating their malicious intent and violation of super app regulations.
- **Taxonomy of malicious miniapp payloads (§IV).** We further dissected the collected evasive miniapp malware, focusing on aspects such as vetting evasion, infection mechanisms, activation triggers, and the nature of malicious payloads. Using static control and data flow techniques, we examined function calls, layout components, and content displayed by the miniapps. The malicious payloads identified through this process were then organized into 6 distinct categories, in accordance with the official guidelines governing miniapp operations [36].
- **Characterization of miniapp malware (§V).** Building on our taxonomy, we characterized the malware families by examining how they combine malicious payloads and evasive techniques to harm the ecosystem. This characterization focuses on three primary categories: privacy collection "spyware", monetization abuse "adware", and propagation of rogue services "grayware". Additionally, we compared these miniapp malware with traditional malware to highlight their distinctive features.

[1]The instructions of requesting the dataset is published at https://minimalware.github.io/

II. BACKGROUND

A. Resources Vulnerable to Miniapp Malware

Super app platforms, which serve vast numbers of users, have become prime targets for malware distribution due to their extensive user base that significantly magnifies the potential security impact of such threats. With millions of users interacting daily within these ecosystems and their data being accessed by miniapps, certain super-app-specific resources are particularly vulnerable to cyber attacks. These include (1) user account data hosted and shared with third parties by super apps, (2) the advertisement-based monetization mechanisms that can be abused for financial gain, and (3) the expansive social networks that enable malware to spread rapidly, potentially compromising large numbers of users in a very short period of time.

Account data: privacy harvesting. Unlike traditional platforms such as personal desktops, which collect user identity data in a distributed and localized manner, super apps operate as fully-fledged applications that centralize and store significant amounts of user data, including phone numbers, nicknames, and addresses, in the cloud. This centralized data collection is integral to their core services, enabling miniapps to access this information for tasks such as account registration and online food delivery. While this unified data access mechanism is convenient for both the platform and developers, it also creates a substantial risk for large-scale data breaches if miniapps are allowed unrestricted access to sensitive user information. To mitigate this, the super app platforms have built an additional layer of security mechanisms called "authorization scope" [25], akin to permission-based data access in Android and iOS. However, malicious miniapps may still seek to bypass these grant-based mechanisms by invoking APIs that are able to collect other personal data not protected by authorization scopes such as device information and system versions to perform user tracking in background, or simply circumvent the authorization process by inducing users to manually enter the sensitive information, causing privacy leaks.

Advertisement mechanism: monetization abuse. In addition to generating revenue from users through paid services, miniapps can earn profits directly from the platform via

the pay-per-click advertising mechanism, utilizing APIs or UI widgets to embed ads. The platform then compensates the miniapp based on the "traffic", a metric based on how many users visit the miniapp [4]. According to a third-party report [11], a miniapp receives about 2 cents per click on a banner advertisement, and 7 cents per click for video advertisements. Since platforms retain 50% of the revenue generated from advertisements, this advertising model has become a significant revenue stream for both miniapps and the platforms themselves [15]. As a result, malware developers are incentivized to manipulate traffic metrics, leading to artificially inflated advertisement interactions to maximize financial gains.

Social network: rogue service propagation. In super apps, the miniapps can be and are encouraged to be shared among users' social networks, including friends, group chats, and users' personal timelines. This differs remarkably from traditional apps and is facilitated not just with URLs but also with meta-information such as miniapp names, thumbnails, and descriptions. To further encourage users to try out shared miniapps, the miniapp is automatically downloaded and executed within the super app, without requiring the user to enter an app store, download, and manually install the package. The simplified click-to-use and install-less mechanism of miniapps enables rapid spread among users, which unfortunately also serves as an effective distribution channel for malware. This unfortunately also results in the propagation of rogue services, and poses legal liability issues for the platforms due to the challenges in detecting and regulating these services.

B. The Super App Vetting Mechanism

In face of the security risks posed by miniapp malware, super app vendors have adopted mandatory vetting mechanism on all miniapps submitted to their stores, involving both code and content vetting.

Code vetting. Miniapps must undergo a thorough review process before they can be distributed, as mentioned in a study by Lu et al. [48]. In accordance with regulations outlined in the specified reference [36], once malicious payloads are identified, miniapps must either eliminate those payloads or be removed from the market. This vetting mechanism is particularly effective as miniapps are distributed exclusively through official markets, unlike desktop and Android apps which have alternative distribution channels.

Content vetting. In addition to code vetting, super apps also place a strong emphasis on content security. Given their vast social networks, the dissemination of rogue contents and services within these networks could expose vendors to significant legal liabilities. Therefore, miniapp contents are rigorously vetted. For example, the operational regulations for miniapps [36] prohibits content that includes sexually explicit material, hate speech, violence, terrorism-related content, and illegal activities.

```
1  <!--pages/index/index.js-->
2  onLoad: function(e) {
3    wx.login({
4      success: function(e) {
5        n.getState();
6        ...
7  },
8  getState: function() {
9    var e = this;
10   wx.request({
11     url: a.versionUri, //malicious domain
12     success: function(a) {
13       //set state according to the data
14     }
15   });
16 },
17
18 <!--pages/add/add.wxml-->
19 //This is benign path
20 <view wx:if="{{state===0}}" class="p">
21   <view class="w_view">
22     <navigator class="w_list" url="{{item.url}}"
       ↪ wx:for="{{lists}}">
23       <image class="w_icon"
         ↪ src="{{item.icon}}"></image>
24       <image class="w_text"
         ↪ src="{{item.text}}"></image>
25       ...
26     </navigator>
27   </view>
28  </view>
29 //This is malicious path
30 <web-view src="weburl"
   ↪ wx:elif="{{state===1}}"></web-view>
```

Figure 1: Code example for offline payload controlling

C. Evasive Techniques Against Vetting

Given that existing vetting mechanisms require miniapps to pass a security check before release, it is crucial for malware developers to adopt evasive tactics to circumvent these checks, such as by submitting benign code for vetting but later updating to malicious versions. To bypass vetting, these malware dynamically change their program behavior, either through the APIs invoked or the contents displayed. To achieve this, they leverage various interpreter-based JavaScript libraries and APIs as a convenient way to execute arbitrary sequences of bytecode or JavaScript strings. Although WeChat disables various JavaScript APIs such as `eval()` to ensure such dynamic hot update is not feasible for miniapps, there are cases where malicious developers attempt to perform hot update via third-party libraries, such as using interpreters (e.g., `evil-eval` [17]) and sandboxes such as `vm` [34] and `vm2` [33]. In response, since July 2022, WeChat prohibits the usage of a list of JavaScript libraries enabling dynamic code execution in miniapps [28], which provided a list of libraries prohibited from the miniapps. However, these malware may also implement their own interpreters [27], making the identification of miniapp malware even more challenging.

For content vetting evasion miniapps, attackers may exploit conditional display features supported by miniapps alongside dynamic updating. For instance, as illustrated in Figure 1, the malware implements a `<web-view>` in line 30 pointing to a malicious domain for displaying rogue contents. The visibility of this `<web-view>` component is controlled by the `state`

variable. This variable is set by `getState` at line 8 upon the miniapp loads, which fetches a file from `versionUri` (line 11). Consequently, the attackers may change the contents of the file hosted on `versionUri` to control the value of `state` variable, setting the value either to 0 or 1. Specifically, during vetting, the attacker may set the value to 0, which allows the miniapp to display the `view` at line 20, essentially showing a list of foods along with their nutrition information. After the miniapp passes vetting, the attacker can update the file to set `state` to 1, enabling the display of the `web-view` at line 30, which points to `weburl`, where the attackers can host and display their malicious content.

III. Identification

Motivated by the existence and potential security impacts of the evasive miniapp malware, we aim to automatically collect evasive miniapp malware for future research. In this section, we detail the scope of malware we collected, the methodology we employed, and provide an overview of the dataset, including a validation of the collection result.

A. Scope

In light of the significant security risks posed by miniapp malware, we aim to devise an automated method to collect and identify the evasive malware and facilitate future research. Unlike mobile and desktop malware, an established database for miniapp malware is not yet available, and thus identifying miniapp malware presents a unique challenge due to the absence of established ground truths. To tackle this challenge, we specifically focus on WeChat in this paper for three reasons. First, as a pioneering platform in the miniapp ecosystem, WeChat hosts over 4 million miniapps and serves 1.2 billion monthly active users [1]. This vast repository of miniapps not only ensures a rich dataset but also underscores the potential security ramifications of uncovering malware within it. Second, similar security and functionality features (particularly the vetting, data access, and monetization mechanisms) are also adopted by other miniapp platforms like Baidu [31] and Alipay [3], which suggests that methodologies developed for detecting malware on WeChat could be applicable across these platforms. Third, WeChat's miniapp ecosystem, with seven years of development, is among the most mature, offering a more diverse range of malware compared to newer platforms like Zalo [21].

However, collecting malware from WeChat is challenging because, unlike mobile and Linux malware, which benefit from established databases for monitoring and analysis, there is no such comprehensive ground truth for WeChat miniapps. Nevertheless, as miniapps can only be submitted to the official miniapp store where mandatory vetting is enforced, the malware must develop evasive techniques to bypass the vetting process by hiding malicious contents during the review and activating harmful behavior only after the miniapps are released. As such, this paper focuses specifically on this type of evasive malware (i.e., malware having evasive signatures),

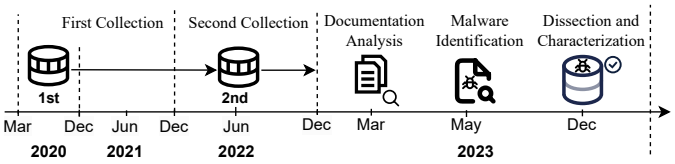

Figure 2: The timeline of the malware collection

a strategy similarly used in studies on web extensions [44] and Android malware [66], which has proven to be a powerful indicator for identifying malware samples and analyzing the malicious activities enabled by their evasive behaviors.

B. Methodology

To collect the malware exhibiting the evasive signature, we implemented a cross-check identification policy designed to minimize false positives in malware detection. First, the malware must demonstrate capabilities to evade the vetting process. Second, the malware must have been removed from the official miniapp store, indicating that the platforms have recognized its malicious behavior and subsequently delisted it, thereby confirming its status as malware. To collect malware that satisfy these two criteria, our collection comprises two steps: gathering removed miniapps, and identifying with evasive characteristics.

Step 1: Gathering removed miniapps. To capture these removed miniapps, we perform a longitudinal monitoring of the WeChat miniapp store's inventory. As illustrated in Figure 2, the collection for delisted miniapps was carried out from March 2020 to December 2022, spanning two cycles of miniapp collection. Initially, in 2020, we collected the available miniapps from the miniapp store. In 2021, we scaled up our collection efforts by incorporating MiniCrawler [65], and completed the first round of miniapp collection in June 2022, amassing 4,595,680 miniapps. Immediately after that, we revisited the miniapp store, querying the miniapp market whether the appIDs of the miniapps collected in the first round still exist. By the end of 2022, we noticed that a significant number of miniapps, $360,467$ in total, had been delisted.

Step 2: Identifying with evasive characteristics. To circumvent vetting, a malware can hide either the code to execute malicious behavior or the resources to display malicious contents, which can be categorized as code vetting evasion or content vetting evasion. However, either of these two approaches leave trace in the code, as malware developers need to implement modules that allow them to control the behavior of their malware after the malware passes vetting. By examining malicious examples exhibiting evasive behavior, the key to identify such malware is to detect the implementation of malicious dynamic code execution and content rendering.

1) Detecting malicious dynamic code execution. The capability for a miniapp to execute an arbitrary JavaScript code string is vital for evasive malware to perform dynamic code execution, as it allows malware developers to send malicious

```
1  Us = function() {
2    function t(e) {
3      var r = e.vm, n = e.key, o = e.value, a =
       ↪  e.parent;
4      Mi(this, t), this.value = o, this.vm = r,
       ↪  this.key = n;
5      var i = Ns(n, a);
6      this.root = i.root, this.path = i.path, this.dep
       ↪  = new Rs(), Ps(o, "__ob__", this),
7      Array.isArray(o) ? ((ks ? Is : Cs)(o, Ds, js),
       ↪  this.observeArray(o)) : this.walk(o);
8    }
9    return Ri(t, [ {
10     key: "walk",
11     value: function(t) {
12       for (var e = ft(t), r = 0; r < e.length; r++)
         ↪  qs({
13         vm: this.vm,
14         obj: t,
15         key: e[r],
16         value: t[e[r]],
17         parent: t
18       });
19     }
20   }, {
21     key: "get",
22     value: function() {
23       Rs.target && Fs.push(Rs.target), Rs.target =
         ↪  this;
24       var t = this.getter.call(this.vm, this.vm);
25       return Rs.target = Fs.pop(),
         ↪  this.cleanupDeps(), t;
26     }
27   }, {
28     key: "evaluate",
29     value: function() {
30       this.value = this.get(), this.dirty = !1;
31     }
32   },
33   ...
34   ])
35 }
```

Figure 3: Code example of VM-based obfuscation for evasive library

code to the vetted miniapps, transforming vetted miniapps to malware. As a dynamic language, JavaScript has abundant libraries to support such dynamic execution, including JavaScript interpreters (e.g., `eval`), and sandboxes (e.g., `vm`). Witnessing this situation, WeChat announced in 2022 that it will prohibit the use of such libraries that allow miniapps to perform hot update [28], which was also confirmed by our observation. However, the developers may be aware of the risk for being detected, and thus they may obfuscate their code against the vetting process. As shown in Figure 3, the script is indeed deeply obfuscated, with the exported functions constructed as key value pairs, containing functions such as `walk` (line 10), `get` (line 21), and `evaluate` (line 28). These functions essentially establish a tree containing multiple nodes, allowing miniapps to dynamically evaluate node values.

Fortunately, the complexity of the evasive techniques suggests that malware developers are less inclined to implement these behaviors from scratch, but instead more inclined to adopt existing code snippets to support such evasive behavior. For instance, despite the fact that malware may integrate minified or obfuscated libraries that are prohibited due to dynamic execution capabilities, the APIs provided to the end developers remain similar to their original counterparts. Thus, we still may identify these API usages based on the function signatures. As such, we first examined through the official announcements from WeChat about libraries deemed as capable for miniapps to implement hot update to evade code vetting, resulting in a list of libraries. Further, we extracted the signatures of APIs in each library for executing strings as JavaScript code. With these signatures, we scanned the delisted miniapps for invocations to these APIs, as well as usage of these libraries.

2) Detecting malicious content rendering. Aside from bypassing code vetting with dynamic code execution, malware may also evade the vetting by entirely concealing malicious content from their static code, and then deliver the malicious contents from the cloud after the malware passes vetting. To do this, a miniapp needs to dynamically deliver malicious contents to a post-vetting miniapp, and then control which contents a post-vetting miniapp displays. Therefore, `web-view` and conditional display techniques are commonly used by these miniapp malware.

The `web-view` component is a special view in the miniapp layout, which displays contents in a domain designated by the developer. On the other hand, the conditional display allows the miniapp to switch displayed components based on a Boolean variable. When these two features are used together, it forms an ideal approach for evasive malware to hide contents. For example, the code in Figure 1 uses a single variable, `state`, whose value is fetched from their back-end servers, to control whether to display benign contents or to display malicious information in `<web-view>`. As these contents are hosted on third-party domains not affiliated with WeChat and can be easily updated without having to be vetted, the platform will not be able to detect these malicious contents if the attackers maintain a blank page when the malware is vetted, hiding the malicious contents which will be revealed after the malware passes vetting.

In summary, malware bypassing content vetting with conditional rendering commonly involves two key properties: (1) a web view that displays contents from an attacker-controlled domain, and (2) a variable that controls the display of the web view via `wx:if`. Based on these patterns, we statically analyzed each delisted miniapp's WXML file to identify the components controlled by variables in the lookout of a malicious web-view controlled by cloud-delivered variables.

C. Dataset Overview

With the evasive signatures we extracted from the two types of malware by investigating the malicious examples acquired from Tencent, in Feburary 2023, we deployed the malware identification on the removed miniapps using a server with 16 Intel Xeon Silver 4314 CPUs. The identification process was executed with 16 threads for over one month, resulting in $19,905$ malware samples.

Threats to Validity. To evaluate the accuracy of the malware detection, we sampled a total of 500 miniapps out of the

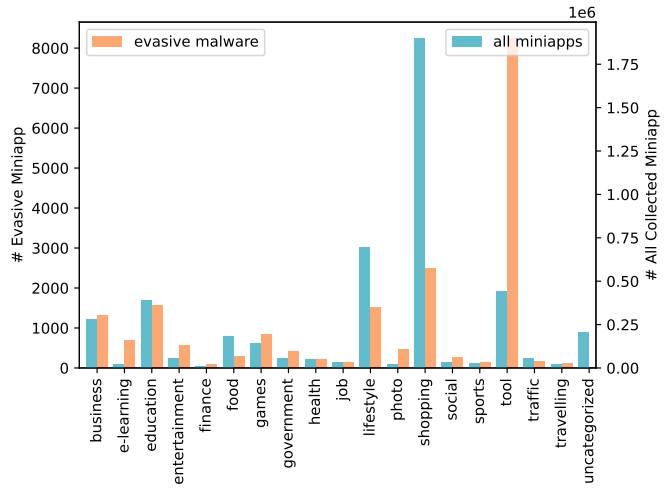

Figure 4: Categorial Distribution between Overall Miniapps and Miniapp Malware. The orange bars represent collected malware, and the blue bars represent all collected miniapps. Please note that the y-axis to the left are for the malware (orange), and the y-axis to the right are for all miniapps (blue).

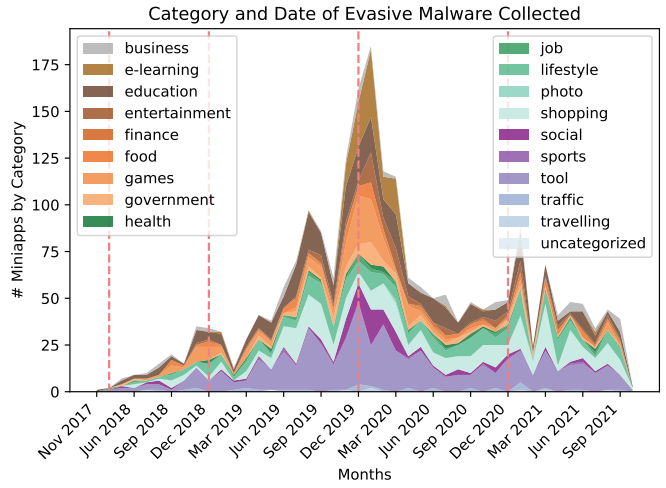

Figure 5: Amount of miniapps per category based on the creation time.

19,905 malware, and manually checked whether the miniapp involves evasive signatures. Despite that these files may be heavily obfuscated, we evaluated the semantics of the associated sensitive APIs to the best of our knowledge to identify whether it involves code signatures we extracted from the evasive samples that may cause code side effects (e.g., change of program behavior or displayed items). In total, 34 of the 500 sampled miniapps are associated with content vetting evasion, whereas the rest 466 sampled miniapps are identified as code vetting evasion. Among all these miniapps, 487 miniapps are correctly identified, whilst 13 miniapp were false positive

```
y.templateSettings = {
  evaluate: /<%([\s\S]+?)%>/g,
  interpolate: /<%=([\s\S]+?)%>/g,
  escape: /<%-([\s\S]+?)%>/g
};
  ...
y.template = function(e, t, n) {
...
var r = RegExp([ (t.escape || I).source, (t.interpolate
↪  || I).source, (t.evaluate || I).source ].join("|") +
↪  "|$", "g"), o = 0, i = "__p+='";
e.replace(r, function(t, n, r, a, u) {
  return i += e.slice(o, u).replace(T, R), o = u +
  ↪  t.length, n ? i += "'+\n((__t=(" + n +
  ↪  "))==null?'':_.escape(__t))+\n'" : r ? i +=
  ↪  "'+\n((__t=(" + r + "))==null?'':__t)+\n'" : a &&
  ↪  (i += "';\n" + a + "\n__p+='"),
  t;
}), i += "';\n", t.variable || (i = "with(obj||{}){\n" +
↪  i + "}\n"), i = "var
↪  __t,__p='',__j=Array.prototype.join,\+
  "print=function(){__p+=__j.call(arguments,'');};\n" + i
↪  + "return __p;\n";
try {
  var a = new Function(t.variable || "obj", "_", i);
} catch (e) {
  e = VM2_INTERNAL_STATE_DO_NOT_USE_OR_PROGRAM_WILL_FAIL.
  handleException(e);
  throw e.source = i, e;
}
var u = function(e) {
  return a.call(this, e, y);
}, c = t.variable || "obj";
  return u.source = "function(" + c + "){\n" + i + "}",
  ↪  u;
},
```

Figure 6: Code example for self-implemented JS/WXML processing

cases. Among these 13 miniapps, 10 are identified as evasive incorrectly because the miniapp implements the API `evaluate` to "allow users to evaluate purchased products", which is not implemented to perform execution of JavaScript code as strings to replace the prohibited `eval` library. 3 of them are identified as content vetting evasion because they involve web views controlled by variables, but they are displaying videos upon users' requests instead of attempting to evade vetting. That is, we do not claim that the malware samples we identified are 100% accurate, but we will explicitly identify these cases we evaluated together with the validation notes in the dataset to help researchers to reproduce our findings.

Evasive behavior categorization. Notably, our validation on the sampled miniapps have revealed different approaches adopted by miniapps for the evasion. In summary, among the 500 miniapps, 451 miniapps contain a file called `vendor.js`, which is commonly used in web development to include and pack third-party libraries or dependencies. As this file is essentially a compilation of multiple libraries, each miniapp's `vendor.js` can be different. Among these 451 miniapps, 32 involve an object called `Watcher` which is designed to interact with the DOM tree, 16 involves regular expression to match WXML tag format, and there is even one that leaves all variable names and code comments in the code, from which we notice copyright information of eval5.min.js, a library prohibited by WeChat but

Category	API/Library	#	Description
E	evaluate	18,428	Signature API for executing strings
	eval	18,112	API for executing JS code
	runInNewContext	46	API for executing code in sandbox
	runInContext	20	API for executing code in sandbox
	vm	69	Library of sandbox execution
	eval5	5	Library alternate of `eval`
	evil-eval	2	Library alternate of `eval`
	vm2	2	Library of sandbox execution
S	showShareMenu	6,022	Toggles share miniapp button
	onShareAppMessage	1,338	Invoked when miniapp is shared
	onShareTimeline	61	Invoked when shared to timelines
R	navigateToMiniprogram	10,519	Inter-miniapp communication

Table I: Sensitive APIs Invoked by the Collected Malware. E: evasive, S: share, R: redirection

imported as code snippets into `vendor.js` by the miniapp. Moreover, 3 miniapps even encode the file names, such as `7C56DE058DE715DF1A30B602F2364CA4.js`. As shown in Figure 6, the script implements `templateSettings` at line 1 that uses regular expressions to identify WXML tags by matching the "<" and the ">" globally, and then implements `template` at line 7, which is a function that eventually concatenates JavaScript functions for execution in line 11. Additionally, 2 miniapps integrate `dw_shared.js`, which breaks down the traversal of AST trees and expression evaluation as JSONObject in the return value.

For the content vetting evasion malware, we found that server-controlled data is widely used to show both benign contents before vetting and malicious contents after vetting. In the example shown in Figure 1, the malware fetches `item` from the malicious domain and displays nutrition knowledge (which is irrelevant from the actual gaming service that the miniapp provides), such as calories of foods. After the server-controlled variable `state` is changed to 1, the malware displays the web view to an external domain of a game with in-game purchasing. However, this miniapp is registered under the "information query" category as a tool rather than as a game or shopping app, where the platform does not require the developer's commercial certificates. By disguising itself as a non-commercial tool to avoid stricter vetting processes, this malware's behavior is categorized as "non-conformation of category", which also violates platform operation regulations.

Service categories of the collected malware. As illustrated in Figure 4, we are surprised to find that there are significantly more malware in the Tool category, compared with overall miniapps where the Shopping miniapps is the top-hit category. By scrutinizing the official miniapp categories and subcategories [30], we notice that almost all subcategories under the Tool category do not require developers to submit any qualification certificates, such as the certification for providing commercial license. On the contrary, most of the Shopping miniapps are required to submit these commercial licenses along with other types of proof, and thus the vetting process for non-tool miniapps are much stricter. Therefore, it is highly possible that malware developers deliberately choose the categories involving minimum requirements on the license or the identities of developers to evade vetting and deploy their malware with minimum cost.

Creation time of the collected malware. After we finished the initial collection of miniapps, we also started crawling meta information displayed in the "details" page of miniapps since June 2022. Unfortunately, the interface to crawl such information has strict rate limit, and we were only able to crawl the meta data for 829,288 miniapps by the end of the revisiting process in December 2022. Among these miniapps, we found 2,257 miniapps to be malware. Unfortunately, by the time of December 2022, all the rest malware we included in the dataset was already removed from the market, and thus we were unable to retrieve the meta information of the rest of the malware. However, we still made some interesting findings that, as shown in Figure 5, attackers started to release malware to the miniapp ecosystem since 2017, which is the year when the miniapp ecosystem initially debuted. During the next two years, the published malware steadily growed, and peaked in January 2019. Additionally, this figure also shows that tool-based miniapps continued to be the predominant category among the malware samples, as illustrated in the categories of collected malware.

IV. Dissection

In this section, we dissect the malware based on their post-vetting lifecycle, which is broken down into three key stages as illustrated in Figure 7: propagation to victim users (§IV-A), activation of malicious behaviors (§IV-B), and the execution of malicious payloads (§IV-C).

A. Malware Propagation

To propagate the miniapps to victims, it is intuitive that the attacker may bypass the vetting and release the malware to the miniapp store, waiting for victims to click and use the malware published on it. However, the infection of malware may be significantly augmented by the rich features supporting miniapp sharing among user social networks and cross-miniapp redirection between miniapps.

(I1) Miniapp store discovery. The most intuitive way for a malware to be distributed is to wait until the malware be discovered by users in the built-in miniapp store directly. These malware often involve inducing miniapp names and descriptions, such as "play the game and earn money", or "your churning tool for coupon codes", to attract potential users.

(I2) Sharing among social networks. As the super app allows and encourages miniapps to be shared among the user's social network, e.g., group chats, with convenient share-related APIs, malware can exploit this sharing capability. By customizing the information displayed to users, malware can present more enticing content to attract potential victims. In some cases, malware explicitly prompts users to share the

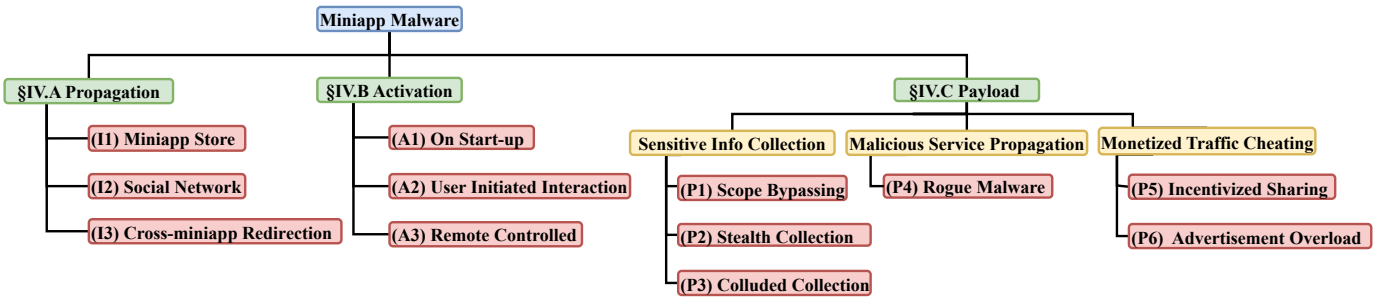

Figure 7: The Dissection of Miniapp Malware

miniapp or even offers financial incentives for sharing it with more group chats. As illustrated in Table I, over one-third of the collected malware customize the title and description of the shared links via share-related APIs to help customize shared information, tailoring the shared content to increase the likelihood of cascaded sharing and further dissemination.

(I3) Cross-miniapp redirection. As miniapps may redirect to each other with `wx.navigateToMiniProgram`, the cross-miniapp redirection become a new approach for miniapps to be distributed to victims. By developing multiple interlinked malware that redirect to each other, once a victim uses one malware, he/she may be redirected to all of these malware via redirection. Moreover, to induce users to "try out" more of these miniapps, these malware may adopt tactics similar to sharing-based malware, offering financial rewards for users trying out additional malicious miniapps. This strategy effectively bundles the malware, increasing the likelihood of distribution to victims' devices through these "bundle-sales" activities.

B. Activation

After the malware reaches the victim users, the next step is to execute malicious payload on the victim devices. However, being evasive by nature, these malware do not necessarily exhibit malicious behaviors immediately, but only under certain conditions. Thus, the activation of malicious behavior can be categorized into those executed upon start-up, after user initiates certain interaction, and via remote control.

(A1) On start-up. Among the various miniapp lifecycle callback functions, `onLoad`, `onLaunch`, and `onShow` are the most common ones, particularly for those malware attempting to stealthily collect user information. These functions allow malware to immediately gather and transmit tracking data to their back-end servers as soon as the miniapp is launched. Other than stealth privacy collection, these lifecycle functions include a default parameter `option`, which contains various context information such as URL parameters or appId, if the user enters this miniapp from other miniapps or via URL. This feature enables miniapps to give user financial benefit under induced sharing or collection scenario. For example, a miniapp induces Alice to share the miniapp and invite new customers, promising financial rewards or coupons for both Alice and the new users. As such, the miniapp generates a sharing link with parameters such as `&userid=`. When a new customer Bob enters the miniapp with this link, the miniapp can fetch this parameter in the lifecycle function to confirm Alice's user ID, thereby ensuring that Alice receives the reward for sharing the miniapp.

(A2) User triggered activation. The user-triggered activation is closely related to the infection of malware, which comprises two major approaches: via sharing (**I2**) and via redirection (**I3**). When the user shares a miniapp, callback functions such as `onShareAppMessage` is triggered to allow the miniapps to customize the shared links with icons, titles, and descriptions to induce more victims to use the malware. When the user clicks on an arbitrary component associated with `navigateToMiniProgram`, the malware may now directly redirect the users to external malware to cause more impacts.

(A3) Remote controlled. Remote-controlled activation is generally used by malware aiming to bypass code vetting with hot update and those distributing rogue services via content vetting bypassing, as they typically use variables or codes for execution that are fetched from remote servers to control the behavior of the miniapp. By doing so, the attackers can freely control the malware and then execute malicious payloads.

C. Payload Execution

Next, we illustrate the practical impacts of the collected malware, by first categorizing the malicious behaviors commonly employed by miniapp malware, and then presenting real-world malware cases uncovered through a combination of automated scanning and manual confirmation.

Malicious behavior categorization. Although the behaviors of evasive malware can be highly varied, in this paper, we particularly focus on the payloads identified by official regulations as malicious, specifically those that negatively impact platforms and users. To identify and categorize such malicious payloads, we performed a top-down analysis based on the regulations outlined by the WeChat platform [36].

While violating the official code of conduct is a necessary criterion for defining miniapp malware, it is important to note that not all such behaviors are of interest for our study. Some violations are either too vague or do not pose significant risks

to end users, despite being against the official guidelines. For instance, our taxonomy does not include intellectual property violations. Although this is a common issue in app stores, there is currently no solid metric for determining the extent of similarity with existing miniapps that would constitute a violation.

To refine our focus, we reviewed behaviors considered as violations of operational rules and involved three security researchers to help categorize those with clear and measurable malicious signatures. This process resulted in six categories as shown in Table II, including authorization bypass, stealth privacy information collection, collusion, rogue malware, incentivized sharing, and advertisement overload.

Payload scanning. To illustrate the concrete impact of these malware, we proceed to identify real-world cases that involve these malicious payloads from our $19,905$ evasive malware. To do so, we extracted signatures from publicized malware examples [9], as well as examples we obtained during our interaction with WeChat security teams. These signatures involve both static code signatures (such as invocation to certain APIs) and content-related signatures generated by a hierarchical keyword list. Then, we manually sampled 5 miniapps in each of these 6 categories. Below, we present the practical impact of these malware with real-world cases.

1) Sensitive information collection. Compared with mobile systems, super apps offer miniapps a wider range of privacy-sensitive data to facilitate their services, particularly the account and phone information that users have provided to the super apps, the acquisition of these cloud-hosted data are protected with a permission-based mechanism called authorization scopes to prevent abuse. These scopes are essential to ensure that users know and thus may grant or reject information accesses initiated by miniapps. However, malware exploiting the user data still attempt to harvest these types of sensitive information, either by directly inducing users or circumventing the grant mechanism with a single miniapp, or collaborating with other malicious miniapps to receive sensitive information via cross-miniapp channels.

(P1) Authorization bypass (single miniapp). To collect user information, the most straight-forward way is to use official APIs, but these official APIs will prompt the users to grant access first. Thus, to increase the likelihood of users consenting, the first variation of explicit collection malware will display texts such as *please provide your contact info so we can reach you later*, or *provide your phone number for log in*, to encourage users to grant the access. In addition to using official APIs, the second variation of these malware requires users to enter their private data manually into text boxes. Since official APIs are not used in this case, the collection process does not trigger an authorization dialog, making it less apparent to users. Those with limited knowledge of the miniapp platform's authorization mechanisms may unknowingly enter their information, which is then collected without being monitored by the platform, making it more difficult to detect.

	Category	Sub Category	# Miniapps	# Families	%
P1	Auth. Bypass	-	4,360	48	21.91%
P2	Stealth Collection	getSystemInfoSync	1,078	17	5.42%
		getSystemInfo	192	22	0.96%
		getScreenBrightness	1	1	0.01%
		getDeviceInfo	1	1	0.01%
		getClipboardData	2	2	0.01%
P3	Collusion	Account info	17	2	0.09%
		Password	16	2	0.08%
		User ID	33	6	0.17%
		User Name	7	2	0.04%
		Extradata	23	3	0.12%
		Phone	18	5	0.09%
		Address	1	1	0.01%
		Userdata	1	1	0.01%
		Vehicle Plate	2	1	0.01%
P4	Rogue Malware	Web Earning	4,105	41	20.63%
		Redpocket	1,202	29	6.04%
P5	Incentivized Sharing	Pyramid Selling	5,040	38	25.33%
		Induce Share	2,167	31	10.89%
		Forced Share	1,456	28	7.32%
P6	Ad Overload	-	420	30	2.15%

Table II: Breakdown of malicious payloads of evasive malware

(P2) Stealth collection (single miniapp). By examining the privacy data protected by authorization scopes, we found that many APIs used for acquiring system and device information are not protected by permission scope, allowing malware to collect these information unnoticed. For example, access to clipboard, system information, and screen information is not protected with scopes in WeChat, whereas they are commonly used for user tracking. As listed in Table II, we identified 5 such APIs from the official API lists that may provide information for tracking unique users for device fingerprinting. As shown in Figure 9, at line 2 and line 9 separately, the code snippet obfuscates the collection of information such as phone model, screen information, and system information. Then, these information is encoded and sent to back-ends, which may facilitate event tracking [37], causing potential privacy issues as reported by many recent works in web security community [50], [53], [51], [38]. Approximately 6% of the collected malware use this API to gather such information via the synchronous or asynchronous versions of the API, with a few instances even collecting data from the clipboard.

It is worth noting that even for the resources protected by permission scope, the authorization is only granted on the first time, and thus if a user is redirected to the miniapp by mistakes, the miniapp will be able to stealthily collect granted data upon the miniapp launches, potentially against the user's intentions. For instance, Figure 8 shows a miniapp attempting to collect users' system information upon start at line 2 to line 5, and check whether the location permission has been granted at line 11. If the user had previously granted this permission, then each time the miniapp is launched, whether intentionally by the user or through redirection by another miniapp, this miniapp will immediately collect the location data upon launch to track the user's geographical information.

```
try {
    var on = wx.getSystemInfoSync();
    K.br = on.brand, K.pm = on.model, K.pr =
    ↪ on.pixelRatio, K.ww = on.windowWidth, K.wh =
    ↪ on.windowHeight,
    K.lang = on.language, K.wv = on.version, K.wvv =
    ↪ on.platform, K.wsdk = on.SDKVersion,
    K.sv = on.system;
} catch (o) {}
return wx.getNetworkType({
    success: function(n) {
        K.nt = n.networkType;
    }
}), wx.getSetting({
    success: function(n) {
        n.authSetting["scope.userLocation"] ?
        ↪ wx.getLocation({
            type: "wgs84",
            success: function(n) {
                K.lat = n.latitude, K.lng = n.longitude,
                ↪ K.spd = n.speed;
            }
        }) : D.getLocation && wx.getLocation({
            type: "wgs84",
            success: function(n) {
                K.lat = n.latitude, K.lng = n.longitude,
                ↪ K.spd = n.speed;
            }
        });
    }
}),
```

Figure 8: A miniapp stealthily collecting location upon start

(P3) Colluded collection (multiple miniapps). In addition to using APIs to fetch information from the platform, miniapps may also transmit sensitive data via cross-miniapp communication channel, which poses a threat to the miniapp ecosystem. This practice can undermine the data authorization model. Since WeChat does not enforce strict protection on data transmitted between miniapps, a "privileged" miniapp that has been granted access to sensitive data like a phone number can directly transmit the user's ID and phone number to a third-party miniapp through redirection, without requiring additional user confirmation. However, the fact that a user grants miniapp A access to their phone number does not imply consent for miniapp B to access the same data. As a result, miniapp B can potentially obtain privileged data without having to go through the proper data authorization process. To evaluate the prevalence of this issue, we collected the data transmitted and received as `extraData`. Our findings revealed that 18 miniapps were transmitting users' phone numbers via redirection, 33 miniapps were transmitting users' names, and 23 miniapps were transmitting the entire `extraData` payload obtained from previous redirections, leading to a cascading transmission of privileged data across multiple miniapps.

2) Malicious service propagation. A significant difference between super apps and traditional mobile systems lies in the convenient and versatile sharing features. In super apps, miniapps are designed to be easily shared among users' friends and group chats, encouraging wider propagation to increase revenue by reaching more users. However, this convenient and rapid, socially-oriented propagation can also

```
var p = [ {
    method: wx.getSystemInfo,
    infos: [ "brand", "model", "pixelRatio",
    ↪ "screenWidth", "screenHeight", "windowWidth",
    ↪ "windowHeight", "language", "version", "system",
    ↪ "platform" ...]
} ... ]
function s() {
    // execute all methods in p and return info of return
    ↪ value
}
function a(t) {
  var o = [ "brand", "model", "pixelRatio",
  ↪ "screenWidth", "screenHeight", "system", "platform"
  ↪ ];

  var n = t.reduce(function(e, t) {
    return o.indexOf(t.key) > -1 ? e + t.value + "," : e
    ↪ + "";
  }, "");
  _ = f.hex_md5(n.substring(0, n.length - 1)),
  ↪ l.setCookie({
    data: {
      shshshfp: {
        value: _,
        maxAge: 3153e3
      }
    }
  });
}
module.exports = {
    Jdwebm: function() {
        var e = s();
        w.all(e).then(function(e) {
            ...
            a((e = e.concat(t, o)).reduce(function(e, t)
            ↪ {
                return e.concat(t);
            }, []));
        }).catch(function(e) {
            console.log(e);
        });
    },
    CookieUtils: l
};
```

Figure 9: Malicious user fingerprinting with obfuscation

be exploited by malware, enabling it to spread quickly across a large number of users. Consequently, rogue malware that offers illegal services or violates platform regulations has become a critical issue within the super app ecosystem.

(P4) Rogue malware. Other than privacy collection, malware may also provide services against the platforms' regulation as well. More specifically, WeChat prohibits the services of "web earning" and "non-official lucky draw" for the potential financial concerns. A typical web earning fraud malware implements pages inducing users to share the miniapp to friends in return for a small amount of money, but when the user clicks on the "withdraw money" button, the malware simply redirects the user to a blank page. When the user finally realizes that the miniapp does not allow withdrawal, the malware had already been shared to numerous group chats. Also, there are 1,175 malware involving self-implemented "red packet" (a digital envelope filled with money) services, which is essentially a game where a user puts money for group members to draw a random amount. However, these services are not provided by the WeChat officially, but are implemented by these third-party malware instead, without

the guarantee that the users will eventually get the money. Thus, the non-official "red packet" services are prohibited by WECHAT due to financial fraud concerns.

3) Monetized traffic cheating. The monetization model of miniapps is unique compared with mobile apps, as super apps offer developers the chance to monetize through ad-based revenue models tied to app popularity. In response to this, a new breed of malware has emerged, exploiting this system to churn revenues directly from the platform. Unlike traditional app marketplaces such as Google Play, where fees may be associated with developer account registration and app publication, miniapp platforms often allow free account creation and the publication of multiple miniapps (up to 10 in the case of WECHAT [16]). This has led to a proliferation of traffic cheating miniapps that either manipulate metrics to artificially inflate popularity or induce users with excessive advertising, severely degrading the user experience. Such practices not only have potential financial ramifications but also risk damaging the reputation of miniapp platforms and adversely affecting users in terms of user experiences.

(P5) Incentivized sharing (manipulating miniapp traffic). Our analysis revealed a prevalent tactic among malware to incentivize users to share miniapps in group chats by offering reimbursements or discounts. This strategy is employed because sharing miniapps, especially those laden with advertisements, can significantly amplify "traffic", and thus increase revenue from embedded pay-per-click ads. To maximize this traffic, the malware often entices users with a nominal monetary reward, although typically far less than the profits accrued by the attackers, as compensation for their role in disseminating the miniapp. This form of malware bears resemblance to "internet earning" miniapps, with the primary objective of leveraging users to spread the miniapp across their social networks, thereby generating substantial traffic and, consequently, having financial gains from the super app platform.

(P6) Advertisement overload (manipulating Ad traffic). While miniapp platforms permit the integration of advertisements within miniapps, platform guidelines often highlight that these advertisements should neither be excessively intrusive nor obstruct essential app functionalities, so as to preserve the user experience [20]. However, there are two prevalent forms of malware that contravene these guidelines by overloading miniapps with advertisements. Given that many advertisement models are based on user clicks, some miniapps resort to overlaying essential content with interstitial advertisements to guarantee user engagement. Others might aggressively prompt users with advertisements in a way that even hinders the app's normal functionality. Beyond advertisements obstruction, certain malicious miniapps exploit the situation further by offering users incentives similar to Incentivized sharing malware (**P5**), such as cashback or discounts for interacting with these advertisements. This approach is notably prevalent in gaming miniapps, where, for example, users depleted of hitpoints (HP) might be prompted to watch an advertisements in exchange for additional lives or boost items.

Type	Data Category	API/Data	# Miniapps
Acquisition	User Information	getUserProfile	1,314
	Location Information	getLocation	4,870
		startLocationUpdateBackground	50
		startLocationUpdate	15
		getWifiList	31
	Bluetooth Access	openBluetoothAdapter	117
	Phone Information	addPhoneContact	1,198
		getPhoneNumber	403
	Microphone Access	startRecord	177
	Health Information	getWeRunData	72
Storage	Account Information	openid	3,029
		openId	1,336
		user_openid	172
		nickName	162
		avatarUrl	168
	User Information	$userInfo	2,794
		userInfo	2,680
		userinfo	310
		phone	306
		mobile	117
		city	2,234
		address	195
		username	205
		latitude	1,888
		longitude	186
	Device Information	$ip	2,776
		versionInfo	921
		aldstat_uuid	327
	Share Information	shareDate	776
	Cryptographic Keys	session_key	323

Table III: Sensitive Data Acquired and Accessed by the Collected Malware

V. CHARACTERIZATION

A. Malware Data Access Practices

Beyond identifying sensitive APIs indicative of malware activities, it's crucial to ascertain the types of data these malicious entities access and manipulate. To evaluate this, we split the sensitive APIs into two categories: APIs that acquire sensitive data, and APIs that can store data for future use. For the sensitive acquisition APIs, we scanned through the API list provided by WECHAT, and summarized 6 types of APIs that require user permission to access, which indicates that the data acquired by these APIs are more sensitive. For the storage APIs, we focused on the interactions with `setStorage`, a prevalent API among the malware in our dataset, used by 18,737 instances, that facilitates local data storage within miniapps for subsequent use. As this API stores data as a key value pair, we particularly extracted the keys, which are the names of the variables stored by the miniapps.

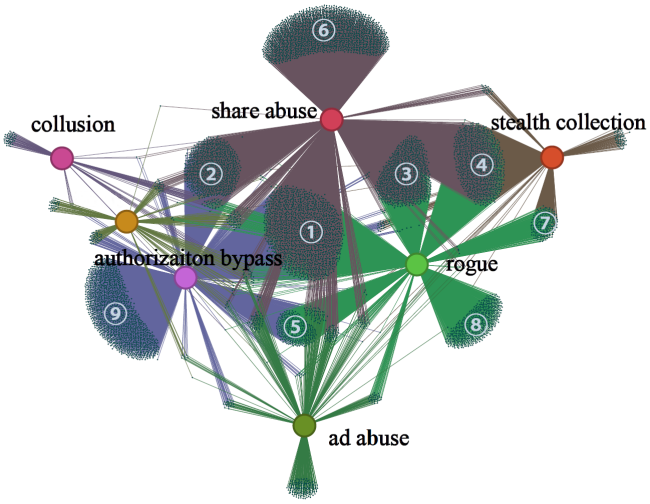

Figure 10: Cluster of miniapps sharing similar malicious payloads. The colored nodes (large) represent the malicious payloads, and grey nodes (tiny) represent malware. An edges from malware to a type of payload means that the malware contains such payload.

Then, we grouped these keys to the stored data, and involved three security researchers to evaluate the security impacts.

As delineated in Table III, approximately a quarter of the malware, was found to collect users' device location data. Additionally, these malicious miniapps often gather comprehensive user details and access the phone contacts of victims. Beyond exploiting APIs for accessing data managed by super apps, these malware extensively store information on devices, encompassing user account details, names, and addresses, as well as device specifics potentially utilized for user tracking. Moreover, some malware maintain logs of shared information to monitor their spread through social networks and store cryptographic data, such as `session_key`, to decrypt user information originally encrypted by super apps.

B. Malware Families

To understand the association between miniapps and malicious payloads, we visualize their correlation in Figure 10, where the large colored nodes represent the malicious payloads, and tiny dark nodes represent each malware, with edges connected to associated malicious payloads. We discover that there are four large clusters in the center of the figure associated with multiple payloads (①,②,③, and ④), as well as 2 in the edge of the figure associated with single malicious payloads (⑥ and ⑨). In this section, we further categorize the motivation of these malware into three types.

"Spyware": information collection. The most common type of malware falls into the category of privacy-sensitive information collection, which includes the induced collection of privileged data, stealth collection of device fingerprints for tracking users, and collusive data harvesting. As shown in Figure 10, both authorization bypass techniques to induce users to enter privacy information (②) and stealth fingerprinting methods (④) often involve the abuse of sharing. This is intuitive, as the impact of privacy data collection is maximized by harvesting information from a large number of victims. However, miniapp spyware differs from traditional spyware in several key aspects. Traditional spyware typically intercepts general data types on victims' devices, whereas miniapp spyware primarily targets specific privacy data managed by the miniapp platform under authorization scope, such as phone numbers, user account information, and addresses. Additionally, miniapp spyware is more comprehensive, collecting not only user data stored on devices and account data stored by the platform, but also social network information, such as group chat IDs, which can be exploited for further attacks such as social engineering.

"Adware": monetization abuse. Another prevalent malware family we identified exploits the platform's advertisement monetization model by artificially inflating traffic. These malware leverage the sharing feature to rapidly propagate, generating significant popularity in a short period for maximum click-based profits. By utilizing sharing features, these malware are often spread among a user's friends and group chats, where users are more likely to trust the sharer and click on the miniapp URL. To enhance their spread, these malware frequently employ malicious payloads involving sharing abuse and rogue content, particularly through schemes like internet earnings and non-authentic red packet services, which entice users to share the malware with more victims. As a result, sharing abuse constitutes one of the largest clusters associated with single payloads (⑥). The developers of these malware may choose to share a small portion of the profits with users, or they may deceive users by promising rewards that are never actually provided, such as claiming to offer money but not enabling withdrawals. Consequently, there is a high correlation between rogue content and sharing abuse payloads among the largest clusters (①, ③, and ④). Collectively, these adware families make up approximately half of the total families.

Compared with the traditional adware, the monetization abuse malware differs in that it primarily seeks to exploit the platform's popularity-based monetization mechanism for financial gain, rather than simply downloading and displaying advertisement content to victims. Interestingly, miniapp adware may collaborate with users by sharing a portion of the profits with them in the form of "discounts" or "cashback" via lucky draws. As a result, users may be less motivated to report the malware compared to traditional spamming malware.

"Greyware": rogue content. In addition to ad monetization and privacy harvesting, malware may also exploit the ecosystem to distribute rogue contents. The rogue contents blocked by the platform generally cause potential legal or financial fraud issues. Unfortunately, the victims may as well welcome these services, either because these malware satisfies users' illicit needs or these malware induces users with the promised rewards, making these malware distribute rapidly among social networks, which may explain why there are large clusters

in Figure 7 involving both rogue contents and share abuse (①, ③, and ④). Unfortunately, the "benefits" may instead inflict losses to users' privacy: among the three clusters, two either involve stealth device fingerprinting (⑨) or induces users to enter privacy information (④). Although the rogue malware in miniapp also may provide normal services to end users, resembling traditional greyware, now that these miniapp malware is released to an ecosystem with convenient sharing features and social network of up to a billion of end users, they may now propagate rapidly throughout the users' social network, affecting a large number of users in a very short period of time.

C. Malware Attacks and Evolution

Malware refers to software specifically designed to cause harm to computer systems, including those on desktops, mobiles, and IoT systems. Over the past few decades, malware has evolved significantly, adapting to new technologies and environments. During the early desktop era, malware primarily focused on activities related to fun and profit, such as stealing passwords through keyloggers, engaging in crypto mining to earn cryptocurrencies like Bitcoin, and deploying ransomware. The spread of malware during this period was relatively limited, often relying on methods like infecting systems via floppy disks or, later, USB drives. As the mobile era emerged, the focus of malware shifted towards the collection of highly sensitive user information. With data being regarded as highly valuable assets, varying across different contexts, malware targeted the collection of such data for strategic purposes. Additionally, with the advent of super apps and their diverse programming interfaces, malware evolved from simple file-infection viruses that relied heavily on floppy disks or USB drives to more sophisticated self-contained programs. These modern malware leverage social network propagation channels and deploy a variety of malicious payloads, making them far more versatile and dangerous.

We aim to provide a systematic view of malware running on various platforms and highlight the distinctions between malicious miniapps and malware found on other platforms, as illustrated in Table IV.

- **Malicious miniapps have more restricted capabilities.** Due to the stringent security measures implemented by super apps, malicious miniapps have limited capabilities. First, they are unable to access the underlying system interfaces since they operate within the confines of a native app, rather than directly on the system itself. Second, super apps do not provide interfaces that allow miniapps to spread through means like SMS or peripheral devices such as USB. Third, while miniapps can access files, they do not have direct access to the underlying disk. Instead, they are confined within the sandboxes created by the super apps. Lastly, malicious miniapps are unable to run in the background, as their execution is restricted when they are actively being used by the user.

Category	Item	Destop	Mobile	Miniapp
Capabilities	Invoke System Call [22]	●	●	○
	Accessing Network [8]	●	●	●
	Accessing SMS [32]	○	●	○
	Accessing Peripherals [13]	●	●	○
	Accessing Disks Directly [47]	●	●	○
	Running Background [24]	●	●	○
Infection	Market to Device [32]	●	●	●
	Web to Device [35]	●	●	●
	QRCode to Device [14]	○	●	●
	Wireless to Device [13]	●	○	○
	USB to Device [47]	●	●	○
	Email to Device [18]	●	●	○
	SMS to Device [32]	○	●	○
	App to Device [23]	●	●	●
Payloads	Information Collection [42]	●	●	●
	Rootkits [22]	●	●	○
	Spyware [8]	●	●	●
	Ransomware [35]	●	●	○
	Adware [12]	●	●	●
	Backdoor [23]	●	●	●
	Worm [47]	●	●	○
	Phishing (or Trojans) [18]	●	●	●
	Financial Charge [32]	●	●	●
	Bots and Botnets [10]	●	●	○
	Keylogger [24]	●	●	○
	Wiper [29]	●	●	○
	Hijackers [7]	●	●	○

Table IV: Comparison of malware on different platforms.

- **Malicious miniapps heavily rely on social networks.** In the past, malware primarily used networks and USB drivers to infect others during the desktop era. However, modern malicious miniapps heavily rely on social networks for spreading and causing harm. For instance, Tencent has implemented restrictions to prevent malware from sending SMS, and even WeChat has specific rules to prevent malware from exploiting the host app to infect other users. Our investigation have revealed numerous instances of malware deceiving users into sharing them with their friends and relatives. Additionally, we have observed that many malware collect users' information related to their social network, which can be used for launching phishing attacks.
- **Victims can be the super apps.** In the early days, there were no super apps; instead, operating systems like Android and iOS played similar roles by providing environments for malware to operate. However, the ultimate target of such malware was still primarily the end users, and they would often first compromise the OS and then use the compromised system to attack users. In the miniapp era, we observed that the major target of miniapp malware now involve both the users and the platforms, and in many circumstances, users mainly act as a means for malware propagation, especially for monetization abuse type of malware. Meanwhile, due to the sensitivity and amount of privacy data of millions of users stored by the super app platform, malware gains significantly more motivation for attacking the platform for large-scale privacy breaches.

VI. Discussion

A. Ethics Considerations

We have given the utmost consideration to ethics throughout our research process to ensure the well-being of end users, platforms (such as Tencent), and developers. First, although we have collected numerous malware samples for this study, we have kept them private and have not distributed them to the public. Following similar practice as in the Android Malware Genome Project [5], the dataset will be released only to the parties whose identities have been verified before granting the access. Second, in our study, we downloaded over 4 million miniapps. To prevent imposing a burden on Tencent's servers, we deliberately limited the download speed to a few seconds per miniapp, which is also the partial reason of why it took many months to collect all these miniapps. Third, despite the fact that all the malware samples we collected have already been removed from the WeChat's miniapp market, we have reached out to Tencent and shared our findings, including insights and heuristics developed during our study. We are now working with Tencent security teams for improved countermeasures and extended insights for detecting these malware.

B. Cross-platform applicability

While our paper focuses solely on one super app, namely WeChat, to reflect the current state of the art, the results and methods employed in our study are representative and generalizable. First of all, as popular super apps including Alipay, Tiktok, and Baidu commonly adopt JavaScript as the language for implementing miniapps, the concern of dynamic code execution and content rendering is applicable across platforms. For instance, other platforms such as Baidu and Tiktok also disable dynamic updating techniques (e.g., by disabling functionalities such as `eval`). On the other hand, super apps follow similar frameworks and share similar concerns regarding violation of rules from malware. For example, regulations against the 6 types of malicious payloads as listed in this paper can be commonly found in the rules of operation published by major super app platforms, including Alipay [2], Douyin [19], Baidu [26], and even Zalo [21], which is a Vietnamese super app.

C. Limitation

Our study does have certain limitations that need to be acknowledged. First, our detection of evasive malware is based on function signatures of prohibited libraries, and false positives may occur due to synonym issues. To validate this issue, we manually examined 500 cases to the best of our knowledge, and are actively working with the platform to cross-check our findings. Second, our approach is based on downloaded miniapp packages, which are at the front-end. However, due to the evasive nature of these malware, the malware may dynamically hide malicious contents without distributing them to the front-end by the time we tested the cases. As such, in this paper, we particularly focus on malware's capability to perform vetting evasion, cross-checked by the fact that the miniapp is taken down from the platform's miniapp store, to ensure that the discovered miniapps' maliciousness. To improve this line of work, we are actively working to extend the malicious semantics of evasive miniapp malware. Third, when seeking real-world malware samples, we adopted keyword-based heuristics to scan for certain types of malicious payloads, which may cause false-positives. However, the keyword-based scanning is for obtaining cases to demonstrate the impact, and we do not claim its comprehensiveness for payload categorization. Moreover, we manually confirmed each example in this paper that they are indeed malicious, and will continue improving our algorithms to generate robust malware categorization in the future.

In short, this paper aims to provide a dataset of evasive miniapp malware to facilitate related research in this field. Further, this paper presents a systematic categorization of miniapp malware's lifecycle and malicious payloads with real-world examples to demonstrate their impacts. We hope that this work further augments the security landscape of miniapps and super apps, and call for the community's action to contribute to this critical endeavor.

D. Mitigation

While the malware has been engaged in a continuous arms race against the platform to bypass vetting mechanisms and inflict losses to the platform and users, countermeasures can still be adopted by the platforms to thwart the efforts from the attackers. First of all, platforms can enforce forced execution of branches controlled by conditional variables, such as the flags for displaying dynamic content, so as to identify miniapps adopting dynamic content rendering to circumvent vetting. In the meanwhile, the platforms can enhance continuous monitoring, including continuous API tracking and user behavior tracking. If a miniapp incorporates a self-implemented interpreter to execute cloud-transmitted JavaScript code to change behavior, the self-implemented API for executing the code may be frequently invoked with parameters received from cloud via APIs such as `request`, which may cause change of API invocation pattern. Similarly, content-related evasive malware may involve web views displaying contents from attacker-controlled URLs to trap users in the web view displaying malicious contents, which may change user behavior pattern, e.g., victims may commonly stay in a certain page with a single web view for significantly more time than others. As such, by incorporating anomaly detection in APIs and user interaction pattern, the platforms may significantly thwart such evasive malware.

VII. Related Works

Super app security. Recently, super app security has gained significant attention with various efforts such as the demonstration of access control flaws and phishing attacks by Lu et al [48], race condition attacks by Zhang et al [63], undocumented API attacks by Wang et al [57] and also their cross-platform discrepancy attacks [56], [58], and the cross miniapp

request forgery attacks [61]. In addition to these attacks, there are also efforts to identify security threats [60] and develop proactive defense such as TaintMini [55], which identifies cross-language and cross-program data flows in mini-apps for privacy leaks detection, and the work by Wang et al. [59], which highlights the differences between the threat model of traditional browsers and super-apps. Compared to these efforts, our work is the first to perform a large-scale analysis to understand real-world miniapp malware attacks.

Malware attacks. Since the 1980s, the landscape of malware attacks has undergone significant transformation, adapting to technological advancements and finding new avenues for exploitation. In the era of mobile computing, malware has notably evolved to exploit the connectivity and functionalities inherent to mobile devices, targeting sensitive user data and leading to widespread financial and privacy ramifications [66], [43]. This evolution has necessitated the development of sophisticated defenses [54], [64], [52], [40], [41] and comprehensive research efforts [46], [49], [39], [45], [62] aimed at understanding and mitigating these modern malware threats.

Within the complex ecosystem of super apps, various forms of evasive malware have emerged, exploiting both the large user base and the extensive integration capabilities of these platforms. Despite the deployment of multiple defense mechanisms, these miniapps often leverage social networks to spread their malicious payloads, underscoring the dynamic and persistent nature of malware threats in the domain. This paper seeks to shed light on these challenges, highlighting the enduring impact and sophistication of malware in the digital age.

VIII. Conclusion

In this paper, we have presented a thorough three-year investigation, during which we analyzed and categorized $19,905$ malicious miniapp samples from a larger pool of over 4.5 million miniapps on the WeChat platform. We identified the malware that may evade vetting and release malicious payloads by identifying the invocation of sensitive API signatures generated from publicized malware cases, and reveals a significant concern: malicious miniapps effectively use social networks to increase their spread and harmful effects, posing notable risks to user privacy and the WeChat ecosystem's security. The insights from this study aim to inform the academic and tech communities about the changing landscape of threats within the miniapp ecosystem. By sharing our findings and the dataset, we hope to support the development of stronger protective measures against such malicious miniapps, enhancing the security and reliability of mobile super apps.

Acknowledgment

We thank the anonymous reviewers for their insightful feedbacks. This research was supported in part by NSF award 2330264. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the NSF.

References

[1] "55+ wechat statistics - 2022 update," https://99firms.com/blog/wechat-statistics/#gref.
[2] "Alipay mini program operation rules," https://opendocs.alipay.com/b/0a8w4g.
[3] "Alipay Miniprogram Center — Alipay ," https://opendocs.alipay.com/mini/developer/getting-started.
[4] "Analysis of types of advertisements and revenues in wechat mini-apps," https://developers.weixin.qq.com/community/develop/article/doc/000482ef31c32830526ee228e56413, accessed: 2023-12-26.
[5] "Android malware genome project," http://www.malgenomeproject.org/, (Accessed on 01/28/2024).
[6] "Biggest app stores in the world 2022," https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/, (Accessed on 04/30/2022).
[7] "Browser hijacking," https://www.kaspersky.com/resource-center/threats/browser-hijacking.
[8] "Darkhotel," https://en.wikipedia.org/wiki/DarkHotel.
[9] "Dissecting the violative miniapp cases," https://developers.weixin.qq.com/community/business/course/00080470cb41c0c6094ab3b785b00d.
[10] "Echobot," https://malpedia.caad.fkie.fraunhofer.de/details/elf.echobot.
[11] "Explaining the advertisement revenues of mini-apps," https://www.sohu.com/a/680423468_121708220, accessed: 2023-12-26.
[12] "Fireball – the chinese malware of 250 million computers infected," https://blog.checkpoint.com/research/fireball-chinese-malware-250-million-infection/.
[13] "Flame (malware)," https://en.wikipedia.org/wiki/Flame_(malware).
[14] "Grifthorse android trojan steals millions from over 10 million victims globally," https://www.zimperium.com/blog/grifthorse-android-trojan-steals-millions-from-over-10-million-victims-globally/.
[15] "Half of the ad revenues are demanded by wechat - "tencent" mini-games are gaining popularity," https://www.jiemian.com/article/2306249.htm, accessed: 2023-12-26.
[16] "How many miniapps can an entity register?" https://developers.weixin.qq.com/community/develop/doc/000484159a0308e4ffcf942ca56c00.
[17] "A javascript interpreter written in javascript," https://github.com/jkeylu/evil-eval.
[18] "Meet crowdstrike's adversary of the month for february: Mummy spider," https://www.crowdstrike.com/blog/meet-crowdstrikes-adversary-of-the-month-for-february-mummy-spider/.
[19] "Mini program operation rules," https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/operation/management/specification/standard.
[20] "Miniapp advertisement regulation violation and punishment," https://ad.weixin.qq.com/pdf.html?id=rynYA8o3f.
[21] "Miniapp censorship policy," https://mini.zalo.me/docs/zalo-mini-app-censorship-policy/.
[22] "More nefarious strain of zacinlo malware infecting windows 10 machines," https://www.eweek.com/security/more-nefarious-strain-of-zacinlo-malware-infecting-windows-10-machines/.
[23] "Mytob," http://virus.wikidot.com/mytob.
[24] "Olympic vision keylogger and bec scams," https://www.phishlabs.com/blog/olympic-vision-keylogger-and-bec-scams/.
[25] "Open Capabilities - User Authorization," https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/authorize.html.
[26] "Platform operation rules," https://smartprogram.baidu.com/opensourcedocs/operations/specification/.
[27] "Prohibiting miniapps to use interpreters? confronting tencent again," https://zhuanlan.zhihu.com/p/539725089.
[28] "Regarding the requirement of prohibiting miniapps to use javascript interpreters," https://developers.weixin.qq.com/community/minihome/doc/0000ae500e4fd0541f2ea33755b801.
[29] "Russia vs ukraine cyberwarfare: Lessons learned," https://www.researchgate.net/publication/364111929_Russia_vs_Ukraine_Cyberwarfare_Lessons_Learned.
[30] "Service categories opened for miniapps," https://developers.weixin.qq.com/minigame/product/material/.
[31] "Smart Miniprogram Platform — Baidu ," https://smartprogram.baidu.com/developer/index.html.
[32] "Toll fraud malware: How an android application can drain your wallet," https://www.microsoft.com/en-us/security/blog/2022/06/30/toll-fraud-malware-how-an-android-application-can-drain-your-wallet/.
[33] "vm2 - npm," https://www.npmjs.com/package/vm2.

[34] "vm.js: Javascript bytecode compiler," https://github.com/tarruda/vm.js/.

[35] "Wannacry," https://en.wikipedia.org/wiki/WannaCry_ransomware_attack.

[36] "Weixin mini program platform operation rules," https://developers.weixin.qq.com/miniprogram/en/product/.

[37] "What is mobile app event tracking?" 2023. [Online]. Available: https://mixpanel.com/blog/what-is-mobile-app-event-tracking/

[38] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 674–689. [Online]. Available: https://doi.org/10.1145/2660267.2660347

[39] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features," in *Network and Distributed System Security (NDSS) Symposium*, ser. NDSS 20, February 2020.

[40] M. Brengel and C. Rossow, "YARIX: Scalable YARA-based malware intelligence," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3541–3558. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/brengel

[41] B. Cheng, J. Ming, E. A. Leal, H. Zhang, J. Fu, G. Peng, and J.-Y. Marion, "Obfuscation-Resilient executable payload extraction from packed malware," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3451–3468. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/cheng-binlin

[42] R. Flores and L. Remorin, "Piercing the hawkeye: Nigerian cybercriminals use a simple keylogger to prey on smbs worldwide," *Trend Micro. June*, vol. 19, 2015.

[43] X. Han, N. Kheir, and D. Balzarotti, "The Role of Cloud Services in Malicious Software: Trends and Insights," July 2015.

[44] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in *22nd USENIX Security Symposium (USENIX Security 13)*. Washington, D.C.: USENIX Association, Aug. 2013, pp. 637–652. [Online]. Available: https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/kapravelos

[45] E. Kim, S.-J. Park, S. Choi, D.-K. Chae, and S.-W. Kim, "Maniac: A man-machine collaborative system for classifying malware author groups," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2441–2443. [Online]. Available: https://doi.org/10.1145/3460120.3485355

[46] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does every second count? time-based evolution of malware behavior in sandboxes," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[47] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security and Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[48] 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*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 569–585. [Online]. Available: https://doi.org/10.1145/3372297.3417255

[49] A. Mantovani, S. Aonzo, X. Ugarte-Pedrero, A. Merlo, and D. Balzarotti, "Prevalence and impact of low-entropy packing schemes in the malware ecosystem," in *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

[50] J. R. Mayer and J. C. Mitchell, "Third-party web tracking: Policy and technology," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 413–427.

[51] P. Pearce, A. P. Felt, G. Nunez, and D. Wagner, "Addroid: privilege separation for applications and advertisers in android," in *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 71–72. [Online]. Available: https://doi.org/10.1145/2414456.2414498

[52] R. Petrik, B. Arik, and J. M. Smith, "Towards architecture and os-independent malware detection via memory forensics," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 2267–2269. [Online]. Available: https://doi.org/10.1145/3243734.3278527

[53] F. Roesner, T. Kohno, and D. Wetherall, "Detecting and defending against Third-Party tracking on the web," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. San Jose, CA: USENIX Association, Apr. 2012, pp. 155–168. [Online]. Available: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/roesner

[54] L. Shi, J. Ming, J. Fu, G. Peng, D. Xu, K. Gao, and X. Pan, "Vahunt: Warding off new repackaged android malware in app-virtualization's clothing," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 535–549. [Online]. Available: https://doi.org/10.1145/3372297.3423341

[55] 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*, ser. ICSE '23. IEEE Press, 2023, p. 932–944. [Online]. Available: https://doi.org/10.1109/ICSE48619.2023.00086

[56] 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 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 6629–6646. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/wang-chao

[57] ——, "Uncovering and exploiting hidden apis in mobile super apps," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2471–2485. [Online]. Available: https://doi.org/10.1145/3576915.3616676

[58] ——, "Rootfree attacks: Exploiting mobile platform's super apps from desktop," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 830–842. [Online]. Available: https://doi.org/10.1145/3634737.3645001

[59] Y. Wang, Y. Yao, S. Shi, W. Chen, and L. Huang, "Towards a better super-app architecture from a browser security perspective," in *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps*, ser. SaTS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 23–28. [Online]. Available: https://doi.org/10.1145/3605762.3624427

[60] Y. Yang, C. Wang, Y. Zhang, and Z. Lin, "Sok: Decoding the super app enigma: The security mechanisms, threats, and trade-offs in os-alike apps," 2023. [Online]. Available: https://arxiv.org/abs/2306.07495

[61] 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*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3079–3092. [Online]. Available: https://doi.org/10.1145/3548606.3560597

[62] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad, "An inside look into the practice of malware analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3053–3069. [Online]. Available: https://doi.org/10.1145/3460120.3484759

[63] 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 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 1597–1613. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/zhang-lei

[64] 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*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 757–770. [Online]. Available: https://doi.org/10.1145/3372297.3417291

[65] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin, "A measurement study of wechat mini-apps," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 2, jun 2021. [Online]. Available: https://doi.org/10.1145/3460081

[66] Y. Zhou and X. Jiang, "Dissecting android malware: Characterization and evolution," in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 95–109.