MiniCAT: Understanding and Detecting Cross-Page Request Forgery Vulnerabilities in Mini-Programs

Zidong Zhang
School of Cyber Science and Technology, Shandong University
Qingdao, China
kee1ongz@mail.sdu.edu.cn

Qinsheng Hou
Shandong University; QI-ANXIN Technology Research Institute
Qingdao, China
houqinsheng@mail.sdu.edu.cn

Lingyun Ying*
QI-ANXIN Technology Research Institute
Beijing, China
yinglingyun@qianxin.com

Wenrui Diao*
School of Cyber Science and Technology, Shandong University
Qingdao, China
diaowenrui@link.cuhk.edu.hk

Yacong Gu
Tsinghua University; Tsinghua University-QI-ANXIN Group JCNS
Beijing, China
guyacong@tsinghua.edu.cn

Rui Li
School of Cyber Science and Technology, Shandong University
Qingdao, China
leiry@mail.sdu.edu.cn

Shanqing Guo
School of Cyber Science and Technology, Shandong University
Qingdao, China
guoshanqing@sdu.edu.cn

Haixin Duan
Tsinghua University; Quancheng Laboratory
Beijing, China
duanhx@tsinghua.edu.cn

ABSTRACT

Mini-programs are lightweight apps running in super apps (such as WeChat, Baidu, Alipay, and TikTok), an emerging paradigm in the era of mobile computing. With the growing popularity of mini-programs, there is an increasing concern for their security and privacy. In essence, mini-programs are WebView-based apps. This means that they may be vulnerable to the same security risks associated with web apps. In this work, we discovered a new mini-program vulnerability called MiniCPRF (Cross-Page Request Forgery in Mini-Programs). The exploit of this vulnerability is easy, and the attack consequences are severe, leading to unauthorized operations, such as free shopping, and the exposure of confidential information, such as credit card numbers. The root causes of MiniCPRF can be attributed to multiple design flaws in both mini-programs and their super apps, including the insecure routing mechanism, lack of message integrity check, and plain-text storage. To evaluate the impacts of MiniCPRF, we designed an automated analysis framework called MiniCAT. It can automatically crawl mini-programs, perform static analysis on them, and generate detection reports. In large-scale real-world evaluations with MiniCAT, we identified that 32.0% (13,349/41,726) of analyzable mini-programs are potentially vulnerable to MiniCPRF, including some famous ones with millions of users, such as Sohu and Wenjuanxing. Following the responsible disclosure principle, we have reported verified vulnerable mini-programs to the corresponding vendors and developers, and three real-world cases have been confirmed by CNVD. Additionally, we suggest mitigation strategies to resolve the security issue related to MiniCPRF.

CCS CONCEPTS

• **Security and privacy → Software and application security**.

KEYWORDS

Mini-program Security; Program Analysis; Vulnerability Detection

ACM Reference Format:
Zidong Zhang, Qinsheng Hou, Lingyun Ying, Wenrui Diao, Yacong Gu, Rui Li, Shanqing Guo, and Haixin Duan. 2024. MiniCAT: Understanding and Detecting Cross-Page Request Forgery Vulnerabilities in Mini-Programs. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3670294

1 INTRODUCTION

Mini-programs, lightweight apps within a super or host app, have played an important role in mobile computing due to their convenience and functionality. The global popularity of mini-programs (hosted by WeChat [22], Baidu [6], Alipay [2], TikTok [26], and others) underscores the growing concerns about security and privacy.

Mini-programs have features of both mobile and web apps and operate within a super app. For example, page navigation and communications are managed via routing, similar to many web apps, and they can manage user states. This allows user-specific data to be stored and retrieved in future interactions. However, the routing mechanism in mini-programs can become susceptible to attacks,

*Corresponding authors.

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3670294

resulting in issues like broken authentication, request forging, and sensitive data leaks.

Existing research has predominantly assessed the security of mini-programs from the perspective of mobile apps, such as exploring permissions, common development bugs, and cross-mini-program communications [68, 70, 72]. Nevertheless, a noticeable gap persists in studying mini-program security from a web app standpoint.

Our Work. We have discovered a new vulnerability that results from the flawed design of page routing and user state management in mini-programs, including dependence on URL schema for navigation and transmission of parameters, unencrypted parameter communication, and inconsistent preservation of user state.

In particular, utilizing the sharing & forwarding features of mini-programs and the insecure local storage, an attacker can manipulate the URLs designated for page routing within mini-programs. This manipulation allows controlling the targeted routing page and the transmitted parameters. This specific vulnerability, termed ***MiniCPRF*** (Cross-Page Request Forgery in Mini-Programs), can result in the following consequences:

- *Consequence I:* Inducing victims to access modified mini-program page routes, thus executing sensitive operations (e.g., free shopping and unauthorized device control).
- *Consequence II:* Stealing sensitive information from the mini-program routes (e.g., the credit card number).

After further investigation into the root causes of MiniCPRF, we found they were linked to several design flaws in both mini-programs and their super apps, including plain-text routing parameter transmission, a lack of integrity check for chat messages, and modifiable local storage of the super app. Thus, due to the simplicity of implementing MiniCPRF and the potential for attackers to expand the attack surface by exploiting the sharing mechanism of mini-programs, it is imperative to measure the impact of MiniCPRF on the current mini-program ecosystem and their corresponding platforms comprehensively.

To conduct such a large-scale measurement, we designed an automated detection framework called MiniCAT (MiniCPRF Analysis Tool). It consists of a mini-program crawler and a MiniCPRF automatic detector based on reverse taint analysis. With MiniCAT, we crawled mini-programs on a large scale and detected the corresponding potential MiniCPRF issues. In the experiments, we collected 44,273 WeChat mini-programs, and 41,726 (94.2%) can be successfully unpacked for further analysis. The final results show that 32.0% (13,349/41,726) of them are risky to MiniCPRF, including some famous ones with millions of users, such as Sohu [21], a leading Chinese Internet and media mini-program, and Wenjuanxing, a leading online survey service mini-program. Moreover, we also propose measures to mitigate MiniCPRF, such as deploying encryption methods and performing message integrity checks.

Our research primarily concentrates on WeChat mini-programs due to their prevalence, as WeChat has over 900M daily active users worldwide [40]. Nevertheless, our findings on MiniCPRF in WeChat mini-programs led us to extend our analysis to other mini-program platforms, such as Baidu and Alipay, where we discovered similar vulnerabilities. This indicates that MiniCPRF is not an isolated issue but a widespread security concern across the mini-program ecosystem.

Responsible Disclosure. Every attack experiment described in this paper that involves real-world mini-programs was performed in a controlled environment. Moreover, we have reported all our discovered vulnerabilities to the corresponding vendors and developers. Currently, three of them have been confirmed and assigned vulnerability IDs: CNVD-2024-05527 (high-severity), CNVD-2023-75836 (moderate-severity), and CNVD-2023-75837 (moderate-severity).

Contributions. The main contributions of this paper are:

- *New Vulnerability.* We identified a new type of mini-program vulnerability named MiniCPRF. This vulnerability enables attackers to forge mini-program routing and parameters, leading to various security consequences, such as controlling sensitive operations and information leakage.
- *New Tool.* We developed an automated analysis framework called MiniCAT, specifically designed for MiniCPRF. This framework can automatically crawl WeChat mini-programs and perform MiniCPRF detection based on static analysis.
- *Large-scale Evaluations.* We measured 41,726 out of the 44,273 crawled WeChat mini-programs and found that 13,349 (32.0%) could be identified as potentially vulnerable to MiniCPRF.

Open Source. Our analysis framework, MiniCAT [49], has been released on GitHub.

Demo Site. The anonymized PoC attack demos of the above vulnerabilities can be found at https://sites.google.com/view/minicprf.

Roadmap. The rest of this paper is organized as follows. Section 2 provides the necessary background of the WeChat mini-program framework and introduces the threat model used in this paper. Section 3 discusses the motivation case and summarizes MiniCPRF. The detailed design of MiniCAT is illustrated in Section 4, and Section 5 gives its prototype implementation. Section 6 analyzes the evaluation results. Section 7 discusses the limitations of our work and the lessons learned. Section 8 reviews the existing related work, and Section 9 concludes this paper.

2 BACKGROUND AND THREAT MODEL

2.1 Mini-Program

WeChat Mini-Program Framework. WeChat is a killer app developed by Tencent for messaging, social media, and mobile payment. Besides, WeChat is also a super app that provides a runtime environment for WeChat mini-programs [35]. These mini-programs are lightweight without installation and provide users with various services, such as e-commerce, games, and tools. The architecture of a WeChat mini-program has two parts, as illustrated in Figure 1: 1) a front-end running on the super app to interact with the user and access system services; 2) a back-end providing the running environment (super apps) and performing server-side operations.

According to the official documentation [9], the mini-program front-end can be further divided into a render layer and a logic layer [30]. WXML (Wechat Markup Language) templates and WXSS

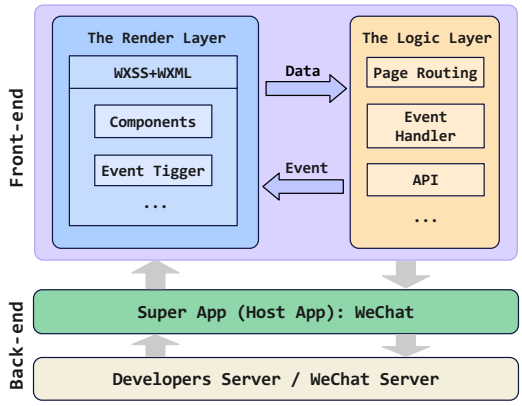

Figure 1: WeChat mini-program architecture.

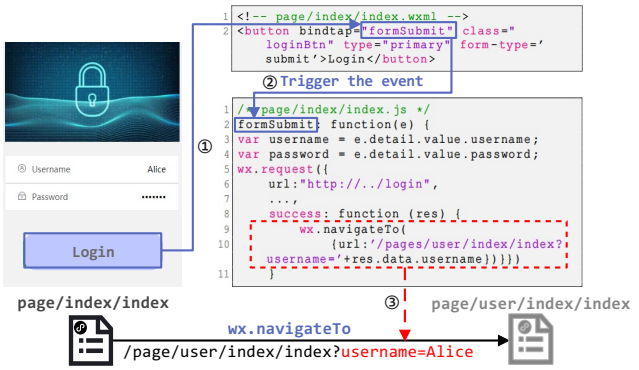

Figure 2: Example of event communication and routing.

(Wechat Style Sheets) are used in the render layer, while JavaScript is used in the logic layer.

Figure 1 shows how WeChat mini-programs enable communication between the render and logic layers using *Events* [13]. User actions are transferred from the render layer to the logic layer via events for further processing by the super app, similar to JavaScript DOM Events [15]. When an event is triggered by the render layer activity (e.g., user tap), the logic layer will execute the corresponding function named *Event-Handling Function*. Events drive the functionality and interactivity of the WeChat mini-program framework.

Routing Implementation. Routing [20] is a common concept in web apps, referring to determining the network scope for the end-to-end path when packets are transmitted from a source to a destination. Similarly, WeChat mini-programs require interactions and communications between different pages. In this paper, we define these processes as **mini-program page routing**. In detail, mini-program page routing is the rule for navigating from one page to another based on routing rules (i.e., the path) [37]. The URL schema controls it and can pass parameters in the page URL. For example, a completed mini-program routing path may look like `/page/index?param=1` (similar to the GET method in HTTP). WeChat officially provides three APIs for navigation between mini-program pages: `wx.navigateTo` [45], `wx.redirectTo` [46], and `wx.reLaunch` [47].

Figure 2 shows an example of event communication and page routing implementation. The `loginBtn` button component in `index.wxml` is bound to the event-handling function `formSubmit`. When the user enters the username and password and presses the login button (①), this action will trigger `bindtap` and then trigger `formBubmit` (②) of `index.js`, resulting in sending the user's username and password to the developer's server for verification. If the verification is successful, the username will be a routing parameter of `/page/user/index/index`, and the mini-program will jump to `index` page via `wx.navigateTo` (③).

User State. The user state refers to the user's authenticated status in a system or app, meaning that a user has successfully authenticated and can durable access the protected resources or functionalities. It serves as the foundation for maintaining user sessions and enforcing security measures throughout the user's interaction with the system. In HTTP-based web apps, cookies and session mechanisms typically manage the user state. However, WeChat mini-programs do not support such HTTP-like mechanisms. To implement user states in WeChat mini-programs, developers need to utilize WeChat's specific authentication mechanism based on the OAuth 2.0 model [31]. Figure 3 shows an example of mini-program authentication and user state process in the logic layer and the back-end. Specifically:

- **Step ❶**: The mini-program initiates `wx.login`, prompting the WeChat client to produce a temporary credential, generally 5-minute valid, named `code`.
- **Step ❷**: Developers need to setup a handler page `handler.php` in their own servers. The front-end of mini-programs sends the `code` to the handler page, and the page will communicate with the WeChat server by the back-end API `code2Session`, retrieving the user's unique `OpenID` and `session_key`. At this point, the `session_key-OpenID` forms a pair of credentials for the user state.

There are two options to check the user state on specific pages:

- **Step ❸**.A: Design a custom user state using the `OpenID`, stored locally [38] via `wx.setStorage`. When validation is needed, developers can use `wx.getStorage` to fetch the `OpenID`, as demonstrated in `Page I` of Figure 3.
- **Step ❸**.B: Use `wx.checkSession` for the validation of user state generated by `wx.login`, as seen in `Page II` of Figure 3.

It is crucial to mention that the custom user state method mandates verification on every authentication page. Otherwise, the page will lack user state by default.

Sharing and Forwarding. As WeChat is a social-focused app with social features, users can share mini-programs with their friends through `Moments` or chats [34]. The shared mini-programs will appear as ***WeChat mini-program cards*** as normal chat messages. Moreover, developers can customize the shared mini-program card and use the `onShareAppMessage` function of one specific page in the logic layer to determine whether this page can be shared or forwarded to other users.

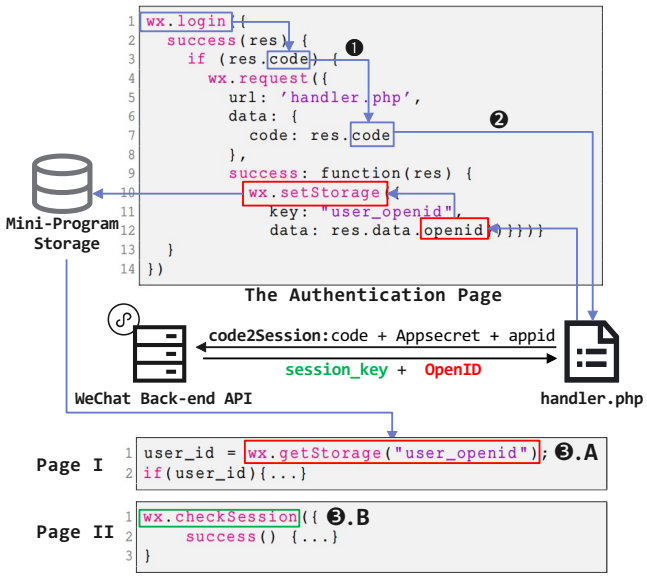

Figure 3: Case of authentication and user state management.

2.2 Threat Model

Here, we discuss the threat model used in our work, including the attack scenario and the attacker's capabilities.

Scenario. Both the attacker and the victim are WeChat users. They can access WeChat mini-programs using the official WeChat client on any platform (Windows/macOS/iOS/Android). Additionally, their devices run normally without malicious mini-programs. In this scenario, the attacker does not intercept communications directly between WeChat clients, nor is there a need for physical or remote manipulation of the victim's device. Moreover, the attack can also occur indirectly (e.g., through sharing mini-program cards).

Attacker's Capabilities. An attacker can exploit MiniCPRF vulnerabilities to generate malicious mini-program cards and either personally click on them or induce the victim to do so, enabling both individual and mass targeting. Furthermore, the attacker can create or modify malicious mini-program cards, with or without an existing card as a base. To construct or modify these cards, attackers only need to acquire the page routing URL and corresponding parameters, which can be obtained directly or indirectly from the victim. These capabilities are relatively easy for the attacker to achieve because WeChat mini-program cards are stored in the local storage, and a public method [12] is still functional in the latest version of WeChat (Jan 2024). Consequently, the attacker can access the detailed content of mini-program cards for the purpose of modification or forgery.

3 MOTIVATION CASE AND MINICPRF

3.1 Motivation Case

Here, we give a real-world case to illustrate MiniCPRF. In this case, an attacker can bypass the authorization of OKLOK (a mini-program for smart locks management) through MiniCPRF and take over the smart locks of the victim users.

Normal Workflow. The unlocking process of OKLOK is illustrated in Figure 4. OKLOK offers two binding options for users Bluetooth smart locks: QR code scanning (①.A) and a mini-program's device search page (①.B). In both methods, the mini-program communicates with the back-end server, submitting the QR code or receiving Bluetooth broadcast packets and retrieving a unique, user-invisible `_id` identifier (used to identify each lock). This `_id` guides the user to the Bind Page (`pages/index/deviceAdd?_id=...`), verifying whether the lock has been bound (②). If the lock is already bound, other users will receive a message indicating the need for permission to access it. If unbound, the current user's account will be linked to the lock, and the process advances to the Unlock Page (`/pages/index/deviceDetail?_id=...`), carrying the `_id` (③). The login status (user state) is checked on this page, and then the mini-program will start the unlocking process to unlock the target lock (④).

Vulnerability and Attack. However, the above process can present opportunities for attackers. Through manual analysis, we discovered that the `Bind Page` allows users to share, presenting as a WeChat mini-program card. As illustrated in Figure 4, a WeChat mini-program card is an XML text stored in the local storage of the WeChat client. It contains the page routing URL (i.e., `<pagepath>`) that the mini-program navigates to after a user clicks on this card. Also, OKLOK allows users to forward the `Bind Page` to any chat. Besides, we found that the `Unlock Page` does not implement complete user state verification, which means that this page only checks if the user is logged in, not if the user is the owner of the `_id` lock. In other words, any user who provides the correct `_id` can unlock the target lock.

The attack process is described as follows: First, the attacker can share the `Bind Page` to any chat (❶). Then, the attacker can extract this card from the local storage to obtain the lock's `_id` (❷). Next, they can modify the URL in the `<pagepath>` XML section of the card to the URL of the `Unlock Page` with the device's `_id` (`pages/index/deviceDetail?_id=...`), and update the modified card to the local storage (❸). Since the `Unlock Page` implements incomplete user state checks, the attacker can click on the modified mini-program card (❹) and then navigate to that page and perform unauthorized unlocking (❺). It should be noted that attackers can acquire the victim's `_id` directly (e.g., by accessing the victim's device) or indirectly. For example, in an Airbnb rental scenario, if the landlord (victim) remotely shares a temporary mini-program card with the tenant (attacker) for unlocking, it may accidentally expose the device's `_id`, enabling the attacker's permanent control of the device by exploiting MiniCPRF.

Even worse, the mini-program sharing feature enables attackers to widely distribute malicious mini-program cards, potentially affecting all OKLOK products. To carry out the attack mentioned above, the attacker only needs the device's `_id`. Besides, WeChat's functionality of navigating through mini-program QR codes (i.e., WeChat mini-program code [29]) allows attackers to enable unauthorized unlocking for any user who scans such a QR code. In other words, attackers can widely distribute forged mini-program

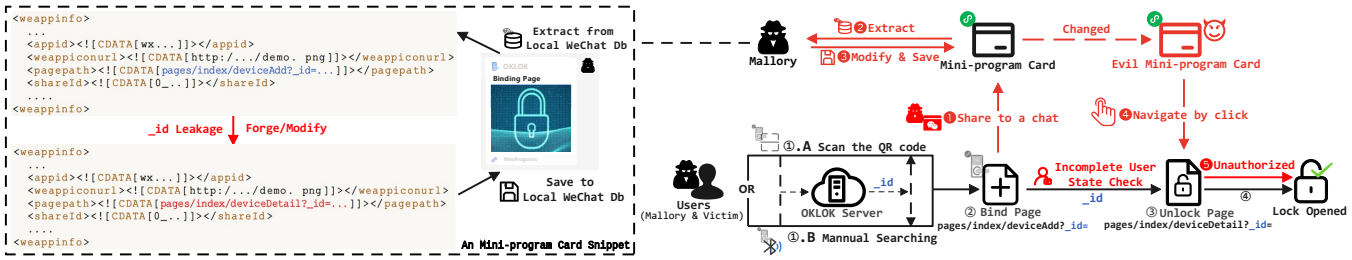

Figure 4: The workflow and attack path of the OKLOK case.

cards, affecting many users simultaneously in a "spray and pray" approach.

3.2 Summary of MiniCPRF

To summarize, a complete attack flow exploiting MiniCPRF can be generalized in three steps.

Step 1 Attackers identify potentially vulnerable pages and obtain parameters from the page's routing URL by forwarding (say ❶).

Step 2 The attacker modifies a mini-program card or generates a new one by modifying the page routing URL (say ❷ & ❸).

Step 3 When the victim (or the attacker in some exploiting cases) clicks on the crafted mini-program card, the attacker can perform sensitive operations (say ❹ & ❺).

Root Causes. We have identified three main factors that cause the appearance of MiniCPRF.

- The routing of WeChat mini-programs only allows developers to convey parameters in the URL schema. If the developer lacks security awareness, such as transmitting sensitive parameters in plaintext within page routing URLs, this can lead to information leakage and associated security risks.
- WeChat mini-programs lack a unified and overall user state implementation, with existing user state security depending on the developer's awareness. In the above case, the lack of implementation of user state verification on sensitive pages is a key factor for the attack's success.
- WeChat's lack of integrity checks enables the forwarding of modified and forged malicious mini-program cards, and these cards are stored as plain-text XML in the local storage, allowing attackers easy access to view and modify page routing parameters from those cards.

Consequences. In summary, MiniCPRF can result in two potential consequences.

- *Unauthorized operations.* If developers include sensitive operation-related parameters in the mini-program page routing URLs, attackers can manipulate or forge those URLs to carry out unauthorized operations. Additionally, through sharing or generating a mini-program code, these malicious mini-program cards can be used by others, creating a significant security threat.
- *Sensitive data leakage.* Sensitive data may be leaked during the attack. In the above case, the lock's `_id` should be invisible to the mini-program users. However, attackers can obtain `_id` through the MiniCPRF vulnerability. If the `_id` is used on other pages of the mini-program or involved in other sensitive operations, it can increase the attack surface, leaving it vulnerable to further exploitation.

Table 1: Comparison of CMRF, CSRF, and MiniCPRF.

MP: Mini-program; WA: Web app.

Vulnerablity	Target	Mechanism	Scope
CMRF	MP	Cross-MP	Two/Multi MPs
CSRF	WA	WA routing	Single WA
MiniCPRF	MP	MP routing	Single MP

Remarks. It is important to distinguish MiniCPRF from CMRF (Cross Miniapp Request Forgery) proposed by Yang et al. [70]. MiniCPRF targets security issues in routing within a *single* mini-program. On the contrary, CMRF concentrates on the risks in communication between *two* mini-programs. CMRF assesses if a target mini-program verifies the source called by another mini-program, similar to cross-app security concerns in mobile apps. Besides, it should be noted that MiniCPRF is a new type of vulnerability. Although its attack method (using URL modifications and forgeries) might look similar to CSRF, they have different targets for attack and underlying mechanisms. The differences among CMRF, CSRF, and MiniCPRF are summarized in Table 1.

3.3 Automated Detection of MiniCPRF

Although WeChat mini-program pages and web apps have similar routing mechanisms, applying existing studies on security issues like CSRF in web apps to mini-programs is difficult. Specifically, previous studies [55, 63] usually use hybrid static-dynamic analysis approaches to detect CSRF. However, WeChat has strict security measures to prevent instrumentation analysis on real-world mini-programs. As a result, we cannot obtain the current page's routing URL or the user state at run-time. Besides, since all back-end communication in the mini-program is invisible, we cannot access the communication context. Thus, existing methods are inapplicable for detecting MiniCPRF in mini-programs. There is a need for a new solution to address these challenges.

Fortunately, unlike web apps, mini-programs can be easily crawled and unpacked. With direct access to the source code, it becomes feasible to detect potential MiniCPRF vulnerabilities using static analysis. In detail, we need to design a system that can automatically acquire the source code of mini-programs and perform code analysis on both the logic layer (i.e., JavaScript files) and the render layer (i.e., WXML files). The system should accurately collect

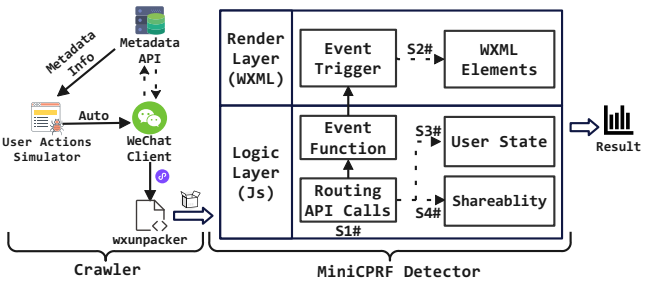

Figure 5: The Workflow of MiniCAT.

risky page routing URLs, associated parameters, and attack paths for MiniCPRF. This collected information can be used to create malicious mini-program cards to launch MiniCPRF attacks.

4 DESIGN OF MINICAT

We designed an automated analysis framework, MiniCAT, to detect potential MiniCPRF vulnerabilities in WeChat mini-programs. As shown in Figure 5, MiniCAT contains two modules: Mini-Program Crawler to collect massive mini-programs and MiniCPRF Detector to detect MiniCPRF vulnerabilities in mini-programs.

4.1 Mini-Program Crawler

Crawling WeChat mini-programs is challenging since there are no official or third-party markets similar to Google Play [14] or Apkpure [4] for Android apps. Typically, users access mini-programs by scanning mini-program QR codes or searching within the WeChat client. Zhang et al. [73] developed MiniCrawler to download mini-programs using specific WeChat APIs and designated AppIDs, which are unique identifiers for mini-programs. However, batch querying of AppIDs has become impossible due to Tencent's restriction on related API access.

During our investigation, we discovered that when a user accesses a mini-program on the WeChat Windows client, the client creates a directory under the user profile folder to store the mini-program, located at `user_file/Applet/AppID`. Furthermore, a WeChat metadata API [23] can provide metadata information for mini-programs, such as description, type, and developer information, based on the AppID. This discovery inspired us to develop an automated crawler that simulates user actions on the WeChat Windows client.

Our crawler, called Mini-Program Crawler, utilizes Natural Language Processing (NLP) techniques to construct a keyword dictionary to search for mini-programs within the WeChat Windows client, similar to the approach mentioned in MiniCrawler [73]. Additionally, Mini-Program Crawler leverages the mini-program metadata to generate the keyword dictionary. The metadata can be obtained through the metadata API using the mini-program AppID from the user profile directory.

Furthermore, the crawler retrieves mini-program packages from the user profile directory and unpacks them into source code using *wxappUnpacker* [42] for further analysis, serving as input for MiniCPRF Detector.

```
/* deviceAdd.wxml */
/* <button bindtap="__e" class="ubgc-blue umar-t100"
     data-event-opts="{{[ [ 'tap',[ ['toBindDevice'] ] ]
      ]}}">{{''+(lan.btn_bind||'bind')+''}}</button>
     */

/* deviceAdd.js */
④ Page({
...
onLoad: function(e) {
  var data = this ;...
  wx.setStorageSync ("user",data.user)
  ...
}
onShareAppMessage() {
    return {..}
},...
③ toBindDevice: function() {
 ....
  var a = {
    _id: t.form.code,
    user: t.user._id,
    remark: t.form.remark };
 /* binding the device from the server-side */
 n.default.httpPost({
    name: "device/bind",
     data: a,
    /* If the binding process is successful, the
     device's _id will be connected to the URL as a
     parameter */
    ② success: function(a) {
      var n = {
       _id: t.form.code
       };
    ① wx.redirectTo({url: "/pages/index/deviceDetail/
     deviceDetail?param=" + JSON.stringify(n) });
     },
     fail: function(e, t) {
         n.default.showToast(t);
     }
   });}...})

/* deviceDetail.js */
Page({
...
onLoad: function(t) {
  var o = this;
  o.app = wx.getStorageSync('user'),
  if(o.app){o.getDetail();}
},...
getDetail: function() {
  var t = this,
  o = {_id: t.param._id};
       ...
    success: function(o) { ...
      t.blue.device = o,
        /* Unlock the corresponding lock */
      t.toStart();
           ...});
},...})
```

Listing 1: Code snippet of OKLOK.

4.2 MiniCPRF Detector

Following the attack for MiniCPRF flow described in Section 3, we designed MiniCPRF Detector in the following steps.

Step I: Identifying the Nodes Calling Page Routing APIs. According to the WeChat developer documentation [37], page routing APIs of WeChat mini-programs are called in the logic layer of the mini-program (i.e., in JavaScript files). We focus on three routing APIs: `wx.navigateTo`, `wx.reLaunch`, and `wx.redirectTo`. In detail, MiniCPRF Detector constructs an Abstract Syntax Tree

(AST) of the source code in the mini-program logic layer (i.e., JavaScript code). Then, it filters the callee nodes that match the names of those APIs. Next, MiniCPRF Detector filters the callee node and extracts the `url` property to obtain the URL passed through the page routing API. The URL is then split by regular matching into the potentially vulnerable mini-program page, (for Steps III and IV) and parameters (for Step II). As in the case of OKLOK (see Listing 1), MiniCPRF Detector can locate the API call to `wx.redirectTo` (Line 30~31). It extracts the `url` property from this API as the page routing URL, including the vulnerable page `/pages/index/deviceDetail/deviceDetail` and its parameters `JSON.stringify(n)`.

Step II: Building the Attack Path. After obtaining the routing and parameters of a potentially vulnerable mini-program page, the next step to launch the MiniCPRF attack is to build an attack path. In other words, our goal is not to obtain exact parameter values, which may be server-generated and hard to extract statically. Rather, we focus on outlining and formulating the attack path, which are the steps an attacker must take at the render layer to activate the page routing API, reach the vulnerable page, and eventually extract its routing URL. As described in Listing 1, after retrieving the `URL`, our MiniCPRF Detector aims to identify how to initiate the page routing API for `url` attribute manipulation within the attack context. Through manual analysis, we have charted the reverse attack path to engage the routing API at the mini-programs logic layer as follows: ①`wx.redirectTo()` → ②`success: function(a)` → ③`toBindDevice: function()` → ④`Page({..})`.

However, the existing static analysis tools for mini-programs [57, 60, 65] are insufficient for automated analysis of mini-programs like OKLOK, as they fail to detect MiniCPRF issues. In detail, these tools use forward static analysis, focusing on client-side data flow, such as TaintMini [65], which constructs data flows from user inputs to sensitive APIs. However, our approach differs from this, as we aim to reconstruct the entire attack path instead of building a data flow through page routing APIs. Thus, we need to find a new method for detecting MiniCPRF.

Since the attack path always ends with a page-routing API callee node, reverse taint analysis can potentially automate the analysis of this path. To verify our assumption, we manually analyzed the attack path in the mini-program. In the related view page in the render layer (`deviceAdd.wxml`), we discovered that the event `toBindDevice` interacts with the logic layer through a WXML attribute in a button component. As shown in Listing 1, when the user clicks the (`bindtap`) button, it triggers the logic layer (`deviceAdd.js`) to execute the `toBindDevice` event-handling function. The `_id` is then passed to `toBindDevice` through the event channel and ultimately linked with `?param=` to construct the complete page routing URL.

To address this issue, we propose an analysis method based on reverse taint analysis. Specifically, we set the page routing API node as the sink node and aim to locate its event-handling function `toBindDevice` through reverse taint query (as the target source node). Following that, the MiniCPRF Detector attempts to identify the WXML elements and attributes that trigger the event on the corresponding logic layer page. Note that reverse taint analysis aids our analysis by accurately identifying components on the attack path, such as event-handling functions and WXML components

Algorithm 1 Finding the event-driven function in the logic layer.

```
1:  function findEventHandling(rawAST, routingAPI_calleeNode)
2:      sourceNodeList ← ∅
3:      sink = routingAPI_calleeNode
4:      sourceNodeList += reverseTaint(rawAST,sink)
5:      for source in sourceNodeList do
6:          sourceScope = source.getContainer().getScope()
7:          sinkScope = sink.getContainer().getScope()
8:          if sourceScope == sinkScope then
9:              PR_node = sourceScope.getpredNode()
10:             if PR_node.getContainer() instanceof Function then
11:                 continue
12:             else if PR_node.getContainer() instanceof TopLevel then
13:                 EV_FunctionNode = source
14:                 break
15:             end if
16:         end if
17:     end for
18:     return EV_FunctionNode
19: end function
```

that trigger events. Therefore, we must solve the following two challenges to automate the above manual process:

Challenge I: Locating the correct event-handling function in the logic layer. In Listing 1, we found that the attack path passed two functions: `success` and `toBindDevice`. If MiniCPRF Detector mistakenly identifies `success` as an event-handling function, it cannot find the corresponding WXML element on the render layer page, resulting in an interruption in the attack path. Therefore, it is crucial for our approach to identify the correct function accurately.

To address Challenge I, we analyzed the AST of the source code and identified that the part executing the page routing API (Lines 15~35 in Listing 1) acts as a function container. This container could either be an event-handling function (*EV function*) or another function type (*OT function*, e.g., the `success` callback function). Distinguishing between these functions is possible by examining the scope of their preceding node (*PR node*). Based on the standardized architecture of mini-program logic layers [24], the *PR node* of an *EV function* aligns with its module node at the AST's top level (the `Page()` object in Listing 1), while the *PR node* of an *OT function* is located within a function container.

In particular, MiniCPRF Detector identifies each function's *PR node* using reverse taint analysis, as shown in Algorithm 1. It defines the *PR node* scope per node, accurately locating the *EV function* while avoiding *OT function* confusion. If an *EV function* ties to mini-program page functions [36] like `onLoad` or `onHide`, it suggests user interaction response, such as page loads or refreshes. In summary, we can solve Challenge I by analyzing the scope of *PR nodes* in the AST, achieving accurate locating of event-handling functions without interference from other functions.

Challenge II: Identifying WXML components in the render layer. After solving Challenge I, we can locate the event-handling function on the logic layer. Then, we must implement a data flow analysis to find the event-triggered element in the render layer. Although tools like [57, 60, 65] do analyze WXML data flows, they fall short of our needs as their analysis initiates at the render layer.

Algorithm 2 Finding event trigger attribute in the render layer.

```
1: function findEventWXML(wxmlFile, jsFile, EV_FunctionNode)
2:     newSourceCode ← ∅
3:     newSourceCode += (convertoHTML(wxmlFile), jsFile)
4:     newAST ← ∅
5:     newAST += buildAST(newSourceCode)
6:     wxmlSinkNode = EV_FunctionNode
7:     wxmlSourceNode = reverseTaint(newAST, wxmlSinkNode)
8:     EV_attribute = wxmlSourceNode
9:     return EV_attribute
10: end function
```

This necessitates a technique for the MiniCPRF detector to analyze WXML files effectively and make the attack path uninterrupted.

As the syntax of WXML is similar to HTML [43], we can use a public tool [44] to convert WXML files to HTML files while maintaining the raw WXML tags and attributes. Then, utilizing well-established HTML-supported analysis tools (such as CodeQL [11]), we can reconstruct the AST with the transformed mini-program source code for further cross-layer analysis. As shown in Algorithm 2, MiniCPRF Detector uses the previously queried *EV function* as the new sink node of the new reverse taint analysis in the new AST. Finally, MiniCPRF Detector can locate the attribute corresponding to the *EV function* (called *EV attribute*). Since the converted WXML files preserve all original attributes, we can identify the EV attribute and its corresponding element as the target event-triggered WXML components.

In the OKLOK case (Listing 1), we can construct a complete attack path of the MiniCPRF vulnerability page after Step II. That is, to get the value of `t`, an attacker only needs to click the target `<button>` on the `deviceAdd` page (`deviceAdd.wxml`). After clicking, `t` will be passed into `toBindDevice` and used as part of the parameter `url` of `wx.redirectTo` in `deviceAdd.js`. Then, the OKLOK mini-program will jump to the vulnerable page `deviceDetail`. If page `deviceDetail` can be shared or forwarded, the attacker can easily obtain the value of `t` by extracting its routing URL.

Step III: Checking User State. In the OKLOK case, a crucial factor for the success of the attack was the lack of correct user state implementation on the `deviceDetail` page, allowing attackers' unauthorized access and sensitive operations. As mentioned in Section 2, the implementation of user state in WeChat mini-programs is more unified than that of web apps, enabling us to perform general static analysis on them.

Our static analysis focuses on detecting two types of API calls in the `onLoad` page load functions of potentially vulnerable pages. The absence of these calls suggests that the page does not check the user state upon loading, leading to higher risks, especially if MiniCPRF vulnerabilities exist.

(1) The APIs can check if the user state obtained by `wx.login` is expired, such as `wx.checkSession`.
(2) The APIs can get the local mini-program storage cache, such as `wx.getStorage` and `wx.getStorageSync`.

In the OKLOK case, after the user logs into the mini-program, `wx.setStorageSync("user")` on the logic layer is used to store the user field in local storage. However, when users navigate to the `deviceDetail` page and trigger the `onLoad` function, it lacks complete verification of the user state. That is, the page logic can be executed regardless of whether the user is logged in. This inconsistency in user state is key to the success of MiniCPRF attacks.

Step IV: Shareability Check. As mentioned in Section 2, the `onShareAppMessage()` function can control the shareability of the mini-program page. MiniCPRF Detector analyzes the page with page routing API calls in Step I, checking whether it uses this function to determine its shareability. As shown in Listing 1, MiniCPRF Detector identifies that the page `deviceAdd.js` calls the `onShareAppMessage()` function in the OKLOK case. The attacker can share the page within any chat and extract the full page routing URL from the WeChat local storage, which simplifies the attack.

5 IMPLEMENTATION OF MINICAT

Here, we introduce the implementation of MiniCAT, including the Mini-Program Crawler and MiniCPRF Detector modules. In summary, MiniCAT contains 2,374 lines of Python code and 900 lines of QL code.

Mini-Program Crawler Implementation. First, we used `pywinauto` [19] (a set of Python modules to automate the Windows GUI) to simulate the user search and access process on the WeChat Windows client. Moreover, using the NLP word segmentation library, `jieba` [16], we processed mini-program descriptions from the metadata API mentioned in Section 4 to create the search keyword list. Finally, we used *wxappUnpacker* (a public WeChat mini-program unpacking tool) to unpack the crawled mini-program packages.

MiniCPRF Detector Implementation. After obtaining source codes of mini-programs, we implemented MiniCPRF Detector following the steps designed in Section 4 based on CodeQL, a static analysis tool that can build ASTs and conduct static data flow analysis. Furthermore, we used the `wxml-transformer` to convert WXML to HTML files.

6 EVALUATIONS

In this section, we discuss the detection results of MiniCAT. We also design a passive DNS-based approach to evaluate the popularity of mini-programs. Moreover, two real-world cases are analyzed deeply, and measurements on multi-platforms are discussed.

Experiment Setup. Our Mini-Program Crawler was deployed on a Windows 10 laptop (i7-6600u/16 GB RAM) with Python 3.8.0. For MiniCPRF Detector, we performed the analysis on a server running Ubuntu 20.04 with 32 CPU cores and 256 GB memory, utilizing 10 threads to analyze all mini-programs.

Dataset. We collected and successfully unpacked 44,273 WeChat mini-programs using Mini-Program Crawler, which occupied a storage space of 126.38 GB. These mini-programs consisted of 2,264,377 pages, with an average of approximately 51 pages per mini-program. For those non-unpackable mini-programs, we found that they were incompatible with the *wxappUnpacker* tool due to their use of a newer version of the WeChat mini-program base library. Additionally, failures in unpacking were also attributed to missing main packages [17].

Result Overview. Among the mini-programs collected, MiniCAT successfully analyzed 41,726/44,273 (94.2%), identifying 13,349/41,726

(32.0%) as potentially vulnerable with a cumulative 119,471 risky pages. On average, each such mini-program contained nine potential MiniCPRF vulnerabilities. The remaining 2,547/44,273 (5.8%) mini-programs were not analyzed due to absent codes from WeChat Cloud Development [28] or highly obfuscated source codes.

Efficiency. Mini-Program Crawler took about 15 seconds to crawl a single mini-program, enabling an average collection of 2,687 mini-programs daily. Finally, our crawler collected a dataset of 44,273 mini-programs, proving the capability to gather WeChat mini-programs.

For MiniCPRF Detector, the total analysis time was eight days, averaging 106.75 seconds for each mini-program. Building each mini-program AST and performing a global data flow analysis in a CodeQL query was time-consuming. To optimize the process, MiniCPRF Detector introduced a 5-minute timeout for CodeQL queries, consistent with the default timeout of CodeQL CLI [10] and another previous JavaScript static analysis work using CodeQL [61]. As a result, 14,920 out of 41,726 (35.8%) mini-programs were skipped due to timeouts. Our further analysis revealed that these mini-programs featured heavily obfuscated source codes, leading to timeouts due to endless CodeQL iterations.

6.1 False Positive and False Negative

As described in Section 4, MiniCAT generates potential MiniCPRF attack paths by scanning the source code of mini-programs. In other words, incorrectly identifying secure mini-program page routes (such as pages that cannot be shared) as potential attack paths will result in false positives (FP), as attackers are unable to exploit these routes for constructing MiniCPRF attacks. Similarly, note that if essential elements on the attack paths (such as page routes and their parameters, trigger pages, WXML elements, and page shareability) are not properly identified, MiniCAT may overlook potential MiniCPRF vulnerabilities, leading to false negatives (FN).

Due to the absence of ground truth, we followed the methodology outlined in previous studies [65, 70] to select 100 samples from mini-programs flagged as potentially vulnerable. These samples were manually inspected to determine whether they could construct MiniCPRF attack paths, thus measuring FP. Additionally, we manually selected 100 samples from mini-programs excluded from the MiniCAT query results. Then, we manually examined them to identify whether insecure page routing implementations (i.e., potential MiniCPRF attack paths) existed, thus measuring FN.

False Positives. Among the selected 100 potentially vulnerable samples of mini-programs, we did not detect any FPs. Moreover, we found that MiniCAT strictly analyzes source code models derived from MiniCPRF attack processes, following the guidelines set in the WeChat official developer documentation. This approach allows attackers to potentially use MiniCAT's detection results to forge mini-program cards for MiniCPRF attacks, ensuring MiniCAT does not produce false positives.

False Negatives. Among the selected 100 mini-programs not identified as potentially vulnerable by MiniCAT, we discovered 3 FN cases. These three FN cases correspond to different scenarios: 1) MiniCAT identified a function that invokes the page routing API as being embedded within another function registered in `Page()`, rather than as an independent event-handler function. 2) MiniCAT failed to capture the full page URL due to its construction from concatenated paths and strings. 3) The page's use of `getApp()` [48] to fetch the mini-program instance before defining `Page()` altered its position in the AST, leading MiniCAT to mistakenly exclude it based on Algorithm 1.

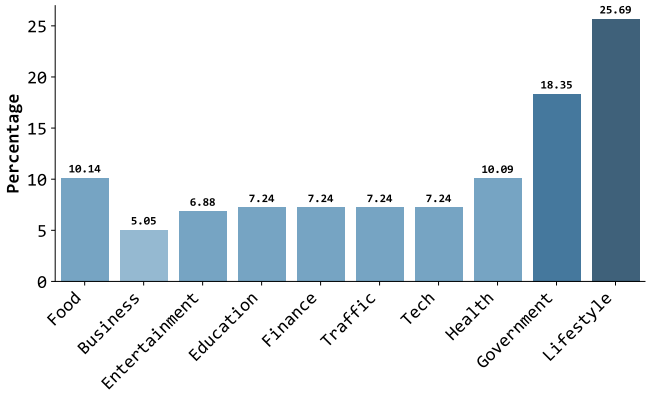

Figure 6: The MiniCPRF statistic based on categories.

In detail, Scenario 1 represents a complex challenge in the domain of static analysis tools (such as MiniCAT) due to the dynamic nature of function calls and event handling in JavaScript. Thus, it can lead to unpredictable outcomes that hinder systematic detection. Scenario 2 and Scenario 3 involved deviations from standard practices as recommended in the mini-program development documentation. We regard the latter two scenarios as instances where developers deviated from the guidelines outlined in the mini-program development documentation.

Note that the definition of our FP and FN is based on the static query results provided by MiniCAT, rather than the actual harm caused by the vulnerabilities detected during scanning. We will discuss the latter in Section 6.3.

6.2 Insight Evaluation

To evaluate the impact of MiniCPRF on mini-programs, we performed a detailed analysis of the evaluation results from multiple dimensions.

Category Analysis. We measured the potential impact of MiniCPRF on mini-programs based on their categories, as shown in Figure 6. Among all the mini-program categories affected by MiniCPRF, `LifeStyle` has the highest percentage (25.7%). What surprises us is that the `Government` category (18.4%) and the `Finance` category (7.2%) have relatively high proportions. Developers should ensure the security of these mini-programs since they often require frequent interactions with sensitive personal information, such as ID cards and credit card numbers.

Template Development. After analyzing results from MiniCAT, we identified 36 potentially vulnerable mini-program templates with similar risk-related page routes, event-function names, and WXML components. These templates, often used in similar functionalities such as shopping and dining categories, are sourced from third-party providers or WeChat itself. 503/13,349 (3.8%) matched

these templates. For instance, we discovered two completely unrelated mini-programs with similar structures. MiniCAT identified a camping reservation app and a coffee shop ordering app that both concatenated the price as a parameter in the page route on their order submission pages (`pages/order/order?action=selectOrder&showall=1&is_chooese_tick=..&price=..`), allowing attackers to modify the price and successfully invoke WeChat payments. Nevertheless, whether the vulnerabilities stem directly from the templates or how developers implement them remains unclear. Our finding suggests caution when using templates in development as there could be associated security risks.

Inconsistent User States. To discern whether developers inadvertently introduced MiniCPRF vulnerabilities, we analyzed all 119,471 risky pages. Of these, 13,109 (11.0%) involved user-state checks on their target pages, while 106,362 (89.0%) did not. In those 106,362 pages, except for only 3 (0.2%) potentially vulnerable mini-programs excluded user states across all pages, 6,792 (53.9%) of the potentially vulnerable mini-programs lacked user-state involvement on target pages. As mentioned in Section 4, inconsistent user state implementations are security hazards. Therefore, our analysis found that the current user state implementation of the WeChat mini-program framework may inadvertently cause developers to overlook user state verification on certain pages.

Impact Measurement by pDNS. To assess the impact of MiniCPRF, we aimed to measure the popularity of potentially vulnerable mini-programs. Although WeChat has no official mini-program ranking, previous studies [70, 73] used ratings to assess their popularity and impact. However, since the rating is optional for users, it may not accurately represent the user base. After investigation, we found that each WeChat mini-program had a domain allowlist [32], which restricts communication to specific domains. MiniCAT can crawl this list via the mini-program metadata API [23]. Therefore, we leveraged passive DNS (pDNS) [18] data related to these domains to indicate mini-program popularity.

Specifically, pDNS collects and examines historical DNS data, providing insights into domain resolutions. By tracking the pDNS records of the domain allowlist over time, we can infer the popularity of the mini-program. Utilizing a commercial tool with extensive pDNS records, we filtered out common service domains (e.g., Content Delivery Network (CDN) and Object Storage Service (OSS)) and specific duplicated domains (e.g., official WeChat/Tencent and third-party APIs). We focused on the unique domains of each mini-program. Our study targets 3,208 mini-programs with 9,007 related domains and measures their pDNS records in seven days (4th June to 11th June 2023). The top 10 mini-programs by pDNS record counts are detailed in Table 2. The result shows many potentially vulnerable mini-programs are very popular. Although pDNS does not give an exact user count for each mini-program, it can serve as a relative measure of popularity, offering insights into the impact of potentially vulnerable mini-programs.

6.3 Vulnerability Verification

As mentioned above, MiniCAT identifies potential vulnerabilities in mini-programs. However, verifying a MiniCPRF vulnerability is challenging due to the absence of ground truth about the server-side processes of these mini-programs, for example, with the route

Table 2: Mini-programs with top 10 pDNS record counts.

#Vul: Number of MiniCPRF potential vulnerabilities;
#pDNS: Number of pDNS records.

Mini-Program Name	Category	#Vul	#pDNS
LingLingFa	Job	9	226,572,893
Changan E-parking	Traffic	6	11,271,812
Sohu	Lifestyle	28	11,191,188
OPPO Shop	Shopping	2	10,489,618
BianLiFeng	Food	8	10,364,160
LengshuiJiang Parking	Traffic	11	9,311,400
Xiwo E-learning	E-learning	11	8,717,258
WMS	Business	4	8,019,688
MOGO	Traffic	9	7,474,291
CMobile CloudVision	Entertainment	20	4,984,846

`wx.navigateTo("/pages/pay/submit?orderid=...&price=")`, we can find that it redirects users to a payment page, showing the payment price and order id. However, it is uncertain whether modifying the `price` leads to a real vulnerability due to the lack of additional server-side authentication mechanisms for the mini-program.

Thus, for MiniCPRF, our verification focuses on the front-end of mini-programs. Specifically, we determine a mini-program as vulnerable if: 1) Parameters within the page routing URL can be obtained by manually sharing its card. 2) These parameters can be modified. 3) Users can open and be redirected from the modified mini-program card.

Building the Test-case Dataset. Since manually analyzing all 13,349 potentially vulnerable mini-programs is impractical, we thus constructed a potentially vulnerable mini-program set for manual vulnerability verification and evaluating the effectiveness of MiniCAT. We implemented the method based on an existing study [70]. In detail, we extracted parameter names linked to potential vulnerabilities from the results of MiniCAT. Then, we classified these parameters into five categories:

- `Payment Info`: Covers payment details, such as method and outcome. If breached, attackers can shop without payment.
- `Promotion Info`: Consists of membership cards, coupons, and promotions. Vulnerabilities allow attackers to claim undue benefits.
- `Order Info`: Information like order IDs and total prices. Attackers could tamper with prices and item counts, leading to some severe consequences.
- `Device Info`: Pertains to device data, especially from IoT devices, like device ID and UUID. Compromised info risks unauthorized access or leaks.
- `Personal Info`: Includes mobile numbers, addresses, and credit card numbers, leading to sensitive personal data leakages of users.

We prioritized these categories because their modifications carry serious consequences.Table 3 displays the categorized results. Note that a mini-program can fall under multiple categories due to various insecure routing implementations.

Table 3: Categorized results of parameters.

#IVM: Number of potentially vulnerable mini-programs involved.

Category	Examples	#IVM
Payment	payprice, isPaysuccess	2,432 (18.2%)
Promotion	coupon_id, membercard	1,903 (14.3%)
Order	orderId, goods_id	3,385 (25.4%)
Device	deviceID, device_uuid	5,805 (43.4%)
Personal	mobile, creditcardNum	4,642 (34.8%)

Verification Overview. To build our test case for verification, we randomly selected 80 mini-programs from each category, totaling 400 mini-programs. We detected severe security issues in 316/400 (79.0%) mini-programs, such as data leaks. Besides, we found that 33/400 (8.3%) mini-programs were unverifiable due to deprecation and inaccessibility, e.g., we could not verify some internal use mini-programs without specific credentials. Among the 316 mini-programs with vulnerabilities, 234/316 (74.1%) mini-programs have at least sensitive information leakage (e.g., leakage of phone number or address, etc.), and 82/316 (25.9%) mini-programs can conduct overstepping sensitive operations (e.g., viewing or modifying other users' information without permissions, etc.)

Harmless Mini-program with MiniCPRF. After excluding 33 mini-programs that cannot be analyzed, we found that 38/367 (10.4%) mini-programs cannot modify the parameters of their page routing URLs, or we failed to open their modified mini-program cards. We can categorize these 38 cases into three scenarios: 1) Operations implemented through MiniCPRF remain within the normal user role (12/38, 31.6%), such as a news app where users read news controlled by the id in `page/news/id=`. Even if the `id` is modifiable, it does not allow any sensitive operations. 2) Page routes contain one-time use or tamper-proof parameters (12/38, 31.6%). For example, in cases like `/pages/pay/submit?money=..&user=..&sign=...&timestamp=...`, the parameters sign and timestamp have time constraints, preventing attackers from altering them to bypass authentication. 3) Developers implemented secondary verification on the server side (14/38, 36.8%). For example, in cases where the payment amount is passed in the page route, the server-side re-verification of the payment amount ensures that even if modified, the payment cannot be processed successfully.

6.4 Real-world Case Studies

Based on our evaluation, we introduce some representative MiniCPRF instances to show the impact of MiniCPRF on real-world mini-programs, such as unauthorized control of smart devices and arbitrarily modified shopping prices. These cases demonstrate that MiniCPRF can cause significant security threats even as a front-end vulnerability.

Case I: Unauthorized Operations. DBDLOCK is a smart device management mini-program that allows device owners to authorize other users by adding them as friends, considering the need for multiple people to use a single smart device. Specifically, the `friendDetail` page receives a `user_id` parameter and redirects to the `friendAuth` page. The `user_id` uniquely identifies a mini-program user. Then, the `friendAuth` page adds the user associated

```
/* The trigger page: friendDetail.js */
tofriendAuth: function() {
    var t = {
        user_id: this.detail.friend._id
    };
...,
     success: function(t) {
        /* The target callee node*/
        wx.navigateTo({
            url: "/pages/friend/\texttt{friendAuth}/
     friendAuth?param=" + JSON.stringify(t)}
        });},...

/* Shareability implementation: friendAuth.js */
onShareAppMessage: function () {...}, ...
/* The user state check in the target page */
onLoad: function(e) {
    var n = this;...
    n.user = wx.getStorageSync("login") || {},
    ...
    n.initView();
}
```

Listing 2: Friends authorization in DBDLOCK.

with this `user_id` as a friend of the current user and grants them device permissions. Although `user_id` is transparent to the mini-program and users, attackers can obtain their own `user_id` by registering two accounts and replicating the process due to the shareability of the `friendAuth` page. Subsequently, attackers can generate malicious mini-program cards or malicious WeChat mini-program codes to redirect to `pages/friend/friendAuth/friendAuth?param={user_id of Attackers}`. If victims click on these malicious cards or scan the generated codes, they inadvertently add the attacker as a friend, thereby granting the attacker unauthorized control over their devices.

Case II: Modify Shopping Price. Due to the convenient integration of payment API [33] with mini-programs on WeChat, more and more mini-programs are used for online businesses. In this case, developers must implement secure payment mechanisms and hide sensitive information while jumping to other mini-program pages. Otherwise, attackers can perform sensitive payment operations by exploiting the MiniCPRF vulnerability (e.g., shopping for free).

EasyOrderHelper is a mini-program designed to help vendors sell products online and receive payments. Vendors can share the mini-program with the customers to sell their products online. As shown in Listing 3, when a user submits an order, the mini-program will navigate to the order submission page `submitOrder`, carrying the total number of current user's products in the shopping cart as `cartNum`, and the total price of the shopping cart as `cartPrice`.

However, the payment logic violates the security rules of WeChat Payment [39], which requires the payment order generation process to be invisible to the front-end of the mini-program. Plus, the server side of this mini-program lacks secondary authentication of the payment price. As a result, attackers can forward the `submitOrder` page to generate a mini-program card and modify the `cartPrice` parameter to a lower price, even zero. After clicking the card again, attackers can generate an order and purchase the goods at the modified price. More seriously, if the attacker forwards those malicious mini-program cards to other users, these cards might be widely spread, leading to huge financial losses for the merchant.

```
/* The trigger page */
cart_count: function(t) {
    ...this.setData({
        cartCount: h.data.cartNum,
        cartMoney: parseFloat(a).toFixed(2),...
        });
 },...
to_submit_order: function() {
    this.haveAvatar ? wx.navigateTo({
    url: "/pages/submitOrder/submitOrder?cartNum=" +
     this.data.cartCount + "&cartPrice=" +
     this.data.cartMoney
    }) : this.setData({
            authShow: !0
        });},...

/* pages/submitOrder/submitOrder.js */
onLoad: function(...){
        ...
       this.setData({
                ...
            cartNum: a.cartNum,
            cartPrice: a.cartPrice,
 ... })),...
```

Listing 3: Shopping for free in EasyOrderHelper.

Table 4: Feature comparison of mini-program platforms.

RI: Page Routing Implementation; **USI**: User State Implementation; **US**: URL Schema; **PwU**: Parameter with URL; **ENC**: Encryption; **CF**: Cookie-like Features; **CI**: Custom Implementation.

Platforms	RI			USI		Daily
	US	PwU	ENC	CF	CI	Active User
WeChat	✓	✓	×	×	✓	**928M**
WeCom	✓	✓	×	×	✓	**130M**
Baidu	✓	✓	×	✓	✓	**378M**
Alipay	✓	✓	×	×	✓	**639M**
TikTok	✓	✓	×	✓	✓	**276M**

✓: Implementation found; ×: Implementation not found.

6.5 Measurements on Other Platforms

To further evaluate the all-sided impact of MiniCPRF on the mini-program ecosystem, we analyzed other mini-program platforms besides WeChat. This analysis is divided into two main aspects: *Similarity Mechanism* and *Feasibility of MiniCPRF*.

Similarity Mechanism. Table 4 highlights the page routing and user-state mechanisms' resemblance to WeChat's, indicating potential shared vulnerabilities due to similar routing structures. Our research examined mini-programs from the WeCom [41], Baidu, Alipay, and TikTok platforms. In our test set, we found mini-programs with the same names across these platforms: 49 in WeCom, 49 in Baidu, 23 in Alipay, and 17 in TikTok. Further analysis showed a high degree of similarity in the vulnerability patterns of these mini-programs compared to their WeChat counterparts: 49/49 (100%) in WeCom, 47/49 (95.9%) in Baidu, 20/23 (87.0%) in Alipay, and 12/17 (71.5%) in TikTok. For mini-programs from platforms other than WeChat that produce inconsistent results, we noted they use distinct routing prefixes, suggesting a unified vulnerability classification. Detailed cases of these vulnerable mini-programs across platforms are listed in the public GitHub repository of MiniCAT [49].

Listing 4 illustrates the similarity in parameters and target pages of mini-programs named Wenjuanxing across different platforms despite minor differences in routing prefixes. This uniformity in routing mechanisms means MiniCAT needs only minimal adjustments for each platform. Thus, due to the similarity of the routing mechanisms, MiniCAT only needs to make some simple adjustments to apply to the analysis of the corresponding platform. For example, to analyze Baidu mini-programs, MiniCAT only needs to adjust the API names involved in each step. The MiniCPRF Detector from MiniCAT now supports automated WeCom, Baidu, Alipay, and TikTok mini-programs analysis.

Feasibility of MiniCPRF. As previously discussed, to perform the MiniCPRF attack, the attackers should easily access, view, and modify the page routing URLs and their parameters. If we want to figure out how to implement MiniCMRF on other platforms, we need to answer the following two questions:

- *Can attackers obtain mini-programs page routing URLs and parameters through sharing or forwarding?* Our analysis indicates that only the WeCom and Alipay mini-programs are vulnerable to MiniCPRF attacks. WeCom's mini-programs can be forwarded to WeChat, allowing the transition of MiniCPRF attacks due to similar frameworks. Baidu's mini-programs only permit forwarding via encrypted links and always redirecting to the homepage. Alipay's mini-programs enable forwarding to various applications, with forwarding to DingTalk exposing the path and parameters [3]. On the contrary, TikTok mini-programs lack sharing or forwarding capabilities.
- *Can attackers construct or modify mini-programs page routing URLs and parameters?* Our findings indicate that all examined platforms provide navigation to specific mini-program page paths. Notably, WeCom accepts mini-program cards from WeChat, and due to framework similarities, MiniCPRF attacks can be transferred from WeChat to WeCom. From platforms' developer documents [1, 5, 25] and W3C specifications [27], Baidu, Alipay, and TikTok mini-programs facilitate mini-program launching through the URL schema from H5 or Webview, allowing page and parameter customization. This feature opens avenues for MiniCPRF attacks, where attackers can craft a modified URL schema. Furthermore, even without routing paths and parameters (notably in Baidu and TikTok), attackers can exploit mini-programs with the same name on other platforms and then perform attacks.

In summary, despite certain limitations, our measurements indicate that MiniCPRF is a universal risk in the mini-program ecosystem, requiring great attention from vendors and developers.

7 DISCUSSIONS

Limitations. Although MiniCAT can perform further analysis and evaluations for MiniCPRF, our work still has some limitations. First, MiniCAT is a static analysis tool based on CodeQL, and its analysis algorithm is designed based on the standard code structure of WeChat mini-programs. Thus, like other static analysis methods, MiniCAT is hard to analyze mini-programs that cannot be unpacked or heavily obfuscated. Meanwhile, MiniCAT failed to analyze considerable mini-programs due to timeouts, which can

```
/* WeChat Mini-program */
goToPage: function(e) {
    var t = e.currentTarget.dataset.id;
    wx.navigateTo({
        url: "/pages/wjxqPage/wjxqPage?activityId="+t
    });},...

/* Baidu Mini-program */
goToPage:function(e){
    var t =e.currentTarget.dataset.id;
    swan.navigateTo({
        url:"/pages/baiduAppPages/wjxqPage/wjxqPage?
     activityId="+t
    })},...

/* Alipay Mini-program */
goToPage: function(e) {
    var t = e.currentTarget.dataset.id;
    my.navigateTo({
        url: "/pages/wjxqPage/wjxqPage?activityId="+t
    })
},...
```

Listing 4: Source code of Wenjuanxing in multiple platforms.

be partially alleviated by increasing the timeout value. Second, our crawler cannot successfully collect some mini-programs due to the compatibility limitations of the WeChat Windows client. Finally, MiniCAT mainly focuses on detecting potentially vulnerable mini-programs and cannot automatically verify the vulnerabilities identified.

Lessons Learned and Mitigation. As mentioned by Wang et al. [69], current mini-program frameworks have some missing functionality and security features compared to browsers running traditional web apps. For mini-program routing, we found that the routing implementation of most mini-program platforms uses plaintext URL schema and only allows parameters to be passed via URLs, leading to the risk of MiniCPRF. For user states in mini-programs, as shown in Table 4, we can observe that, except for Baidu and TikTok, other mini-program platforms do not support the cookie-like mechanism. If developers want to implement a reliable user state, they have to customize their implementation by utilizing the storage capabilities provided by the mini-program framework page by page. This process heavily relies on developers' security awareness and poses a significant challenge to the security of user states in mini-programs.

To address the issue of MiniCPRF, we recommend taking the following measures:

- Developers should consistently implement and verify the user state across pages and avoid transmitting sensitive parameters through page routing APIs. Parameters should be fortified with non-easily forgery elements like time-sensitive signs and timestamps. We also advocate conducting sensitive data processing on the server-side, but not on the front-end of mini-programs.
- Frameworks or super app vendors should address the root causes of MiniCPRF. They might consider supplying more secure parameter transmission methods in their page routing APIs and encrypting routing details instead of plain text. For mini-program cards, super apps could integrate server-side, hard-to-forge, time-sensitive signatures, preventing malicious tampering. Vendors may need to modify their frameworks to implement these protections. Alternatively, they can provide warning information on their development documentation.

Ethical Considerations and Disclosure. To ensure ethical research practices, we conducted proof-of-concept attacks only on our accounts, devices, and mini-programs, focusing on potential vulnerabilities in the front-end without exploiting the server-side and collecting sensitive data. When crawling mini-programs or using the metadata API, we set a reasonable rate limit (i.e., 20 requests per minute) to prevent server disruptions.

Meanwhile, we have tried our best to disclose our findings to related stakeholders and be responsible for the community. Just as Yang et al. mentioned [70], similar to CMRF attacks, vulnerabilities from our test-case set were first reported to Tencent, which attributed MiniCPRF issues to third-party developer lapses. Nonetheless, to raise awareness among mini-program developers, we acquired contact details for all 248/316 (78.5%) mini-programs, which provided contact information and informed them via email, providing technical details and remedial advice (the letter is in MiniCAT public GitHub repository [49]). We have also contacted CNCERT/CC [8], the Chinese vulnerability coordination organization, and disclosed our findings with CNVD [7]. As of the submission of this paper, three cases have been confirmed (CNVD-2023-75836, CNVD-2023-75837, and CNVD-2024-05527), with the disclosure process ongoing.

8 RELATED WORK

Mini-Program Security. To comprehensively understand mini-programs, Zhang et al. [73] presented MiniCrawler to crawl WeChat mini-programs and conducted a measurement study of them, including their resource consumption, API usage, obfuscation rate, etc. Lu et al. [59] studied resource management in app-in-app systems and discovered a series of security flaws that can cause system resource exposure and deception attacks. Wang et al. [68] proposed WeDetector to detect three WeChat mini-program bug patterns and discovered 11 previously unknown bugs in 25 mini-programs. Zhang et al. [72] studied the identity confusion vulnerability in app-in-app ecosystems, analyzed 47 super apps and found that all of them are vulnerable to identity confusion attacks. Yang et al. [70] proposed CMRFScanner to detect CMRF, a new type of vulnerability, on a large scale. The results show that 50,281 WeChat mini-programs and 493 Baidu mini-programs are subject to CMRF.

Most recently, Zhang et al. [74] and Baskaran et al. [50] examined the sensitive resource access protocols of mini-programs and discovered master key leakage vulnerabilities. Wang et al. developed TaintMini [65] to track sensitive data flow in mini-programs, finding 11.4% of the 238,886 evaluated ones had such data flows, with 455 risking privacy leaks through collusion. They also developed APIDIFF [66] to detect API execution differences in WeChat across platforms, categorizing discrepancies into API existence, permission, and output. Li et al. [57] developed MiniTracker to automatically track sensitive flows in mini-programs, addressing challenges such as asynchronous executions in JavaScript. Their study on 150K mini-programs revealed common privacy threats and data leak patterns. Yu et al. [71] developed MiniTaintDev from WeChat Dev Tools to perform dynamic taint analysis to detect data leakage and sensitive APIs invokes in mini-programs. Wang

et al. [69] explored the web technologies underlying super apps, detailing their security mechanisms and proposing guidelines for enhanced security from a browser's perspective. Tao et al. [64] introduced JSLibD, a tool for automated extraction and heuristic detection of third-party libraries in mini-programs. Wang et al. [67] proposed a framework for data minimization in mini-programs, focusing on usage scenarios to enhance privacy without compromising functionality. Long et al. [58] investigated dark UI patterns in mobile apps, highlighting the need for increased awareness and regulation. Cai et al. [51] addressed the complexity of the shared account issue in super apps, exploring its implications for security and user privacy. Zhang et al. [75] discovered and measured the TrustedDomain attack in app-in-app ecosystems, a method exploiting domain-based allowlist weaknesses. Zhao et al. [76] examined the security implications of not incorporating signature verification in mini-program plugins. Li et al. [56] developed XPOScope to detect and mitigate cross-user personal data leakage (XPO) problems in mini-programs.

Unlike the above research, our work focused on the practical security issues of the mini-program's web-app-like features. We systematically analyzed the problem and presented MiniCPRF, a novel vulnerability with mini-programs. Then, we developed MiniCAT to assess the prevalence of this vulnerability on a large scale.

Web App Security. CSRF vulnerability is one of the top security threats against web apps. Pellegrino et al. [63] developed a model-based dynamic analysis framework – Deemon, and identified 14 previously unknown CSRF vulnerabilities from 10 web apps. Khodayari et al. [55] focused on the vulnerability of the CSRF client side in web apps. They proposed JAW to detect client-side CSRF vulnerabilities using declarative traversals on hybrid property graphs. Jensen et al. [53] presented a static analysis tool, TAJS, to detect potential programming errors in JavaScript web apps. Kashyap et al. [54] provided JSAI, a robust abstract interpreter specified formally for JavaScript. Hedin et al. [52] proposed a security-enhanced JavaScript interpreter, JSFlow, for fine-grained information flow tracking. Park et al. [62] designed SAFE 2.0, a playground for advanced research in JavaScript web apps. However, all previous work was designed for web apps, and none of them can analyze the WeChat mini-program WXML file and build the attack path required by MiniCPRF, which is implemented by MiniCAT.

9 CONCLUSION

In this work, we presented a new vulnerability within mini-programs called MiniCPRF. This vulnerability is caused by the design flaw in the page routing and user state management of the mini-program framework, exacerbated by developer misuse. To evaluate the impacts of MiniCPRF, we developed MiniCAT, which analyzed 41,726 mini-programs. Finally, 13,349 mini-programs were identified as having potential MiniCPRF vulnerabilities. We responsibly reported our findings to the corresponding vendors, and three of them have been confirmed with assigned CNVD IDs. Moreover, we conducted a mini-program impact measurement based on pDNS records and multi-platform feasibility evaluations. We believe that MiniCPRF poses a certain security threat to the current mini-program ecosystem, and most developers are unaware of its existence.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable comments and suggestions. The authors from Shandong University were supported by National Natural Science Foundation of China (Grant No. 62372268), Shandong Provincial Natural Science Foundation (Grant No. ZR2023MF043), Taishan Young Scholar Program of Shandong Province, China (Grant No. tsqn202211001), and Xiaomi Young Talents Program. Yacong Gu was supported by Postdoctoral Fellowship Program of CPSF (Grant No. GZC20231361).

REFERENCES

[1] Accessed: 2024-01-27. *Alipay Documentation:Mini-Program Scheme.* https://opendocs.alipay.com/support/01rb18
[2] Accessed: 2024-01-27. *Alipay Mini-Program.* https://global.alipay.com/platform/site/product/mini-program
[3] Accessed: 2024-01-27. *Alipay open platform: how to get any Alipay small program appId and the page path.* https://open.alipay.com/portal/forum/post/17101017
[4] Accessed: 2024-01-27. *APKPure.* https://apkpure.net/
[5] Accessed: 2024-01-27. *Baidu smart mini-program:mini-program scheme .* https://smartprogram.baidu.com/docs/develop/function/opensmartprogram/
[6] Accessed: 2024-01-27. *Baidu Smart Program.* https://smartprogram.baidu.com
[7] Accessed: 2024-01-27. *Chinese National Vulnerability Database (CNVD).* https://www.cnvd.org.cn
[8] Accessed: 2024-01-27. *CNCERT/CC.* https://www.cert.org.cn/publish/english/index.html
[9] Accessed: 2024-01-27. *Code Composition of a WeChat Mini Program.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/quickstart/code.html
[10] Accessed: 2024-01-27. *CodeQL CLI.* https://docs.github.com/en/code-security/codeql-cli
[11] Accessed: 2024-01-27. *CodeQL for JavaScript.* https://codeql.github.com/docs/codeql-language-guides/codeql-for-javascript
[12] Accessed: 2024-01-27. *Decrypting WeChat DataBase.* https://www.forensicfocus.com/articles/decrypt-wechat-enmicromsgdb-database/
[13] Accessed: 2024-01-27. *Events in WeChat Mini-programs.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/view/wxml/event.html
[14] Accessed: 2024-01-27. *Google Play.* https://play.google.com/
[15] Accessed: 2024-01-27. *JavaScript HTML DOM Events.* https://www.w3schools.com/js/js_htmldom_events.asp
[16] Accessed: 2024-01-27. *jieba.* https://github.com/fxsjy/jieba
[17] Accessed: 2024-01-27. *Packages of WeChat Mini-Program.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/subpackages.html
[18] Accessed: 2024-01-27. *Passive DNS.* https://docs.umbrella.com/investigate/docs/passive-dns
[19] Accessed: 2024-01-27. *pywinauto.* https://github.com/pywinauto/pywinauto
[20] Accessed: 2024-01-27. *Routing in ExpressJS.* https://expressjs.com/en/guide/routing.html
[21] Accessed: 2024-01-27. *Sohu.* https://www.sohu.com
[22] Accessed: 2024-01-27. *Tencent WeChat.* https://www.wechat.com/en/
[23] Accessed: 2024-01-27. *The Metadata API for WeChat Mini-Program.* https://mp.weixin.qq.com/wxawap/waverifyinfo
[24] Accessed: 2024-01-27. *The Page Object in WeChat Mini-programs.* https://developers.weixin.qq.com/miniprogram/en/dev/reference/api/Page.html
[25] Accessed: 2024-01-27. *Tiktok mini-program:Generate Scheme.* https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/develop/server/url-and-qrcode/schema/generate-schema-v2
[26] Accessed: 2024-01-27. *TikTok Mini-programs.* https://www.tiktok.com/discover/mini-programs
[27] Accessed: 2024-01-27. *w3c:MiniApp Addressing explainer.* https://github.com/w3c/miniapp-addressing/blob/main/docs/explainer.md
[28] Accessed: 2024-01-27. *WeChat Mini-Program: Cloud Base.* https://developers.weixin.qq.com/miniprogram/en/dev/wxcloud/basis/getting-started.html
[29] Accessed: 2024-01-27. *WeChat Mini-program Code.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/qr-code.html
[30] Accessed: 2024-01-27. *WeChat Mini Program Host Environment.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/quickstart/framework.html
[31] Accessed: 2024-01-27. *WeChat Mini Program Login.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/login.html

[32] Accessed: 2024-01-27. *WeChat Mini-Program Network Ability.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/ability/network.html

[33] Accessed: 2024-01-27. *WeChat Mini-Program Payment.* https://pay.weixin.qq.com/wechatpay_h5/pages/product/miniapp.shtml

[34] Accessed: 2024-01-27. *WeChat Mini-Program Share & Forwarding.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/openability/share.html

[35] Accessed: 2024-01-27. *WeChat Mini-Programs.* https://mp.weixin.qq.com/cgi-bin/wx?token=&lang=en_US

[36] Accessed: 2024-01-27. *WeChat Mini Program's Frame Interface-Page.* https://developers.weixin.qq.com/miniprogram/en/dev/reference/api/Page.html

[37] Accessed: 2024-01-27. *WeChat Mini-Programs Page Routing.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/app-service/route.html

[38] Accessed: 2024-01-27. *WeChat Mini-programs Storage.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/ability/storage.html

[39] Accessed: 2024-01-27. *WeChat Payment Guide.* https://pay.weixin.qq.com/wiki/doc/api/wxpay/en/guide/pos/ReasonableQueryMechanism.shtml

[40] Accessed: 2024-01-27. *WeChat Revenue and Usage Statistics (2024).* https://www.businessofapps.com/data/wechat-statistics

[41] Accessed: 2024-01-27. *WeCom Mini-programs.* https://work.weixin.qq.com/wework_admin/wxcontacts/wxconnection_h5_guide?t=miniProgram

[42] Accessed: 2024-01-27. *wxappUnpacker.* https://github.com/system-cpu/wxappUnpacker

[43] Accessed: 2024-01-27. *WXML Introduction.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/view/wxml

[44] Accessed: 2024-01-27. *wxml-transformer.* https://github.com/imingyu/wxml-transformer

[45] Accessed: 2024-01-27. *wx.navigateTo.* https://developers.weixin.qq.com/miniprogram/en/dev/api/route/wx.navigateTo.html

[46] Accessed: 2024-01-27. *wx.redirectTo.* https://developers.weixin.qq.com/miniprogram/en/dev/api/route/wx.redirectTo.html

[47] Accessed: 2024-01-27. *wx.reLaunch.* https://developers.weixin.qq.com/miniprogram/en/dev/api/route/wx.reLaunch.html

[48] Accessed: 2024-04-27. *WeChat Revenue and Usage Statistics (2024).* https://developers.weixin.qq.com/miniprogram/en/dev/reference/api/getApp.html

[49] Accessed: 2024-06-12. *MiniCAT.* https://github.com/kee1ongz/MiniCAT

[50] Supraja Baskaran, Lianying Zhao, Mohammad Mannan, and Amr M. Youssef. 2023. Measuring the Leakage and Exploitability of Authentication Secrets in Super-apps: The WeChat Case. In *Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2023, Hong Kong, China, October 16-18, 2023.*

[51] Yifeng Cai, Ziqi Zhang, Ding Li, Yao Guo, and Xiangqun Chen. 2023. Shared Account Problem in Super Apps. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[52] Daniel Hedin, Arnar Birgisson, Luciano Bello, and Andrei Sabelfeld. 2014. JSFlow: tracking information flow in JavaScript and its APIs. In *Proceedings of the 29th ACM Symposium on Applied Computing (SAC), Gyeongju, Republic of Korea, March 24-28, 2014.*

[53] Simon Holm Jensen, Magnus Madsen, and Anders Møller. 2011. Modeling the HTML DOM and browser API in static analysis of avaScript web applications. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE), Szeged, Hungary, September 5-9, 2011.*

[54] Vineeth Kashyap, Kyle Dewey, Ethan A. Kuefner, John Wagner, Kevin Gibbons, John Sarracino, Ben Wiedermann, and Ben Hardekopf. 2014. JSAI: A Static Analysis Platform for JavaScript. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), Hong Kong, China, November 16 - 22, 2014.*

[55] Soheil Khodayari and Giancarlo Pellegrino. 2021. JAW: Studying Client-side CSRF with Hybrid Property Graphs and Declarative Traversals. In *Proceedings of the 30th USENIX Security Symposium (USENIX-Sec), August 11-13, 2021.*

[56] Shuai Li, Zhemin Yang, Yunteng Yang, Dingyi Liu, and Min Yang. 2024. Identifying Cross-User Privacy Leakage in Mobile Mini-Apps at A Large Scale. *IEEE Transactions on Information Forensics and Security* (2024).

[57] Wei Li, Borui Yang, Hangyu Ye, Liyao Xiang, Qingxiao Tao, Xinbing Wang, and Chenghu Zhou. 2023. MiniTracker: Large-Scale Sensitive Information Tracking in Mini Apps. *IEEE Transactions on Dependable and Secure Computing* (2023).

[58] Mengyi Long, Yue Xu, Jiangrong Wu, Qihua Ou, and Yuhong Nan. 2023. Understanding Dark UI Patterns in the Mobile Ecosystem: A Case Study of Apps in China. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[59] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying Resource Management Risks in Emerging Mobile App-in-App Ecosystems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS), Virtual Event, USA, November 9-13, 2020.*

[60] Shi Meng, Liu Wang, Shenao Wang, Kailong Wang, Xusheng Xiao, Guangdong Bai, and Haoyu Wang. 2023. Wemint: Tainting Sensitive Data Leaks in WeChat Mini-Programs. In *Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, September 11-15, 2023.*

[61] Aaditya Naik, Jonathan Mendelson, Nathaniel Sands, Yuepeng Wang, Mayur Naik, and Mukund Raghothaman. 2021. Sporq: An Interactive Environment for Exploring Code using Query-by-Example. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST), Virtual Event, USA, October 10-14, 2021.*

[62] Jihyeok Park, Yeonhee Ryou, Joonyoung Park, and Sukyoung Ryu. 2017. Analysis of JavaScript web applications using SAFE 2.0. In *Proceedings of the 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume.*

[63] Giancarlo Pellegrino, Martin Johns, Simon Koch, Michael Backes, and Christian Rossow. 2017. Deemon: Detecting CSRF with Dynamic Analysis and Property Graphs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), Dallas, TX, USA, October 30 - November 03, 2017.*

[64] Junjie Tao, Jifei Shi, Ming Fan, Yin Wang, Junfeng Liu, and Ting Liu. 2023. JSLibD: Reliable and Heuristic Detection of Third-party Libraries in Miniapps. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[65] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Taintmini: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis. In *Proceedings of the 45th IEEE/ACM International Conference on Software Engineering (ICSE), Melbourne, Australia, May 14-20, 2023.*

[66] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. One Size Does Not Fit All: Uncovering and Exploiting Cross Platform Discrepant APIs in WeChat. In *Proceedings of the 32nd USENIX Security Symposium (USENIX-Sec), Anaheim, CA, USA, August 9-11, 2023.*

[67] Shenao Wang, Yanjie Zhao, Kailong Wang, and Haoyu Wang. 2023. On the Usage-scenario-based Data Minimization in Mini Programs. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[68] Tao Wang, Qingxin Xu, Xiaoning Chang, Wensheng Dou, Jiaxin Zhu, Jinhui Xie, Yuetang Deng, Jianbo Yang, Jiaheng Yang, Jun Wei, and Tao Huang. 2022. Characterizing and Detecting Bugs in WeChat Mini-Programs. In *Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE), Pittsburgh, PA, USA, May 25-27, 2022.*

[69] Yue Wang, Yao Yao, Shangcheng Shi, Weiting Chen, and Lin Huang. 2023. Towards a Better Super-App Architecture from a Browser Security Perspective. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[70] Yuqing Yang, Yue Zhang, and Zhiqiang Lin. 2022. Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), Los Angeles, CA, USA, November 7-11, 2022.*

[71] Jianjia Yu, Zifeng Kang, and Yinzhi Cao. 2023. MiniTaintDev: Unveiling Mini-App Vulnerabilities through Dynamic Taint Analysis. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[72] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity Confusion in WebView-based Mobile App-in-app Ecosystems. In *Proceedings of the 31st USENIX Security Symposium (USENIX-Sec), Boston, MA, USA, August 10-12, 2022.*

[73] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. 2021. A Measurement Study of Wechat Mini-Apps. In *Proceedings of the 2021 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS), Virtual Event, China, June 14-18, 2021.*

[74] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS), Copenhagen, Denmark, November 26-30, 2023.*

[75] Zhibo Zhang, Zhangyue Zhang, Keke Lian, Guangliang Yang, Lei Zhang, Yuan Zhang, and Min Yang. 2023. TrustedDomain Compromise Attack in App-in-app Ecosystems. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*

[76] Yanjie Zhao, Yue Zhang, and Haoyu Wang. 2023. Potential Risks Arising from the Absence of Signature Verification in Miniapp Plugins. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS), Copenhagen, Denmark, November 26, 2023.*