Towards a Better Super-App Architecture from a Browser Security Perspective

Yue Wang
Ant Group
Hangzhou, Zhejiang, China
darcy.wy@antgroup.com

Yao Yao
Ant Group
Hangzhou, Zhejiang, China
vicky.yy@antgroup.com

Shangcheng Shi
Ant Group
Hangzhou, Zhejiang, China
shishangcheng.ssc@antgroup.com

Weiting Chen
Ant Group
Hangzhou, Zhejiang, China
weiting.cwt@antgroup.com

Lin Huang
Ant Group
Beijing, China
linyu.hl@antgroup.com

ABSTRACT

As multi-service mobile applications, the super-apps provide users with great convenience and satisfy most of our daily needs. Riding on the increasing popularity of super-apps, researchers from academia and industry have studied multiple aspects of mini-apps regarding security issues, including permission mechanisms, secure communication, access control, etc. However, little effort has been spent to analyze the underlying web technologies employed by super-apps. In this paper, we conduct the first study to understand the security mechanisms of super-apps from a browser perspective. We describe the relationship and significant differences between browsers and super-apps, especially the security features of traditional browsers and the challenges in applying them to super-apps. Further, we propose security guidelines about resources, storage, credentials, and privacy management to build a more secure super-app.

CCS CONCEPTS

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

KEYWORDS

super-app, browser, security, privacy

ACM Reference Format:
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 '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3605762.3624427

1 INTRODUCTION

Nowadays, (mobile) super-apps offer various services, such as payment, healthcare, and travel booking within a single app. Gartner predicts that over half of the global population will use multiple super-apps daily by 2027[17]. In contrast, mobile web apps work in mobile browsers, e.g., Safari, and can only access few native features from the underlying operating system. Therefore, mobile web apps provide limited services and suffer worse user experiences. As a workaround, super-apps usually host and support mini-apps within WebViews[5]. Moreover, the super-app provides powerful APIs[20] to its mini-apps via a JavaScript bridge. This hybrid solution facilitates mini-apps with both web technologies and native capabilities so that end-users can use mini-apps without installations and enjoy a similar experience to (mobile) native apps.

Motivated by the widespread adoption of mini-apps, several academic works have studied the security of associated super-apps. For example, Lu et al.[9] revealed the resource management vulnerabilities by super-apps, while Zhang et al.[32] explored an identity-confusing vulnerability in super-apps. On the other hand, Wang et al.[24] studied the vulnerabilities of hidden APIs in super-apps. Nevertheless, no existing works have studied the differences between super-apps and the browser, especially standardized security protection solutions for super-apps from the browser security perspective.

To bridge the gap, we first explore the relationship between super-app and mainstream browsers like Chrome[4] and Safari[7]. It reveals that the super-app takes multiple browser techniques to support the mini-app and thus inherits the threat model of the browser. Through in-depth study, the existing security features and security mechanisms in browsers are not applicable in super-apps. Therefore, a standardized security protection mechanism for super-apps is equally necessary but missing. In summary, this paper makes the following contributions:

- We discuss the relationship between browsers and super-apps, highlighting their notable similarities and differences.
- We explore the security features of traditional browsers and the challenges in applying them to super-apps.
- Finally, we propose a set of security guidelines based on identity entities.

We organize the rest of the paper as follows. In Section 2, we briefly demystify the relationship between super-apps and traditional browsers, including the security features of traditional browsers and their functions. Section 3 describes the differences between the super-app and the browsers and explains why security features for browsers cannot apply to super-apps directly. Then, we propose security guidelines for more secure super-apps in Section 4.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SaTS '23, November 26, 2023, Copenhagen, Denmark
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0258-7/23/11…$15.00
https://doi.org/10.1145/3605762.3624427

We review the related work in Section 5 and eventually conclude the paper in Section 6.

2 BACKGROUND

As common sense, in-browser web apps offer advantages such as cross-platform compatibility, lower development costs, and rapid iteration. Consequently, super-apps leverage similar browser technologies, including markup languages, rendering engines, JavaScript engines[25], and Web APIs[16], to benefit from these advantages.

In [30], three clusters of solutions adopted by super-apps are proposed, namely Integrated WebView, Customized Engine-based Mini-app, and WebView-based Mini-app. The Integrated WebView solutions refer to traditional browsers such as Chrome, Firefox, and Opera. On the other hand, the other two types of super-apps differ regarding the code distribution and execution environment. We will delve into these differences later in Section 3.1, while their commonalities are discussed first.

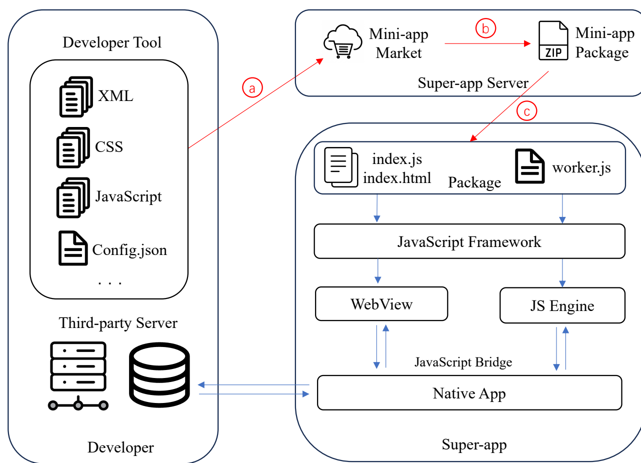

Figure 1: The Architecture of One Customized Engine-Based Super-App

2.1 Relationship Between Super-Apps and Traditional Browser

2.1.1 Similar Markup Language. Super-apps and browsers share similarities regarding front-end programming, utilizing common markup languages and CSS-based stylesheets. Specifically, mini-apps typically use HTML-like markup languages, CSS-like styling languages, and JavaScript.

2.1.2 Similar Rendering Engine and JavaScript Engine. Both super-apps and web browsers rely on rendering engines and JavaScript engines to parse and render mini-apps and web pages. A typical example of rendering engines in browsers is WebKit, while JSC[6] and V8[8] are the most widely used JavaScript engines. Toward this end, super-apps generally employ a rendering engine and JavaScript engine that is either the same or comparable to those utilized in web browsers.

2.1.3 Similar APIs. To either a mini-app or web application, interactions with the back-end and data storage are essential to fulfill various functionalities, such as user authentication, shopping order inquiries, and real-time communication. Traditional browsers provide a large number of Web APIs for web apps, while super-apps provide a wide range of JavaScript APIs to their mini-apps.

2.2 Threat Model

As discussed above, the super-app adopts multiple techniques from the browser so that its threat model is also inherited. Specifically, the super-app maps to the browser, while these webpages correspond to mini-apps. To characterize the security architecture of super-apps, we define the threat model as follows.

The Attacker's Capabilities:

- We assume that an attacker can develop or control multiple malicious mini-apps that try to evade the security checking by super-apps.
- The attackers may lure users into visiting their malicious mini-apps, e.g., scanning QR codes or invoking mini-apps through mobile applications on user devices.
- Attackers can access any publicly available online mini-apps and utilize them without any restrictions.

The Attacker's Goals: The attacker desires to acquire the user privacy stored within the super-app. Besides, the attacker also aims to steal the user data in other mini-apps, including the credential data.

2.3 Security Features of Traditional Browser

Web browsers adhere to strict security models to safeguard user data and privacy. Given the significant correlation between super-apps and browsers, the security features utilized by browsers serve as valuable references when establishing a security mechanism for super-apps. Therefore, we will focus on the popular mobile browsers, including Safari and Chrome.

To address the threat model discussed earlier, we will discuss three key security strategies: Same Origin Policy, Content Security Policy, and Privacy Features. Notably, we only highlight several essential security features related to the above threat model.

2.3.1 Same Origin Policy. The same-origin policy (SOP)[13] is a critical security mechanism of the web that restricts how a document or script loaded by one origin can interact with a resource from another[13]. The SOP restricts the behaviors, including cross-origin network access, cross-origin script API access, and cross-origin data storage access.

Cross-Origin Network Access.The method for cross-origin network access is to use CORS (Cross-Origin Resource Sharing)[27]. This mechanism allows the server to indicate the different origins (domain, scheme, or port) from which a browser should permit loading resources.

Cross-Origin Script API. Browser Object Model (BOM)[26], e.g., Window and Location objects as well as Web APIs like *window.open* and *window.opener*, allows the same origin documents to reference each other directly. However, cross-origin access to Window and Location properties is limited and only available to certain BOM properties in the browser.

Cross-origin Data Storage Access. In the scenario of web browsers, Web Storage APIs and IndexedDB are isolated by origin. In other words, one origin cannot read or write the data belonging to another origin.

2.3.2 Content Security Policy. Content Security Policy (CSP)[10] is a powerful security mechanism employed by websites to protect against various types of attacks, such as cross-site scripting (XSS) and data injection. It allows websites to define and enforce a set of rules that decide which resources and actions are allowed on their website. By implementing a CSP, websites can significantly reduce the risk of malicious activities and enhance overall security.

2.3.3 Privacy Features. Browsers offer privacy-related Web APIs that grant access to potentially sensitive data, and these Web APIs are typically available only in secure contexts[14], meaning they require HTTPS connections. Furthermore, these Web APIs are protected by a user permission system, ensuring that private data is only available to the page after the user grant.

3 DIFFERENCES AND NEW SECURITY CHALLENGES

As mentioned in Section 2, there are significant differences between super-apps based on Customized Engine-based mini-apps and WebView-based mini-apps, and traditional browsers. Specifically, code distribution and execution environment change in the context of super-app, leading to mini-app identity confusion issues. Subsequently, we discuss the challenges when super-apps adopt browser security features.

3.1 Differences Between Super-Apps and Traditional Browser

3.1.1 Code Distribution. For web apps, their code is deployed on the related web server such that the user can access these web pages dynamically through the URL online. On the contrary, it is the super-app that takes charge of the code distribution for its mini-apps. To be specific, the red path in Fig. 1 presents how to create and distribute a Customized Engine-based mini-app from the developer aspect. At the very beginning, the developer needs to use a dedicated IDE tool (from the super-app) to implement his mini-app, which is mainly made up of the following four parts.

- XML: files constructing the UI of a mini-app, e.g., buttons and text area, etc.
- CSS: CSS-style file(s), including font size and color, etc.
- JavaScript: file(s) for executing the logic of the mini-app, e.g., network communication.
- Config.json: a configuration file recording the basic information of a mini-app.

Once completed, the mini-app will be distributed to the end users for practical use, which consists of three steps.

a The developer uploads his code to the mini-app market hosted by the super-app server.
b The super-app server compiles the given code into a formatted mini-app package.
c Before the execution, the resultant mini-app package of the mini-app is distributed to the user's mobile phone.

A WebView-based mini-app follows a similar workflow, but the difference lies in that super-app does not distribute the mini-app package to user devices.

3.1.2 Execution Environment. Web applications are based on Web standards[15][21] and can be executed in any browser. While mini-apps usually can only run in a specific super-app. As shown in Fig 1., When the super-app starts a Customized Engine-based mini-app, it decompresses the mini-app package first and extracts the files inside for the actual execution later. More specifically, a browser thread handles the index.js and index.html files, where the WebView renders the UI pages and sets a universal virtual domain for the mini-app. The virtual domain of mini-apps is usually of the same origin to facilitate management and performance experience in the preload phase. Besides, the super-app creates a JavaScript engine thread to execute the worker.js file, namely the logic of the mini-app. Meanwhile, a customized JavaScript framework has been injected into the WebView and the JavaScript engine beforehand to register a JavaScript bridge object, mainly for calling APIs and the data-binding between the DOM[11] and JavaScript of mini-apps.

The difference of a WebView-based mini-app from the Customized Engine-based mini-app is that it does not download the mini-app package to the local device, but directly accesses resources through network requests. It utilizes the Android/iOS WebView for rendering. Similarly, the WebView-based mini-apps share the same domain, and the super-app registers a JavaScript Bridge.

3.2 Security Challenges

As mentioned above, Customized Engine-based mini-apps and WebView-based mini-apps tend to share the same domain defined by the underlying super-apps. The main reason for such practices is that the super apps cannot allocate separate domains for their mini-apps, whose total amount keeps increasing. Thus, these mini-apps need to use a unified domain. However, this customization introduces challenges for implementing access control mechanisms between mini-apps, as the inherent security features of web browsers rely on domain-based access control.

3.2.1 Resource Sharing. In a typical scenario, a mini-app uses network communication to fetch resources from its server. However, a potential attack vector arises when an attacker controls a malicious mini-app and impersonates other mini-apps to establish communication with their servers. Once such an exploit succeeds, the attacker can access user data from other mini-apps, including sensitive information like credentials.

Traditional web browsers employ CORS mechanisms to restrict requests from untrusted sources by utilizing HTTP headers such as "Access-Control-Allow-Origin". However, in the context of super-app, accurately determining the identity of the mini-app by the CORS protection mechanisms in the request becomes challenging.

Similarly, when a related webpage is embedded into the mini-app, there is a potential vulnerability where an attacker can use a malicious mini-app and embed that particular webpage. By deceiving the user into interacting with the embedded webpage, the attacker can compromise the personal data of the user.

In traditional web browsers, the Content Security Policy (CSP) mechanism enables embedded pages to specify the permitted domains for embedding. However, in the context of super-apps, where mini-apps share the same domain, the embedded page fails to differentiate between authorized and unauthorized mini-apps for embedding.

3.2.2 Storage. Inevitably, mini-apps require data storage capabilities for user preferences and offline functionalities. Traditional web browsers offer various Web APIs for storage, such as *localStorage* and *indexedDB*, which are isolated based on the same-origin policy security mechanism. As to super-app, a potential exploit appears where a malicious mini-app can gain unauthorized access to user data stored by other mini-apps through the browser-provided APIs.

3.2.3 Credentials. During the communication between a mini-app and its backend server, it is common to cache credentials, such as cookies, to authenticate the user identity. In traditional web browsers, the isolation mechanism of cookies is employed to prevent the leakage of credentials between different domains. Concerning super-app, the attacker can manipulate a malicious mini-app to gain access to the credential information of other mini-apps. They can also impersonate other mini-apps to establish communication with their respective backend servers, utilizing their credential information to steal the user data.

3.2.4 Privacy. In certain scenarios, mini-apps may require access to the user privacy, such as location or microphone access. Traditional web browsers provide a set of Web APIs to retrieve sensitive data. When a web app requests access to the privacy, the browser prompts the user with a pop-up, notifying them that a specific site is seeking their private information. Once the user grants the permission, the domain is added to a whitelist, and subsequent requests will be permitted. However, mini-apps share the same domain, so a potential attack surface arises. Once a user has granted a specific mini-app access to their sensitive data via the browser's Web API, an attacker can manipulate a malicious mini-app to retrieve the user's private information by invoking the Web API without the user's awareness.

4 GUIDELINES FOR A MORE SECURE SUPER-APP ARCHITECTURE

4.1 A Proposed Architecture for Super-Apps

Based on the above context, the super-app faces various risks of malicious mini-app impersonation as it cannot recognize the mini-app identity properly. To address this challenge, this article proposes a new security architecture for super-apps based on identity entity.

As shown in Fig 2, the Native part of the super-app contains three components: Handlers, Identity Center, and Security Modules. Handlers include Event Handlers and API Handlers, facilitating cross-process interactions with the WebView and JavaScript engine. They serve as the entry points for interface implementations. The Identity Center is the core of the security design model, maintaining the principal identity of each running mini-app. It is designed to be untamperable and unforgeable. The Security Modules consist of the Storage Manager, Privacy Manager, Resource Permission Checker, and Credential Manager.

The blue arrows in the diagram depict the process of security checks and handling mini-app events as follows:

(1) When the WebView and JavaScript engine call the JavaScript APIs or trigger certain WebView events, such as opening an embedded webpage, the events are sent to the Event Handlers and API Handlers through IPC.
(2) The Handlers first consult the Identity Center to retrieve the current identity of the mini-app. After that, the identity information is associated with the event data.
(3) The Handlers communicate with the security modules, which perform access control and isolation based on the identity of the mini-app.

It is noteworthy that the WebView and JavaScript engine are implemented by isolated processes in the sandbox. Therefore, even if they are exploited, the attackers cannot tamper with the Native process or its critical components, such as the Identity Center.

With the security architecture above, we will now discuss how it can address the four security challenges in the following.

4.2 Solutions to the Security Challenges of Super-apps

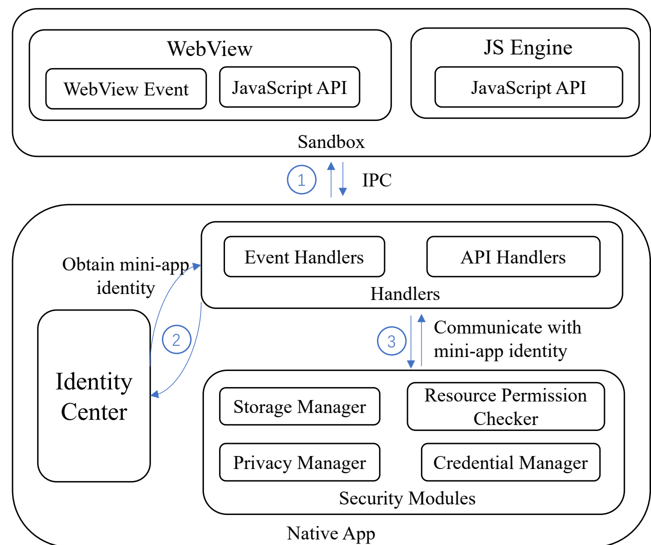

Figure 2: A New Security Architecture for Super-Apps Based on Identity Entity

4.2.1 Resource Permission Checker. Resource Permission Checker uses a whitelist mechanism to determine whether the mini-app can share resources as a replacement for CORS. This whitelist contains the domains associated with the mini-app, which have been verified beforehand. Specifically, the mini-app developers are required to declare their domains in the submitted source code (Step a in Fig. 1), and the super-app server responds with a verification file. Then, the developers need to place this file in the root directory of the claimed domains, which will be later verified by the super-app server by network access. Once the whitelist domain name is verified, the

mini-app is allowed to access resources of this domain name in the whitelist.

Super-app should restrict the usage of traditional browser Web APIs such as *Fetch* and *XMLHttpRequest* and instead utilize the network APIs provided by super-app. Nonetheless, enabling these Web APIs would bypass the Resource Permission Checker and potentially expose the risks mentioned in Section 3.2.1.

Similarly, when embedding a web page within a mini-app, the Resource Permission Checker will validate whether the domain of the embedded page presents in the whitelist associated with the current mini-app. Only when the domain appears, the page is allowed to be loaded.

4.2.2 Storage Manager. Storage Manager allocates a sandbox for each mini-app for file access. This sandbox can only be accessed by the respective mini-app to achieve read and write isolation. Due to the identity confusion issue of mini-apps, super-app should also restrict the usage of traditional browser storage Web APIs such as *localStorage* and *indexedDB* and instead provide new ones for its mini-apps.

4.2.3 Credential Manager. Credential Manager maintains a database for each mini-app, which stores information such as the domain, content, and expiration time of the credential. Through establishing access control mechanisms, each credential database is accessible only by its corresponding mini-app. During the communication between a mini-app and its backend server, the super-app stores the credential data from the server through the Credential Manager. Similarly, when a mini-app sends a request, super-app retrieves the cached credential data from the Credential Manager and includes it in the request.

4.2.4 Privacy Manager. Privacy Manager assigns a permission list to each mini-app. When a mini-app initially requests privacy permission, e.g., location access, super-app prompts the user that a specific mini-app is seeking location permission. Once the user grants authorization, super-app records this information to the permission list of the specific mini-app through the Privacy Manager. When the mini-app requests this permission again later, the super-app will check whether it has been granted and directly return the result if permitted.

Once again, the super-app should restrict the usage of traditional browser Web APIs for privacy permission requests, e.g., *geolocation*, and instead define new ones for their mini-apps. Besides, the super-app can enhance privacy permission management and allow users to revoke permissions. Notably, this feature has not appeared in most browsers[12].

5 RELATED WORK

Mini-app Security. As mentioned in Section 2, the rise of mini-apps has gained significant popularity in the field of mobile security. Zhang et al. [33] present the first large-scale mini-app crawler MiniCrawler and measure the API usage and obfuscation rate of the mini-apps. [34] also uses a mini-apps crawler and specially implements a master key leakage inspector to detect the master keys leaked in mini-apps. [22] presents TaintMini, a static taint analysis framework that can detect the flow of sensitive data in mini-apps and identified privacy risks. However, these studies lack the analysis of the security mechanism of super apps.

Several studies have investigated the security weaknesses that can be exploited by malicious mini-apps. For instance, [31] presents a novel cross-miniapp request forgery (CMRF) attack, which is caused by the missing checks of the sender's AppID in a receiver mini-app. And Zhang et al.[32] explore an identity-confusing vulnerability in super-apps. On the other hand, Wang et al [24] [23] study the vulnerabilities of hidden APIs and discrepant APIs in super-apps leading to security and privacy issues. In contrast to prior efforts, our work presents a standardized security protection solution for super-apps, avoiding common identity confusion flaws and implementing appropriate security controls.

Browser Security. The browser security is related to web extensions [2, 3, 28] and security architecture[1, 18, 35]. For example, [19] reveals web applications and exploits extension privileged capabilities to bypass SOP for accessing user data or credentials on any other web application. Regard the security architecture, Yang et al. [29] believe that information flow control(IFC) is a promising replacement for existing browser security mechanisms. Regardless of the advancements in web browser security mechanisms, they are generally not directly applicable to super-apps due to the inherent differences between the two. To ensure the security of super-apps, it is necessary to re-implement specialized security mechanisms specifically tailored for them.

6 CONCLUSION

In this paper, we discuss the relationship between browsers and super-apps, highlighting their typical differences. We explore the security features of traditional browsers and the challenges when applying them to super-apps. Finally, we propose security guidelines based on identity entity to build a more secure super-app.

Nonetheless, it is important to notice that the security solutions listed in this paper are not the only options that a super-app may adopt, which are crucial security strategies derived from a browser perspective. When designing and developing a super-app, it is also necessary to consider other security features and measures, such as data encryption, access control, etc., to ensure comprehensive security of the application.

REFERENCES

[1] Nataliia Bielova. 2013. Survey on JavaScript security policies and their enforcement mechanisms in a web browser. *The Journal of Logic and Algebraic Programming* 82, 8 (2013), 243–262.

[2] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An Evaluation of the Google Chrome Extension Security Architecture. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association, Bellevue, WA, 97–111. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/carlini

[3] Quan Chen and Alexandros Kapravelos. 2018. Mystique: Uncovering Information Leakage from Browser Extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 1687–1700. https://doi.org/10.1145/3243734.3243823

[4] Chrome. 2023. Google Chrome Browser. Retrieved 2023 from https://www.google.com/chrome/

[5] Google. 2023. WebView. Retrieved June 7, 2023 from https://developer.android.com/reference/android/webkit/WebView

[6] Apple Inc. 2023. JavaScriptCore. Retrieved 2023 from https://developer.apple.com/documentation/javascriptcore

[7] Apple Inc. 2023. Safari Browser. Retrieved 2023 from https://www.apple.com/safari/

[8] Google LLC. 2023. V8 (JavaScript engine). Retrieved 2023 from https://v8.dev/

[9] 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 *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 569–585. https://doi.org/10.1145/3372297.3417255

[10] MDN. 2023. Content Security Policy (CSP). Retrieved July 7, 2023 from https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

[11] MDN. 2023. Introduction to the DOM. Retrieved May 21, 2023 from https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

[12] MDN. 2023. Permissions: revoke() method. Retrieved April 8, 2023 from https://developer.mozilla.org/en-US/docs/Web/API/Permissions/revoke

[13] MDN. 2023. Same-origin policy. Retrieved July 4, 2023 from https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[14] MDN. 2023. Secure contexts. Retrieved Jul 4, 2023 from https://developer.mozilla.org/en-US/docs/Web/Security/Secure_Contexts

[15] MDN. 2023. The web and web standards. Retrieved August 22, 2023 from https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/The_web_and_web_standards

[16] MDN. 2023. Web APIs. Retrieved Feb 20, 2023 from https://developer.mozilla.org/en-US/docs/Web/API

[17] Lori Perri. 2022. What Is a Superapp? Retrieved September 28, 2022 from https://www.gartner.com/en/articles/what-is-a-superapp

[18] Charles Reis, Adam Barth, and Carlos Pizano. 2010. Browser Security: Lessons from Google Chrome Google Chrome developers focused on three key problems to shield the browser from attacks. *Communications of the Acm* 52, 8 (2010), 45–49.

[19] Dolière Francis Somé. 2019. EmPoWeb: Empowering Web Applications with Browser Extensions. In *2019 IEEE Symposium on Security and Privacy (SP)*. 227–245. https://doi.org/10.1109/SP.2019.00058

[20] W3C. 2022. MiniApp Standardization White Paper version 2. Retrieved July, 2022 from https://www.w3.org/TR/mini-app-white-paper/#api_and_component

[21] W3C. 2023. Web Standards. Retrieved 2023 from https://www.w3.org/standards/

[22] 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

[23] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. One Size Does Not Fit All: Uncovering and Exploiting Cross Platform Discrepant APIs in WeChat. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity23/presentation/wang-chao

[24] Chao Wang, Yue Zhang, and Zhiqiang Lin. 2023. Uncovering and Exploiting Hidden APIs in Mobile Super Apps. *CoRR* abs/2306.08134 (2023). https://doi.org/10.48550/arXiv.2306.08134 arXiv:2306.08134

[25] Wiki. 2023. JavaScript engine. Retrieved August 6, 2023 from https://en.wikipedia.org/wiki/JavaScript_engine

[26] wikipedia. 2023. Browser Object Model. Retrieved May 22, 2023 from https://en.wikipedia.org/wiki/Browser_Object_Model

[27] wikipedia. 2023. Cross-origin resource sharing. Retrieved July 4, 2023 from https://en.wikipedia.org/wiki/Cross-origin_resource_sharing

[28] Xinyu Xing, Wei Meng, Byoungyoung Lee, Udi Weinsberg, Anmol Sheth, Roberto Perdisci, and Wenke Lee. 2015. Understanding Malvertising Through Ad-Injecting Browser Extensions. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 1286–1295. https://doi.org/10.1145/2736277.2741630

[29] Edward Yang, Deian Stefan, John Mitchell, David Mazières, Petr Marchenko, and Brad Karp. 2013. Toward Principled Browser Security. In *14th Workshop on Hot Topics in Operating Systems (HotOS XIV)*. USENIX Association, Santa Ana Pueblo, NM. https://www.usenix.org/conference/hotos13/session/yang

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

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

[32] 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

[33] Yue Zhang, Bayan Turkistani, Allen Yuqing Yang, Chaoshun Zuo, and Zhiqiang Lin. 2021. A Measurement Study of Wechat Mini-Apps. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 2 (2021), 1–25.

[34] Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. Don't Leak Your Keys: Understanding, Measuring, and Exploiting the AppSecret Leaks in Mini-Programs. *CoRR* abs/2306.08151 (2023). https://doi.org/10.48550/arXiv.2306.08151 arXiv:2306.08151

[35] Marin Šilić, Jakov Krolo, and Goran Delač. 2010. Security vulnerabilities in modern web browser architecture. In *The 33rd International Convention MIPRO*. 1240–1245.