MiniTaintDev: Unveiling Mini-App Vulnerabilities through Dynamic Taint Analysis

Jianjia Yu
Johns Hopkins University
Baltimore, MD, USA
jyu122@jhu.edu

Zifeng Kang
Johns Hopkins University
Baltimore, MD, USA
zkang7@jhu.edu

Yinzhi Cao
Johns Hopkins University
Baltimore, MD, USA
yinzhi.cao@jhu.edu

ABSTRACT

The security and privacy issues of mini-apps, which are lightweight apps that run inside host apps such as WeChat, have drawn the interest of researchers recently. We propose MiniTaintDev, a dynamic taint analysis tool for mini-app vulnerability detection, focusing on the detection of data leakage and sensitive API execution. We show MiniTaintDev with proof-of-concept attacks and some preliminary results in the work-in-progress (WIP) paper.

CCS CONCEPTS

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

KEYWORDS

Mobile security; Mini-app security; Taint analysis

ACM Reference Format:
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 '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3605762.3624434

1 INTRODUCTION

Mini-apps are lightweight apps that rely on host applications such as WeChat to provide users with various functionalities and features without leaving the host app. As one of the most popular social media applications worldwide, WeChat has 1.671 billion monthly active users in 2023 [2], with 928 million monthly active users of its mini-apps [3].

WeChat mini-apps leverage web technologies like HTML, CCS, and JavaScript. They utilize JavaScript engines such as V8 or JSCore for JavaScript interpretation and WebView to render the pages. WeChat offers a library of mini-app APIs [7] and an application for mini-app development, called Weixin DevTools [5]. Weixin DevTools uses NW.js [4] as the JavaScript engine, which combines Chromium and Node.js for developing cross-platform applications using JavaScript. Although WeChat mini-apps offer a variety of advanced features and have a large market share [9], they face security and privacy vulnerabilities such as data leakage and sensitive API execution [10].

SaTS '23, November 26, 2023, Copenhagen, Denmark

ACM ISBN 979-8-4007-0258-7/23/11.
https://doi.org/10.1145/3605762.3624434

Researchers have been working on security and privacy issues of mini-apps with both static and dynamic approaches. Zhang et al. [22] conduct a thorough measurement of WeChat mini-apps regarding security and privacy issues. They also implement an open-source tool for the crawling of WeChat mini-apps. Taintmini [19] uses static analysis to analyze the data flow of sensitive data and to detect privacy-sensitive data leaks and privacy policy violations. In terms of dynamic approaches, WeJalangi [15] rewrites the read and write operations in JavaScript but it suffers from high overhead. Overall, there has not yet been an efficient dynamic taint analysis engine for the detection of vulnerabilities in mini-apps. The reasons are mainly threefold: First, the host apps usually use their close-sourced customized JavaScript engines or WebView renderers. Such a feature brings incompatibility issues and runtime errors, increasing the difficulty of implementing a third-party vulnerability detection system. Second, the analysis engine is required to handle complicated cases of JavaScript executions in mini-apps, e.g., asynchronous executions, communications between mini-apps, and chained mini-app redirection. The engine should correctly propagate the taint before it reaches the sink function in those cases for the completeness of the analysis. Third, it is non-trivial to generate the correct inputs as the practical exploits to the mini-app for vulnerability validation, and thus severe consequences, e.g., sensitive data leakage and account takeover, could be unveiled and underestimated.

In this paper, we design, implement, and evaluate a dynamic taint analysis engine for mini-apps, MiniTaintDev, to efficiently detect vulnerabilities in WeChat mini-apps. MiniTaintDev is designed to tackle the three challenges mentioned above and thus fill in the research gaps. First, MiniTaintDev is built on Weixin Devtools and it runs across different platforms. Second, MiniTaintDev adds instrumentation on the mini-app API level to handle all of the complicated cases, e.g., rewriting `wx.navigateToProgram` and `wx.navigateFromProgram` to model the inter-mini-app communications. Third, MiniTaintDev further generates input/exploits to trigger vulnerabilities.

2 BACKGROUND

2.1 WeChat Mini-apps

Framework. The framework of a mini-app, its host app, and a third-party server is shown in Figure 1.

Mini-app consists of two layers: the logic layer and the rendering layer. The logic layer runs with a JavaScript engine, communicates with native APIs, processes data, and sends data to the rendering layer. It also listens to events sent from the rendering layer. The rendering layer gets the data from the logic layer and renders the views with the data while sending new events (possibly triggered by

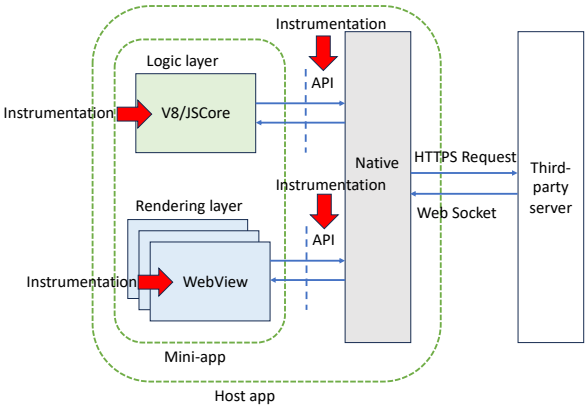

Figure 1: Framework of mini-app with MiniTaintDev

Table 1: Mini-app execution Environments

Platform	Logic Layer	View Layer
iOS/iPadOS/Mac OS	JavaScriptCore	WKWebView
Android	V8	XWeb[8]
Windows	Chromium Core	Chromium Core
Weixin DevTools	NW.js	Chromium Webview

user interaction) to the logic layer. The two layers communicate via the API, which is a design of WeChat to separate logic processing and view rendering. The API also converts Native Java methods to JavaScript code, allowing mini-app developers to utilize Native methods. Besides, the host app also communicates with a third-party server. The host app sends data to the third-party server by HTTPS requests and receives data by Web Socket. The framework is common in many super-app-mini-app ecosystems including Alipay, SnapChat, and Baidu.

Execution Environments. WeChat mini-apps run on several different platforms, including iOS/iPadOS, Android, Windows PC, Mac, and Weixin DevTools, as shown in Table 1. The reasons we chose Weixin DevTools to do the instrumentation are as follows: First, the JavaScript engine NW.js is open-sourced and Weixin DevTools runs across different platforms. Second, according to the documentation of WeChat miniapp [1], with the access of system resources from NW.js and JavaScript functionalities from Brower Object Model (BOM), Weixin DevTools is able to simulate almost all the functionalities of WeChat mobile apps. For example, Weixin DevTools utilizes `XMLHttpRequest` to simulate `wx.request` and `WebSocket` to simulate `wx.connectSocket`.

WeChat Mini-app Launch Scenes. Mini-apps can be accessed in plenty of ways, whether it be through a direct search in the WeChat search box, scanning a QR code, clicking a deep link, or through the mini-app message cards from WeChat chat sessions [6]. Users can also be redirected from one mini-app to another mini-app by `wx.navigateToMiniProgram`. Each way of access provides information such as the path to the target page and the query (if any). Upon launch, the mini-app will open the specific page with a query accordingly. After launch, the mini-app page can access the query through`App.onShow()`, `App.onLaunch()`, or `wx.getLaunchOptionsSyn()`. MiniTaintDev runs dynamic taint analysis on the mini-app pages with those APIs, tracking the data

```
wx.navigateToMiniProgram({
  appId: 'wx123456789',
  path: 'page/index/index?queryName=badQuery',
  extraData:{},
  success(res) {
    // open succeed
  }
})
```

Listing 1: An example of passing malicious queries with `wx.navigateToMiniProgram`. The part after '?' in path will become the query, which is the parameter of APIs such as `App.onLaunch()` and `App.onShow()`. The query can also be accessed through `wx.getLaunchOptionsSync()`.

flow of the query from those APIs to see whether they end up in some sensitive API sinks such as `wx.openBluetoothAdapter`.

2.2 Dynamic Taint Analysis

Dynamic taint analysis has been widely used in analyzing program code, e.g., C/C++ and JavaScript. It leverages concrete inputs to the program to perform taint tracking and thus enjoys the advantages of keeping the runtime context during analysis and having a low false positive rate. There has been a research trend of dynamic analysis on JavaScript [12, 13, 17, 18] which is to modify a modern JavaScript engine to conduct taint tracking for the purposes of defect or vulnerability detection. Our work echoes such research trends in that we adopt the V8 engine of NW.js for dynamic taint analysis to tackle challenges in mini-app vulnerability detection.

3 OVERVIEW

In this section, we present the overview of MiniTaintDev with the threat model and a proof-of-concept attack.

3.1 Threat Model

In our threat model, we consider two parties, a victim and an adversary. The victim is a vulnerable mini-app. The adversary can either be another mini-app navigating to the victim mini-app with malicious queries as shown in Listing 1 or any third party that crafts phishing links and the user is seduced to click the link, such as: weixin://encoded(pingduoduo-appID,path,malicious-url), as shown in [21]. Note that we assume the victim mini-app is vulnerable but not malicious, so the attacker has to make some attempts to make the attack happen. That is, in both cases, the adversary needs to pass some malicious queries in.

With the above settings, MiniTaintDev considers two possible vulnerabilities: First, the malicious query causes the control flow to go to some APIs which has access to sensitive information, and the information flows out to third parties through the network. For example, the attacker passes in a query that leads to the call of `wx.getLocation`, then the user's location is sent to a third party by HTTPS. Second, the data flow of the malicious query goes to some sensitive API sinks, causing something to happen unintended. For example, the attacker passes in a query which will be a parameter in `wx.openBluetoothAdapter`, which leads to accessing system resources.

```
1  /* pages/somepage?url=badUrl */
2  //app.js
3  App({
4      onLaunch(option){
5          wx.getLocation({
6              type: 'wgs84',
7              success (res) {
8                 wx.request({
9                   url: option.query.url,
10                  data: {
11                    x: res.latitude,
12                    y: res.longitude
13                  },
14                  header: {'content-type': 'application/json'},
15                  success(res){}
16                })
17             }
18         })
19     }
20 })
```

Listing 2: A proof-of-concept attack of data leakage. The exploit launch link (Line 1) and the vulnerable code of one page of a mini-app (Lines 3-20)

3.2 Proof-of-Concept Attack

In this section, we present a proof-of-concept attack for data leakage, as shown in Listing 2. The link to launch the mini-app contains the target page path and the query, which are normally controlled by the mini-app itself. Normally, the app.js works like this: upon launch, the `onLaunch` is called with a parameter `option`, which contains launch information such as path, query, and launch scene. In the function `onLaunch`, the mini-app calls `wx.getLocaion` to get the current location data of the mini-app user. If the `wx.getLocaion` call succeeds, the location data will be returned as the parameter `res` of the callback function `success`. Then with a call of `wx.request`, the location data, which contains properties such as `res.latitude` and `res.longitude`, will be sent to the URL `option.query.url`. `wx.request` also requires other parameters such as `header` and `success`, which is a callback function.

However, an attacker could craft such a launch link that targets the same page but with a malicious query, for example, the URL of a malicious website: `badUrl`. When the mini-app user is tricked into clicking on such a crafted link, his or her location information will be sent to the malicious website, causing a private data leakage.

4 DESIGN

The design of MiniTaintDev consists of three main components: dynamic taint analysis, input/exploit generation, and result validation. The dynamic taint analysis includes the instrumentation at two levels. The first is the JavaScript engine level, which involves the logic layer and the rendering layer. The second is the mini-app API level, which serves as the communication layer between the Native and the mini-app. Note that in WeChat mini-apps, `WeixinJSBridge` serves as the middle layers for communications between JavaScript and Native. The JavaScript method `WeixinJSBridge` is injected into the logic layer and rendering layers in mini-apps. It mainly provides four methods for communication: `on`, which collects the callbacks of events triggered by user interactions in mini-app rendering layers. `invoke`, which calls native methods and returns the results with a callback function. `publish`, which sends message to logic layer and `subscribe`, which listens to event callbacks from logic layer. Figure 1 shows where MiniTaintDev adds instrumentation marked with arrows. The instrumentation on API level mainly focuses on the `WeixinJSBridge` function. The instrumentation on the two levels allows MiniTaintDev to propagate taint from a WeChat mini-app API such as `wx.getLocation` or `wx.request` to JavaScript and to Native methods.

Regarding the input/exploit generation, MiniTaintDev considers the sink types and generates the corresponding queries, e.g., `?receiverUrl=https://attacker.tld` for data leakage. In terms of result validation, Weixin Devtools provides a feature that allows developers to add customized compilation mode, passing parameters for compiling, including the path to a specific page it will open after compilation, and the query to that page. MiniTaintDev utilizes this feature to validate the results of the dynamic taint analysis on Weixin Devtools. We discuss each component in detail in the following subsections.

4.1 Dynamic Taint Analysis

4.1.1 Taint Representation. Similar to [12], MiniTaintDev presents object-taints using a one-byte string, in which five bits represent the source type and the other three bits are unused. MiniTaintDev stores object-taints in a key-value map.

4.1.2 Sources and Sinks. Based on the two types of vulnerabilities MiniTaintDev detects, there are two sets of sources and sinks. First, for data leakage, the sources will be the mini-app APIs with access to sensitive data. Generally, the sensitive data will be returned as an object. For example, the API `wx.getClipboardData()`. The API will return with callback functions, `success`, `fail`, and `complete`. The parameter of `success` will be the content of the clipboard. In this case, the sinks are the network APIs, which makes it possible to leak sensitive data to third parties. Second, for sensitive API execution, the sources will be the query objects in APIs such as `wx.getLaunchOptionsSync()`, `App.onLaunch()`, and `App.onShow()`. The sinks will be the sensitive APIs such as `wx.openBluetoothAdapter`.

4.1.3 Taint Propagation. MiniTaintDev propagates taints from the source object, and sets and propagates object-taints for object lookups like `obj[prop]`, making sure the property objects of the query object are also tainted.

4.1.4 Multiple Taints Checking under the Same Context . In some cases, MiniTaintDev needs to check two taints under the same function call context to validate the vulnerability. For example, in Listing 2, MiniTaintDev has to check that both `url` and `data` are tainted and tainted by specific taint sources. Only certain sets of such taint sources will lead to a vulnerability.

4.2 Input/Exploit Generation

Input/Exploit generation is non-trivial for the detection of the aforementioned vulnerabilities in mini-apps. To make the attack practical, the adversary will carefully craft the input queries to lead to further consequences, e.g., system resources access, personal information leakage, or account manipulation. To this end, we design the input/exploit generation module in MiniTaintDev. The module is capable of generating specific input queries, i.e., the exploits

leading to corresponding further consequences and validating the result.

4.3 Result Validation

The main challenge for us to validate the results generated by MiniTaintDev is that WeChat adopts a review mechanism for most sensitive mini-app APIs, such as `wx.requestPayment` and `wx.getLocation`. Most of the sensitive APIs require a special application and reviewing process to be accessed by the developers of a specific mini-app, and some even require business authentications, making it non-trivial for us to obtain access to every mini-app API. To this end, we validate the potential vulnerabilities discovered by MiniTaintDev within Weixin Devtools, where some APIs will not be actually executed but the effect will be similar to the greatest extent when running on actual mobile devices.

5 IMPLEMENTATION

The Weixin Devtools on which we implemented MiniTaintDev is version 1.06.2209070 with NW.js 55, running on Linux. We instrumented the V8 engine and WebView renderer in NW.js in C++, with 139 file patches in total. We also instrument on the mini-app API level, which is written in JavaScript.

6 EVALUATION

Sensitive API List. We use the list of sensitive APIs from [22]. There are 31 in total, including 6 that can access app-specific data such as user profile or billing address, and 25 that can access system resources such as Bluetooth, camera, and user location.

Experimental Setups. We run the experiments on a server with 128G memory, 20 Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz cores, running Ubuntu 16.04.7 LTS.

Proof-of-Concept Results. Here we illustrate how MiniTaintDev detects the data leakage vulnerability in Listing 2. Note that the vulnerability shown in Listing 2 requires two data flows to make it vulnerable. Namely, the sensitive data flows to a network request, and the attacker-controlled string flows to the url of that network request. Therefore, in order to validate the data leakage vulnerability, there are two sets of source and sink that MiniTaintDev checks in this example. The first source is `option.query` and the corresponding sink is `url` in `wx.request`. The second source is `res` and the corresponding sink is `data` in `wx.request`. In the beginning, MiniTaintDev taints `option` object upon calling of `onLaunch` and taints `res` upon calling of callback function `success`. Note that MiniTaintDev taints object properties during object lookups. During execution, when `wx.request` acquires `option.query.url` , the taint flows from `option.query` to `option.query.url` and then to `url`. When `wx.request` acquires `res.latitude` and `res.longitude`, the taint flows from `res` to `res.latitude` and `res.longitude` and then to `data`. Finally, MiniTaintDev checks that both `data` and `url` are tainted under the same context of a `wx.request` call. After detecting the taint flows, MiniTaintDev generates the query with a `url` property with a value of a crafted host. Finally, MiniTaintDev launches the victim mini-app with Weixin Devtools and validates the vulnerability.

7 DISCUSSION

Extensibility. While our implementation of MiniTaintDev focuses on the WeChat platform, it is worth noting that MiniTaintDev is highly extensible to other NW.js- or Electron-based mini-app platforms such as AliPay and Baidu.

Ethics. We respect the security and privacy policies of WeChat and its mini-app platform during crawling and analysis. MiniTaintDev runs in the development environment provided by the official WeChat. When generating inputs/exploits, MiniTaintDev applies dummy strings so that no real-world mini-app is subject to any possible form of attack or harm. After we complete the evaluation, we will responsibly disclose the discovered vulnerabilities to corresponding mini-app owners and developers.

8 RELATED WORK

With the rapid growth of the number of mini-apps and their users, researchers have been conducting studies on the security and privacy issues of mini-apps.

Privacy Leakage. Wang et al. [19] and Li et al. [14] both propose to track the data flow of sensitive data for the detection of sensitive data leaks in mini-apps. Zhang et al. [23] conducted a measurement study to detect AppSecrect leaks in mini-apps and the consequences of this vulnerability such as account hijacking. As a comparison, MiniTaintDev focuses on the detection of privacy-related vulnerabilities comprising sensitive data leakage and sensitive API execution.

Security Breaches. Beside privacy issues, mini-apps are also subject to other vulnerabilities that raise security concerns. Yang et al. [20] introduce cross-mini-app request forgery attack, where the adversary pretends to be a legitimate mini-app to deceive the receiver mini-app, and launches subsequent attacks such as privileged data access and information leakage. Beyond the mini-app itself, the super-app that hosts mini-apps could also be subject to security breaches, such as resource management risks [16], identity confusion vulnerabilities [21], and mini-app-to-super-app authentication bypass [11]. We leave the detection of these security-related vulnerabilities in MiniTaintDev for future work.

9 CONCLUSION

The mini-app is a relatively novel type of JavaScript application running in full-fledged host apps, such as WeChat, TikTok, SnapChat, Alipay, and Baidu. While being lightweight and providing new features, it brings about security and privacy issues. In this paper, we propose MiniTaintDev, a dynamic taint analysis engine for detecting vulnerabilities in mini-apps, such as data leakage and sensitive API execution. We show a proof-of-concept attack and some preliminary results, showing the effectiveness of our method.

ACKNOWLEDGEMENT

We thank anonymous reviewers for their helpful comments and feedback. This work was supported in part by the National Science Foundation (NSF) under grants CNS-21-54404, CNS-20-46361, and CNS-1910133, and a Visa Research Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF or Visa.

REFERENCES

[1] [n. d.]. *About Mini Programs.* https://developers.weixin.qq.com/miniprogram/en/dev/framework/quickstart/.

[2] [n. d.]. *How Many People Use WeChat? User Statistics Trends (Aug 2023) (Source: https://www.bankmycell.com/blog/number-of-wechat-users/).*

[3] [n. d.]. *Number of monthly active users of WeChat Mini Programs in China from September 2020 to May 2023.* https://www.statista.com/statistics/1228315/china-number-of-wechat-mini-program-monthly-active-users/.

[4] [n. d.]. *nw.js.* https://nwjs.io/.

[5] [n. d.]. *Overview: Weixin public doc. Overview | Weixin public doc. (n.d.).* https://developers.weixin.qq.com/miniprogram/en/dev/devtools/devtools.html.

[6] [n. d.]. *Scene value list.* https://developers.weixin.qq.com/miniprogram/en/dev/reference/scene-list.html.

[7] [n. d.]. *WeChat mini-app API.* https://developers.weixin.qq.com/miniprogram/en/dev/api/.

[8] [n. d.]. *WeChat mini-app execution environments.* https://developers.weixin.qq.com/miniprogram/dev/framework/runtime/env.html.

[9] [n. d.]. *WeChat Mini Apps Risk Data Leaks.* https://timebusinessnews.com/wechat-mini-apps-risk-data-leaks/.

[10] [n. d.]. *WeChat mini programs for banking pose 'significant' risks of personal data leakage, says report.* https://www.scmp.com/tech/tech-trends/article/3142239/wechat-mini-programs-banking-pose-significant-risks-personal-data.

[11] Supraja Baskaran, Lianying Zhao, Mohammad Mannan, and Amr Youssef. 2023. Measuring the Leakage and Exploitability of Authentication Secrets in Super-apps: The WeChat Case. *arXiv preprint arXiv:2307.09317* (2023).

[12] Zifeng Kang, Song Li, and Yinzhi Cao. 2022. Probe the Proto: Measuring Client-Side Prototype Pollution Vulnerabilities of One Million Real-world Websites. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022.* The Internet Society. https://www.ndss-symposium.org/ndss-paper/auto-draft-207/

[13] Sebastian Lekies, Ben Stock, and Martin Johns. 2013. 25 million flows later: large-scale detection of DOM-based XSS. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security.* 1193–1204.

[14] 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).

[15] Yi Liu, Jinhui Xie, Jianbo Yang, Shiyu Guo, Yuetang Deng, Shuqing Li, Yechang Wu, and Yepang Liu. 2020. Industry practice of javascript dynamic analysis on wechat mini-programs. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering.* 1189–1193.

[16] 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.* 569–585.

[17] William Melicher, Anupam Das, Mahmood Sharif, Lujo Bauer, and Limin Jia. 2018. Riding out DOMsday: Towards Detecting and Preventing DOM Cross-Site Scripting. In *Network and Distributed System Security Symposium.* https://api.semanticscholar.org/CorpusID:3389782

[18] Marius Steffens, Christian Rossow, Martin Johns, and Ben Stock. 2019. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. https://doi.org/10.14722/ndss.2019.23009

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

[20] 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.* https://dl.acm.org/doi/pdf/10.1145/3548606.3560597

[21] 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 *31st USENIX Security Symposium (USENIX Security 22).* USENIX Association, Boston, MA, 1597–1613. https://www.usenix.org/conference/usenixsecurity22/presentation/zhang-lei

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

[23] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs. *ArXiv* abs/2306.08151 (2023). https://api.semanticscholar.org/CorpusID:259165528