SoK: Decoding the Super App Enigma: The Security Mechanisms, Threats, and Trade-offs in OS-alike Apps

Yuqing Yang
The Ohio State University

Chao Wang
The Ohio State University

Yue Zhang
The Ohio State University

Zhiqiang Lin
The Ohio State University

Abstract

The super app paradigm, exemplified by platforms such as WeChat and AliPay, has revolutionized the mobile app landscape by enabling third-party developers to deploy add-ons within these apps. These add-ons, known as miniapps, leverage user data hosted by the super app platforms to provide a wide range of services, such as shopping and gaming. With the rise of miniapps, super apps have transformed into "operating systems", offering encapsulated APIs to miniapp developers as well as in-app miniapp stores for users to explore and download miniapps. In this paper, we provide the first systematic study to consolidate the current state of knowledge in this field from the security perspective: the security measures, threats, and trade-offs of this paradigm. Specifically, we summarize 13 security mechanisms and 10 security threats in super app platforms, followed by a root cause analysis revealing that the security assumptions still may be violated due to issues in underlying systems, implementation of isolation, and vetting. Additionally, we also systematize open problems and trade-offs that need to be addressed by future works to help enhance the security and privacy of this new paradigm.

1 Introduction

The super app paradigm pioneered by WeChat has emerged as a convenient way for mobile applications catering to massive amount of users. It enables seamless integration and management of third-party services, which are developed and submitted by external developers as add-ons. These third-party services typically take the form of miniapps (self-contained code packages submitted to super apps) that are distributed to and executed within end users' host apps. By leveraging native services like payment and accessing user data stored within the super app, these miniapps greatly expand the range of services inside super apps, effectively transforming the super app into an app ecosystem akin to an operating system.

As a versatile solution for super apps to provide customizable and light-weight services to end users, an increasing number of mobile giants are transforming into super apps in seek of a novel channel for increasing users' stickiness and also monetary profits. For instance, WeChat (a super app with 1.2 billion users) debuted the "miniapp" paradigm in 2017. In Tencent's Q3 2023 report, WeChat have hosted more than 3.5 million miniapps, providing services to over 600 million Daily Active Users (DAU), and generating over 2.7 trillion RMB in transactions [47]. Witnessed by the profits generated from this novel paradigm, there had been more than 20 top mobile app platforms that transformed into super apps, and this includes WeChat in China (social, 2017) [34], Kakao in Korea (social, 2017), Rakuten in Japan (shopping, 2019), and Zalo in Vietnam (social, 2021) [49], since this paradigm has made these apps "*sort of like Twitter, plus PayPal, plus a whole bunch of other things. And all rolled into one great interface*", as mentioned by Elon Musk [36].

With the advent of miniapps, super apps have transformed into comprehensive "operating systems" by integrating an execution environment along with encapsulated APIs, such as `getUserInfo` [55], which manages resource access from miniapps to user account information from the super app. These miniapps are essentially web front-ends that are packaged to run on customized execution engines provided by the super app platforms. They leverage super-app-specific JavaScript and Cloud APIs to deliver their functionalities. This parallels traditional mobile operating systems like Android or iOS, where the OS providers offer the environment and programmable APIs. Additionally, there is an in-app miniapp store, akin to platforms such as *Apple AppStore* and *Google Play*, where users can explore and download a diverse array of uploaded and vetted miniapps.

However, now that the third-party has been given the access to super app managed resources, including a vast amount of user information, cloud services, and OS resources, it is crucial to understand the security challenges and risks it posts to the super app platform and its end users, as the security threats exploited by attackers may affect massive amount of users. To this end, in this paper, we present the first systematic study to reveal the security and privacy of super app paradigms by thoroughly scrutinizing and dissecting the protection mechanisms, security threats, and their root causes. To be more specific:

- **Evolution and Taxonomy (§2).** We analyze the historical evolution of the super app ecosystem and miniapps, comparing them with web apps and native apps to highlight

the recent advancements and difference from traditional paradigms. Our study provides the first taxonomy of super apps based on the architecture and implementation of environments that support the execution of miniapps.

- **Security Mechanisms (§3).** We conduct a comprehensive analysis of 13 security measures enforced by super apps through the lens of the communication between miniapp front-ends, back-ends, and super apps. These mechanisms include those adopted from traditional paradigms and adapted to super apps (e.g., sandboxing), enforced to isolate access ot services and data from unexpected parties (e.g., API restriction), and implemented to scrutinize third-party code for thwarting malware (e.g., code vetting).
- **Security Threats (§4).** We systematically analyze the security threats by summarizing the security assumption on each security mechanisms and potential risk of violation. Consequently, we identify 10 threats that have been confirmed by published documents including research papers, security reports, and announcement from super app platforms. The root causes to these attacks include compatibility issue caused by different implementation of security mechanisms in underlying OSs, implementation issue in super app security mechanisms, trust issue between the platform and developers, and vetting issue due to limitation of the vetting mechanisms.
- **Lessons Learned (§5).** Based on the security threats and root causes, we systematically compile our observations into 5 lessons learned on the challenges super apps face that may provide insight for designing security mechanisms and mitigating the security threats.
- **Open Problems (§6).** Based on the security analysis and lessons we summarized, we identified trade-offs super app platforms face and need to address, as well as 4 open problems, including performing automated security analysis, standardizing security mechanisms, educating developers on security configurations, and developing semantic-aware miniapp vetting. These open problems may be addressed by future works to help enhancing the security and privacy of super app platforms.

2 Super Apps in a Nutshell

2.1 Characteristics of Super Apps

A super app is a mobile application that integrates multiple services offered by third-parties, providing users with a diverse array of functions within a unified app. Unlike traditional approaches that require separate applications for distinct purposes like messaging, ride-hailing, and shopping, a super app integrates these services into a centralized platform by allowing third-party developers to integrate miniapps inside the super app. These services

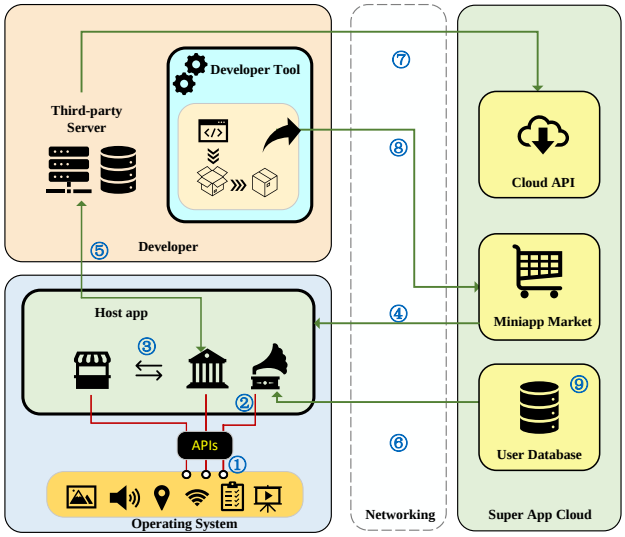

Figure 1: Overview of the key components of super apps

can be accessed by users on-demand via built-in miniapp store, without requiring to install the miniapps on users' mobile devices. In general, these miniapps are akin to Progressive Web Applications (PWA) as derived from the W3C standards, which include HTML, CSS, JavaScript (JS) scripts, as well as manifest files to configure functional and security mechanisms. However, encapsulated APIs are provided by super apps for miniapps to access a variety of native services and user data managed by the super apps. Hence, compared with web apps, the miniapps have more powerful capabilities to interfere with System- and Superapp-specific resources. For example, a miniapp can directly launch in-app payment or pair Bluetooth devices via miniapp JS API, which is handled by the super apps through an interface layer called JavaScript bridge.

As third-party developers are involved in this paradigm, the communication of the super app paradigm generally involves three parties as depicted in Figure 1: the developers (orange box), the super app cloud (green box), and the host apps that are downloaded to mobile devices of end users (blue box). On end devices, host apps operate on top of the underlying operating system on end users' devices, managing both local OS resources available to the super apps (such as Bluetooth and GPS location) and cloud resources hosted by super apps (such as the phone numbers and addresses of users). Access to both local and cloud resources is then made convenient for developers, as they only need to invoke these highly-encapsulated APIs without having to worry about underlying mechanisms. For instance, when accessing location data, developers only need to invoke `getLocation()`, without having to implement mechanisms to prompt the users asking for permission, as these are performed by super apps without interference from developers. However, to

ensure the security of the ecosystem, registering developer accounts and providing proof of identity is required. These developers are being rigorously vetted to prevent malicious developers from sabotaging the platform.

2.2 Taxonomy and Evolution

While WECHAT introduced the concept of miniapps in 2017, similar paradigms of integrating add-ons into standalone apps have existed prior to that. For example, Google Chrome introduced web extensions in 2009 [52], allowing developers to submit add-ons for the browser. Pioneered by these browsers, when super apps seeks to enable third-party service integration into a single app and to address the fundamental problem of providing cross-platform execution support, the W3C standard have become a feasible solution to follow, because the techniques adopted by web apps and browsers from W3C standards have been developed and tested over the years of web browsing, and these standardized solutions support web app browsing across platforms by design, which suits the needs of super apps. Hence, these super apps followed the techniques used by W3C such as JavaScript, HTML, and CSS, although certain customization have been made to allow native capabilities. In the past five years, the concept of a super app, i.e., native apps supporting integrated add-ons, has extended beyond WECHAT. As shown in Figure 2, this paradigm has been adopted by over 20 platforms providing diverse services. Among these super app platforms, the foremost notable feature, as well as the top-level classification criteria, is how the execution environment is implemented to support the add-ons or miniapps to free the miniapp developers from concerns about compatibility across the underlying operating systems (Android, iOS, or Windows). As shown in Figure 2, there are three clusters of solutions adopted by super apps: browser-native environment, cross-platform-framework environment, and super-app integrated environment.

- **(A) Integrated WebView.** Browsers such as *Chrome* and *Opera* began providing APIs for extensions to access services provided by their browser kernels in 2009. These extensions are essentially packages distributed to end users' devices containing JavaScript, HTML, and CSS files, along with a manifest file specifying information about the extension and permission required, which can interact with users' browsers for more powerful functionalities compared with normal web apps. For instance, upon launching an extension, the injected `chrome` object provides developers with access to functionalities such as monitoring the active tab being browsed by users. Meanwhile, the super apps extend their services by integrating browser kernels to support in-app web browsing, with customized JavaScript objects to extend the functionalities as browsers do, marking the precedent of the miniapp paradigm.

- **(B) Customized Engine-based Miniapps.** While in-app browsing enable users to visit web pages providing various services without having to open an external browser app, the performance and security have long been an issue for these super apps. First, opening web pages require the super apps to load resources each time, resulting in extended loading time showing blank pages to end users, especially when the network is slow or unstable, significantly undermining user experiences [41]. Meanwhile, the super apps could not vet the third-party websites for malicious code, which may allow attackers to exploit the super apps. To address these problems, super apps including WECHAT began to optimize the execution environment by separating logic execution and rendering into different threads to optimize the multi-procedural capability of mobile devices. Meanwhile, third-party developers are forced to submit packed web apps to the WeChat platform, and these packages are only released after being vetted, which significantly reduces the threats from malware. As illustrated on the green arrows in Figure 2, while different companies implement different versions of execution engines, the separated and customized execution environment implemented by these super apps remain identical, and some company's frameworks (Nebula from Alibaba) are even adopted in super apps in Africa (MPesa and Voda Pay).

- **(C) Webview-based Miniapp.** Apart from platforms that customize and refactor the entire execution environment, these super apps including ZALO directly adopt Android WebView to support the execution, while still requiring developers to submit miniapps to the platform to be vetted. Instead of letting users download the packages, these packages are hosted online on the platform's CDN (e.g., `h5.zdn.vn` for ZALO). However, while these packages are not distributed to end users' devices as type (B) super apps do, the miniapps are still vetted by the platforms, as the developers still have to submit their packages to the platforms before these packages are hosted on CDNs of super apps. Hence, whether third-party code is vetted and executed within the application is a core criterion to differentiate traditional paradigms from super app paradigms.

2.3 Comparision

While sharing similarities with W3C-derived paradigms such as web pages and PWAs, super apps present unique characteristics and serve specific purposes from the perspectives of platforms, developers, and end users:

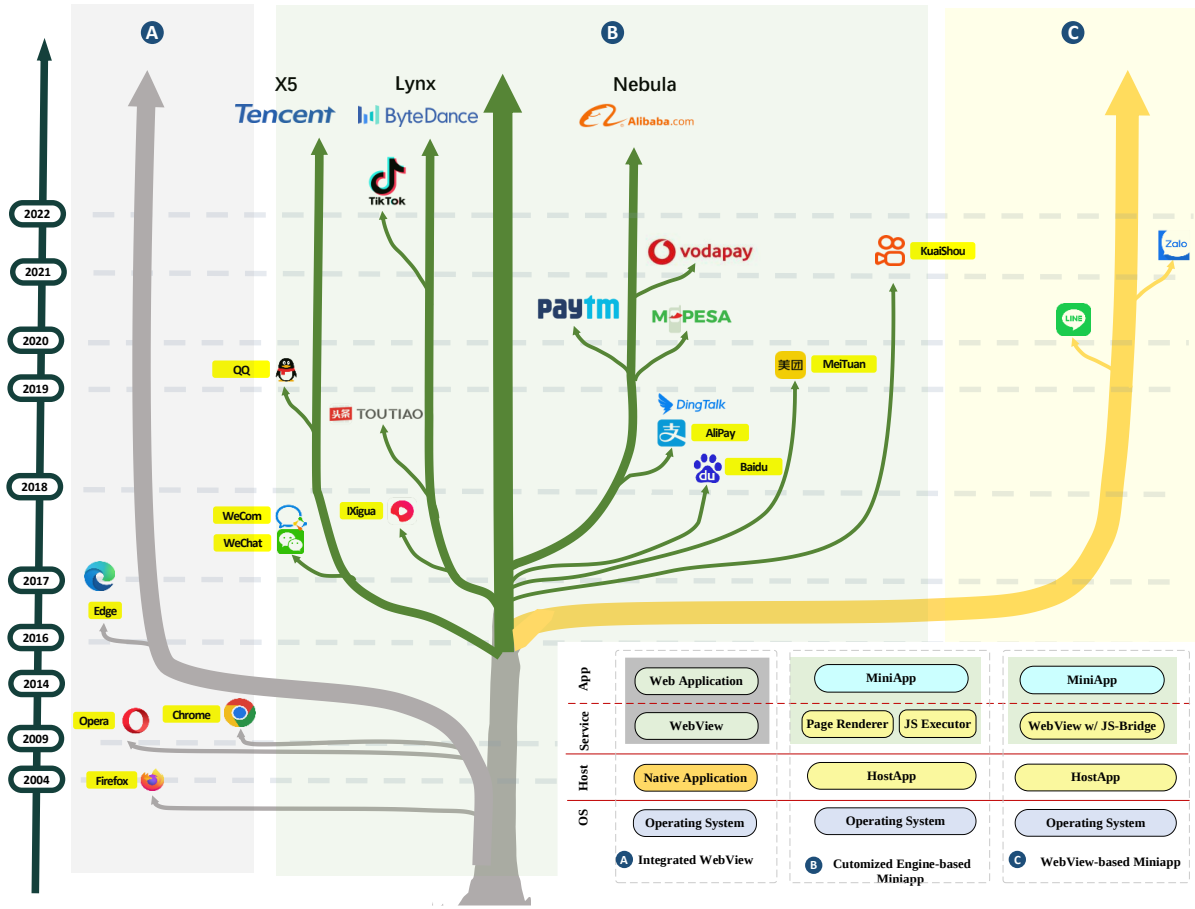

Figure 2: The taxonomy of super app platforms. The architectures of each cluster are shown at the bottom right, where resources in boxes with dotted lines are optional.

Platforms. As the critical component to support third-party code execution, super apps have unique differences compared with traditional systems. As described in Table 1, we use Android, Chrome, and WeChat as examples to illustrate the core differences between these platforms.

- **Code Distribution.** In web pages, the codes (including HTML, CSS, and JavaScript) are distributed via network and referred to online, whereas in native operating systems, app codes are generally distributed as packages, which can be downloaded and installed. Similar to native systems, super apps adopt the packed and self-contained means of app distribution. However, compared with Android system, super apps seek more strict control on the distribution of miniapps to prevent malware distribution. To achieve this, super apps prohibit the users from installing miniapps from third-party sources, and all miniapps must be accessed via the exclusive in-app app store.

- **Managed Resources.** Traditional Operating Systems like Android mainly manage local resources on the users' devices such as Bluetooth connection and storage. However, super apps are built upon these resources and also allow developers to access the cloud-stored user data. Meanwhile, the super apps feature exclusive app store, which provides the super apps means to manage and vet miniapp packages uploaded by third-party developers. Hence, these packages also automatically become managed resource of super apps. On top of that, a fundamental difference between super apps and other paradigm is that registering an account is mandatory for using or developing miniapps in super apps, which grants super apps the ability to manage these accounts as well. *In short, the cloud-stored data, miniapp packages, and super app services differentiate super apps from traditional systems in terms of additional types of managed resources.*

- **Execution Environment.** While the miniapps running in super apps are derived from the similar technologies from W3C standards, such as JS, HTML, and CSS, the miniapp

execution environments are heavily customized and extended beyond the W3C standards by the super apps, with a highlight on access to super app and local resources as mentioned in managed resources. Consequently, such customization makes the execution environment to grant miniapps more native capabilities to interfere with sensitive data and resources.

Developers. As the underlying execution engines are extended web execution engine, from the developer's perspective, developing a miniapp is similar to web apps, but with a more customized set of JS APIs and UI components. Contrary to web apps whose front-end components and scripts have to be delivered from a web domain to users' browsers, the developers of miniapps do not necessarily need to maintain a back-end for delivering the front-end code, as the code is submitted and hosted by the super apps. Moreover, even if some developers require cloud back-ends to process data, the super apps still provide cloud database and cloud function features for developers, serving as a means for super apps to generating more profit via cloud services. Hence, developers do not need to have a back-end to develop a miniapp.

End Users. From an end user's perspective, miniapps, mobile apps, and web apps offer different experiences and functionalities. Miniapps are typically dependent on a super app for updates, and their installation occurs within the super app itself. Mobile apps, conversely, rely on app stores for updates and are installed as individual entities on devices. In contrast, web apps update seamlessly via web servers and are directly accessible through web browsers. Performance-wise, the efficiency of miniapps varies according to the underlying super app or platform. Mobile apps, however, can capitalize on device-specific hardware optimization, providing high-performance experiences. The performance of web apps hinges on factors like Internet connectivity, browser capabilities, and server response times. Despite these variables, native apps usually offer superior speed and responsiveness.

In terms of offline functionality, miniapps may have limited capabilities based on the super app or platform, whereas mobile apps can offer extensive offline features. Traditional web apps typically necessitate an Internet connection for functionality, but progressive web apps (PWAs) provide some offline functionality through caching and service workers. Regarding user login and registration, users of browsers and Android do not have to register account prior to accessing extensions and apps, whereas super apps cannot be used without registering an account, as accounts are considered a resource hosted by super apps, and super apps enforce account management against malicious users.

2.4 Threats to Validity

It is worth to note that the super app paradigm is a novel paradigm and super app platforms are still emerging rapidly across the globe. Although certain platforms such as *Grab* label themselves as super apps, their extension of services is mostly confined to implementing individual webviews on their own. They have yet to fully embrace third-party developer integration within their apps. *Therefore, our paper primarily focuses on those super app platforms that not only enable third-party development but also exclusively host and vet the packages. These characteristics distinguish them from traditional paradigms, making them core to our investigation.* To this end, we have identified 15 super apps that fit these criteria, and a more comprehensive list of the platforms we've examined can be found in Table 5 of Appendix A. Also, to illustrate the security measures and potential threats associated with super apps, we have chosen to use WeChat as an example in the rest of this paper, given its widespread popularity.

Hosts	**Mobile OS (Native Apps)**	**Web Browsers (Web Apps)**	**Super Apps (Miniapps)**
Example Platform	Android	Chrome	WeChat
Platforms			
Managed resources			
Local Resources			
System Resources?	●	◑	●
Super-app Services?	○	◑	●
Cloud Resources			
User Data/States?	◑	●	●
Account?	●	●	●
App Packages?	○	○	●
Cloud Services?	◑	●	●
App Distribution			
App Store?	●	○	●
Exclusive?	◑	○	●
Packed?	●	○	●
Environment	Android Kernel	Web Engine	Web Engine (customized)
Developers			
Language?	C,Java, XML	JS, HTML, CSS	JS, WXML, WXSS
API Support?	Rich	Poor	Median
Compatible with Platforms?	○	●	●
Backend?	◑	●	◑
Centralized Vetting?	●	○	●
Users			
Install-free?	○	●	●
Market?	●	○	●
Storage Consumption?	High	Low	Low
Update?	Client-based	Client-based	Server-based
Performance?	High	Browser-specific	Super-app-specific
Offline Loading?	High	Low	Median
Register and Login?	●	●	○

Table 1: Comparison between different hosts and their supported apps. "●" represents full support; "◑" represents partial support; "○" represents no support.

3 Security Mechanisms in Super Apps

To protect the resources hosted by super apps, a series of mechanisms have been implemented to maintain the integrity, confidentiality, and availability of sensitive informa-

	Mobile OS (Native Apps)	Web Browsers (Web Apps)	Super Apps (Miniapps)
Deployed at Superapps' Frontends			
(S1) Permission Mechanism [27]	●	○	●
(S2) Sandboxing [9]	●	●	●
(S3) API Restriction [8, 39]	◑	○	●
(S4) Cross-miniapp Allowlisting [63]	◑	◑	◑
(S5) Designated Distribution Channel [12]	◑	○	●
Deployed at Superapps' Backends			
(S6) Domain Allowlisting [58, 39]	○	○	●
(S7) Secure Communication [58]	◑	○	●
(S8) Role-Based Access Control [54]	●	●	●
(S9) Data Encryption [66, 44]	◑	◑	●
(S10) Token-based Services Access [44]	◑	◑	●
(S11) User Token Isolation [50, 53]	◑	◑	●
(S12) Code Vetting [32]	●	○	●
(S13) Account Protection [32]	○	○	●

Table 2: Comparison of Security Mechanism. "●" represents full support; "◑" represents partial support; "○" represents no support.

tion and services. Although the exact security mechanisms may differ based on the specific implementation and platform, they fall into two categories: front-end security measures and back-end security measures, as shown in Figure 3:

3.1 Security Mechanisms at Front-ends

Front-end security mechanisms primarily focus on safeguarding the local resources and services managed by super apps at users' devices.

(S1) Permission Mechanism [27]. Super apps typically build a strong permission mechanisms to ensure that the sensitive data (e.g., phone number, location) on users' local devices are only accessible after the users acknowledge and approve the resource access. This mechanism resembles the permission mechanisms deployed by mobile operating systems. For example, WECHAT has a comprehensive permission system covering a wide range of resources including user account information, social network information (e.g., group chat information), and device resources (e.g., Bluetooth). The miniapps must prompt the users of the specific type of resources required by them prior to accessing the resources. If the users reject the resource access, the miniapps cannot access them. As shown in Table 2, this is very similar to the mobile permission mechanisms.

(S2) Sandboxing [9]. Similar to other apps that build containers to run third-party applications or OSs, super apps create controlled environments known as sandboxes as well, where each miniapp operates in independent storage spaces. This isolation ensures that the behavior of miniapps cannot directly interfere with other apps or the underlying system (e.g., writing or reading files outside the designated spaces), especially when different miniapps are given different sets of permissions (e.g., certain mini-apps may possess the privilege to access user data, while others may not), because the access to highly privileged data should not be permitted for miniapps with fewer privileges. For instance, miniapps can only access files via specific URL schemes (e.g., `wxfile` for WECHAT), which are only stored *inside* their own sandbox by super apps. As shown in Table 2, mobile OSs and Web browsers have similar protections.

(S3) API Restriction [39]. While the super apps customize the execution engine to support miniapps, it is natural for the super apps to restrict or remove APIs considered sensitive, dangerous, or vulnerable, to refrain miniapps from using them. For instance, when a miniapp invokes the payment related APIs, the super apps may check whether the developers of the miniapp have already submitted required business certificates to the platforms. Instead, developers failing to do so may not invoke the API. Meanwhile, there are "undocumented" APIs that are only supposed to be used by miniapps from the super app themselves, such as `searchContacts`, an API used to access the friend list of a WECHAT user, which is only supposed to be accessible by miniapps from TENCENT. Super apps may verify the author of miniapps prior to the invocation to make sure the API is only accessed by expected parties.

Aside from verifying the identity, another set of API is strictly prohibited by design to prevent exploitation. A typical example of this is that `eval()` is not accessible by miniapps, as it executes JavaScript code as strings and may be prone to injection attacks. Also, the rendering of miniapp is protected by preventing developers from accessing APIs to manipulate the Document Object Model (DOM) tree. In these super apps, there are no APIs manipulating DOM tree nodes to prevent miniapps from dynamically changing behavior of rendering under the similar concern of fighting against malware.

(S4) Cross-miniapp Allowlisting [63]. In order to facilitate information sharing among collaborating miniapps, super apps employ a mechanism known as cross-miniapp redirection. This mechanism enables a miniapp to redirect to another miniapp with a payload, similar to the Intent mechanism in Android. However, there is a concern that malicious miniapps may exploit this redirection and message sharing functionality to inject carefully crafted payloads to arbitrary miniapps. Consequently, super app platforms such as ALIPAY have implemented an allowlist-based mechanism for miniapps to filter payload sent from unexpected miniapps.

(S5) Designated Distribution Channel [12]. Super apps typically employ a strict policy to prevent hot updates. The concept of hot update refers to the ability to update the code or functionality of an app without requiring the user to manually re-install updated packages from app stores. The hot update is prohibited in super apps because the super app needs to vet all miniapp codes before they are released to

the ecosystem in case of malware. However, the hot update ability grants malicious developers capability of transforming a vetted (benign) miniapp into malware via cloud-transmitted hot update code. Hence, the ability to side-load miniapps or execute code as cloud-delivered payloads (e.g., using JavaScript interpreter libraries) are prohibited.

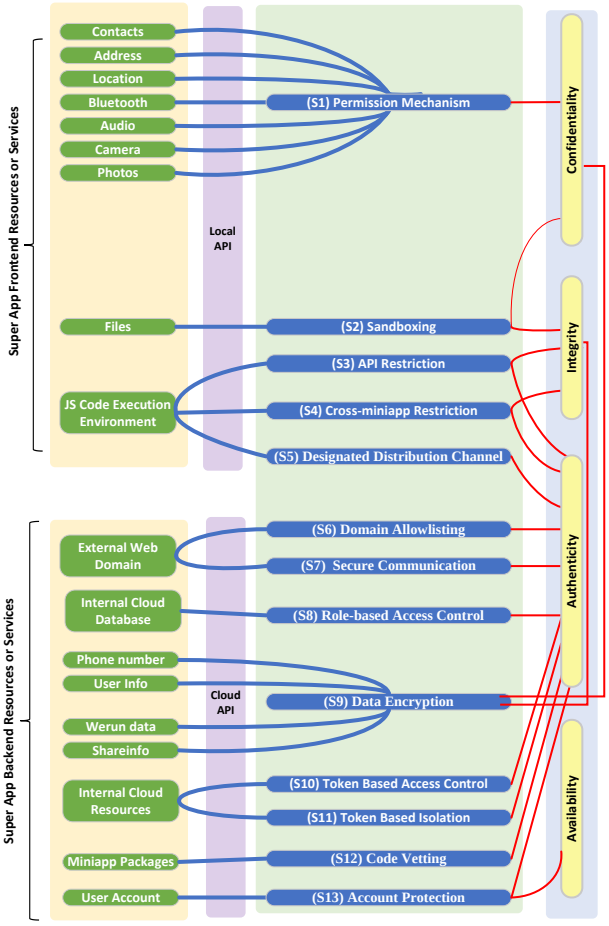

Figure 3: Relationship between the resources, security measures and the security properties

3.2 Security Measures at Back-ends

On the other hand, back-end security measures pertain to the security controls and practices employed within the super app's server infrastructure and backend systems. These measures aim to fortify the app's core cloud functionalities, databases, submitted code, and any other components that reside on the server-side.

(S6) Domain Allowlisting [58, 39]. Super apps employ a domain allowlist mechanism to effectively regulate web page access, ensuring that users are protected from downloading files or accessing content from unknown or untrusted sources. For example, in WeChat, miniapps are required to use the `wx.request` function to access specific URLs. However, before granting access, WeChat's vetting process verifies the requested URL against a domain allowlist. This domain allowlist is configured by developers and vetted by the platform before the miniapp is submitted, and can be viewed by users by opening the "`About`" page of miniapps. This verification ensures that the requested content originates from a trusted source that meets the predefined security policy. As shown Table 2, we have not observed similar protections in mobile OSs and web browsers.

(S7) Mandatory Secure Communication [58]. Super apps enforce the implementation of secure communication protocols, notably HTTPS (HTTP over SSL/TLS), to establish encrypted data transmission between the miniapp and the server. For example, miniapps in WeChat can only use either HTTPS or WebSocket Secure (WSS) protocol, and requests sent via HTTP will be blocked by the super apps after they are released (while developers are allowed to test their communication with HTTP in development tools).

(S8) Role-Based Access Control (RBAC) [54]. The super apps incorporate Role-Based Access Control (RBAC) mechanisms on cloud services provided by them such as cloud database to prevent the data from being accessed or overwritten by unexpected parties. While the developers have access to data stored in miniapp cloud database provided by super apps, the privileges granted to administrators and ordinary developers are different. Hence, this helps prevent unauthorized access to sensitive features and data within the miniapp.

(S9) Data Encryption [66, 44]. Miniapps hosted within super apps like WeChat and Baidu often handle sensitive resources, such as user phone numbers and billing addresses. To prevent these sensitive data transmitted between miniapp front-ends and back-ends from being captured or manipulated by attackers, cryptographic protocols are employed to ensure the security of this data, relying on a master key known as the "AppSecret". This key is generated by super apps, shown to developers, and supposed to be properly stored in developers' back-ends, as it is used by the super apps to verify a back-end's identity. To provide more specific details, the data (such as phone numbers) is encrypted using a key called the "session key", and the resulting cipher is sent to the miniapp's back-end. The miniapp's back-end can then utilize the "AppSecret" to request the "session key" from the super apps, allowing it to decrypt the data.

(S10) Token-based Services Access [44]. Super apps often provide OAuth 2.0-alike protocol as a standard authentication and authorization framework for accessing their services. The protocol enables secure and delegated access to protected resources without the need to share sensitive credentials, such as usernames and passwords. After successful

authentication and authorization, the super app provides access tokens that serve as credentials for the miniapp to access specific services or resources. These access tokens are typically short-lived and can be scoped to restrict the permissions granted to the miniapp, ensuring that it only has access to the required resources.

(S11) User Token Isolation [50, 53]. As mentioned in token-based service access, the cloud data access are implemented based on tokens to avoid sharing credentials via cloud communication. To prevent the user tokens from being misused or intercepted, the tokens used to access data are isolated from each miniapps. For example, when accessing Alice's user information with the data token (i.e., `DT`) under miniapp A, the back-end of miniapp A needs to submit `DT`, miniapp's ID, along with the miniapp's master key "AppSecret". The server checks first if the ID and master key match to ensure that the request is from an authentic source associated with miniapp A, and then send Alice's data associated with `DT`. However, even if this token is obtained by a malicious miniapp B, miniapp B cannot get the user data with `DT`, as the `DT` is only associated with miniapp A and isolated from miniapp B. As depicted in Table 2, many mobile operating systems and web browsers share similar protections to those of **S10** and **S11**. However, these protections may not always come as standard features; they're often not a mandatory part of the system.

(S12) Code Vetting [32]. Super app platforms typically enforce stringent code verification and app review processes to ensure that miniapps hosted on their platforms meet certain security standards whenever miniapp packages are submitted to the platform via cloud to be released to the miniapp market. This helps mitigate the risks associated with malicious or vulnerable apps being distributed to users (e.g., miniapps with leaked AppSecret). This approach is akin to the code verification and review practices implemented by mobile App Markets (e.g., Apple AppStore and Google Play).

(S13) Account Protection [32]. Super apps implement various mechanisms to protect user accounts from being stolen, as well as vetting mechanisms on the accounts to prevent malicious developers from obtaining an account at a low cost to exploit the platform. Upon registration, users are required to provide personal information such as resident ID and phone numbers to the platform. When registering as developers, more information may be required, such as commercial license for enterprise developers. Then, whenever a user uses super apps, or a developer accesses the developer tool, verification is enforced via log in. Meanwhile, super apps have mechanisms in place to suspend or ban user accounts if suspicious or malicious activity is detected, such as modifying network traffic using software or engaging in fraudulent behavior.

3.3 Summary of Security Assumption

The security mechanisms enforced in super apps are designed based on different security assumptions which can be categorized into three categories, including the "Adopted" assumption, "Isolated" assumption, and "Vetted" assumption.

(A1) The "Adopted" Assumption. As the super apps share similarity with mobile and web computing paradigms, four out of 13 mechanisms (**S1**, **S2**, **S7**, and **S8**) introduced in this section are adopted from traditional paradigms with little customization. As these mechanisms are adopted from sophisticated paradigms directly from traditional platforms, the security assumption is assumed to be as strong as traditional platforms. For instance, while permission mechanisms are built upon Android system to manage super app specific data, the mechanisms to require users to acknowledge and approve data access is essentially the same as permission mechanisms in Android systems. Similarly, the idea of RBAC and Sandboxing are adopted from OS resource management model, and Secure Communication (HTTPS requirement) is directly adopted from web computing.

(A2) The "Isolated" Assumption. Super apps incorporate various mechanisms to ensure that resource access from third parties, particularly untrusted or non-authentic entities, is appropriately isolated. Among the 13 mechanisms implemented, four are associated with this assumption (**S3**, **S9**, **S10**, and **S11**). These mechanisms include isolating privileged APIs to prevent non-super-app miniapp developers from accessing them (**S3** API restriction). Additionally, the data access is isolated from non-original miniapp developers through the enforcement of master keys and tokens at a per-miniapp level (**S9**, **S10**, and **S11**). By enforcing these mechanisms and ensuring that each miniapp can only access its own resources, the security of data access is intended to be effectively safeguarded.

(A3) The "Vetted" Assumption. Since super apps now feature exclusive miniapp market where all executable miniapps have to be submitted and vetted by the super apps, it is assumed that the super app platform will vet and identify potential malicious behaviors of third-party miniapps. These vetting covers the rest five out of 13 mechanisms (**S4**, **S5**, **S6**, **S12**, and **S13**), including vetting of miniapp communication between front-ends (**S4**) and back-ends (**S6**), control on miniapp releasing and distribution (**S5** and **S12**), as well as vetting and supervision on developer accounts (**S13**).

4 Security Threats and Root Causes

4.1 Threats at Front-ends

(T1) Data Leakage Due to Flawed Permission [46]. While super apps build permission mechanisms upon underlying systems, the permission enforced by super apps are not necessarily a strict superset of the underlying system. More specifically, there are inconsistencies on permission required between miniapp APIs and system APIs. For instance, Lu *et al.* [46] discovered that while Android system requires applications to be granted location permission (`ACCESS_COARSE_LOCATION` and `ACCESS_FINE_LOCATION`), accessing Wi-Fi List via `wx.getWifiList` in WeChat did not require similar super app permission. While this vulnerability has been fixed by mainstream super app platforms by enforcing miniapps to be granted `location` permission, the mismatch between permission management and super apps still may pose a threat, as the underlying system's permission management is constantly being updated, creating potential mismatches.

(T2) Data Leakage Due to Cross-platform Vulnerabilities [38]. Supporting multiple platforms (such as Android and iOS) introduces challenges in maintaining consistent security measures, as divergent security policies, discrepancies of implementation, and inconsistent security updates across platforms can create gaps that attackers may exploit. For instance, sensor APIs like accelerometer, compass, and gyroscope are typically limited to mobile platforms such as Android and iOS. However, the difference in APIs may go beyond compatibility, introducing security vulnerabilities. One such example is the `makeBluetoothPair` API, which facilitates Bluetooth device authentication. The absence of this API on platforms like iOS prevents Bluetooth devices from distinguishing between trusted and untrusted devices, thereby opening the door for potential Man-in-the-Middle attacks.

(T3) Data Leakage via Hidden APIs [39]. While abundant documentation have been provided to developers about functionalities and APIs, not all APIs are documented. These "hidden" APIs are designed for internal use (e.g., miniapp developers from Tencent only in WeChat), but may potentially enable miniapps to bypass restrictions and gain unauthorized access. For example, in WeChat, a malicious miniapp can exploit the hidden API to access arbitrary malicious contents without being detected by the super apps or secretly download and install harmful Android apps. These malicious apps have the potential to steal sensitive user information. The usage of hidden APIs such as "`captureScreen`" to capture screenshots, "`getLocalPhoneNumber`" to steal the user's phone number, and "`searchContacts`" to extract the user's contact information.

(T4) Data Injection via Cross-miniapp Channel [63]. Miniapps, due to their limited size, often rely on communication with other miniapps to accomplish complex tasks. For instance, a shopping miniapp may need to exchange information, such as the order ID and price, with a payment miniapp to facilitate a purchase. As such, this cross-communication becomes a vital means for miniapps to enhance their functionalities and collaborations. However, if a receiver miniapp fails to perform proper checks on the sender's appId, it can result in a new form of attack called Cross-Miniapp Request Forgery (CMRF), where a malicious miniapp can send arbitrary payloads via cross-miniapp channel. These payloads may be fabricated order data to trigger shopping-for-free, or login data to intrude into an arbitrary user's account.

(T5) Vetting Bypass via Post-vetting Hot Update [12]. The use of hot updates by malicious miniapps to bypass vetting in super apps poses a significant security concern. Hot updates refer to the ability to dynamically update or modify the code of an application without requiring a full version update or undergoing the typical vetting and approval processes. While certain known APIs like `eval()` and the use of `VM` library [59] is prohibited in super app platforms to prevent dynamically loaded or interpreted code, the malicious actors may still exploit hot update restriction by implementing and integrating an interpreter on their own [43], as JavaScript is essentially an interpreted language, and the code does not have to be compiled.

4.2 Threats at Back-ends

(T6) Vetting Bypass via Identity Confusion [64]. Security concerns arise in the super app ecosystem regarding access control for privileged APIs. Existing super apps lack atomic identities, violating the principle of least privilege. For example, Zhang *et al.* [64] discovered that an unprivileged miniapp may contain a privileged web domain, a privileged miniapp ID may include unprivileged third-party web domains, or an unprivileged miniapp may gain privileged capabilities. This situation creates a vulnerability known as "identity confusion". Adversaries can exploit this vulnerability by manipulating their own identity to masquerade as entities with granted permissions, thereby confusing the super app during the identity verification process. This identity confusion vulnerability, if present in an app-in-app ecosystem, poses a significant security risk that needs to be addressed.

(T7) Data Leakage Due to Key-Misuse [66]. As discussed in **S9**, robust cryptographic protocols rely on a master key called the "AppSecret" generated and managed by the super app to ensure the security of data, which is essential for au-

Security Mechanism Analysis								Security Threat Analysis							Security Impact Analysis			
Security Mechanism		At			Assumption			Threat ID		Root Cause				Privileged Access	Vetting Bypass	Data Issue	
		#	F	B	A	I	V			Comp.	Impl.	Trust	Vetting			Injection	Leakage
S1	Permission mechanism	①	✔		✔			T1	Flawed Permission	Permission Management				Data			✔
S2	Sandboxing	①	✔		✔			T2	Cross-platform Vulnerability	Resource Management				Data			✔
S3	API Restriction	②	✔			✔		T3	Hidden API Access	Missing				Service			✔
S4	Cross-miniapp Allowlisting	③	✔				✔	T4	Cross-miniapp Injection		Miniapp			Miniapp		✔	
S5	Designated Distribution Channel	④	✔				✔	T5	Post-vetting Hot Update				Post-vetting	Service	✔		
S6	Domain Allowlisting	⑤		✔			✔	T6	Identity Confusion		Miniapp			Service	✔		
S7	Secure Communication	⑤		✔	✔			-			-			-		-	
S8	Role Based Access Control	⑥		✔	✔			-			-			-		-	
S9	Data Encryption	⑥		✔		✔		T7	Master Key Misuse		Developer			Data			✔
S10	Token-based Access Control	⑦		✔		✔		T8	Abused API Token		Developer			Data			✔
S11	User Token Isolation	⑦		✔		✔		T9	Weak Token Isolation	Weak				Data			✔
S12	Code Vetting	⑧		✔			✔	T10	Evasive Malware				Intra-vetting	Data	✔		
S13	Account Protection	⑨		✔			✔	-			-			-		-	

Explaination of the Abbreviations									
Mechanism Enforcement (§3.1, §3.2)			Security Assumption (§3.3)			Root Cause Analysis (§4.3)			
#	F	B	A	I	V	Comp.	Impl.	Trust	Vetting
ID of edge in Figure 1	Enforced At Front-end	Enforced At Back-end	"Adopted" Assumption	"Isolated" Assumption	"Vetted" Assumption	Compatibility Issue	Implementation Issue	Trust Model Issue	Vetting Issue

Table 3: Summary of protection mechanisms, security assumptions, and threats due to violation of assumptions. The rows are colored based on categories of security assumption.

thenticating miniapps and securely transmitting sensitive information. However, if this key is leaked in the front-end of the miniapps, it can have severe consequences for both developers and users. Upon extracting the miniapp packages and harvesting these AppSecrets, attackers can now obtain the encryption key and manipulate the encrypted data for harmful activities, including account hijacking (unauthorized access to others' accounts), promotion abuse (exploiting promotional activities for personal gain), and service theft (utilizing others' paid services without authorization).

(T8) Data Leakage Due to Abused Token [66]. Super apps and web apps alike use OAuth 2.0 to protect user data accessed by miniapps. On top of that, API tokens are also generated from AppSecrets for developers to invoke various cloud services such as Optical Character Recognition (OCR). However, as the AppSecrets and tokens may be leaked by developers to miniapp front-ends, which can be intercepted and abused by attackers, these cloud data and services may be provided to unexpected malicious parties. Moreover, many cloud services provided by super app platforms are not free, henceforth the abuse of API tokens may further inflict financial losses to victim developers who leaked the keys.

(T9) Data Leakage Due to Weak Token Isolation. The tokens used to access user data are designed as an access control mechanism as the token is isolated at per-user-per-miniapp basis, which indicates that a token is only associated with a specific user under a specific miniapp, and this is why the AppSecret of miniapps has to be submitted to super app cloud to get the user data. However, the implementation of isolation is not the same across platforms. For example, in super app DINGTALK, upon fetching a data token from a user, the token can be used by different miniapps under the same developer. However, this poses a threat of colluding attack, as a low-privilege miniapp may receive data token from a high-privilege miniapp, but still can fetch the user data protected by the data token from the platform.

(T10) Vetting Bypass by Evasive Malware. With the vetting mechanism enforced, malicious miniapps seek to bypass the vetting, which results in an endless arms race between vetting mechanisms and evasive malware. For example, malicious developers may obfuscate their code to make it harder for the vetting process to detect their true intentions, or deploy dynamic loading techniques. Moreover, as the super apps involve social network among users, it has become a favorable target for launching phishing and scam malware to be rapidly distributed among users' social networks. Beyond that, there are more types of vaguely-defined malware, such as miniapps cheating the popularity by forcing users to forward miniapps to friends, because miniapps with advertisement embedded receive more profit based on higher count of clicks. These cheating behaviors include forcing users

to share, faking as authentic miniapps, or acting as miniapp portal, because user will then use the portal miniapp to enter all other miniapps, where portal miniapps automatically get clicked and thus receive more profit from advertisers.

4.3 Root Cause Analysis

In this paper, we identified 10 threats that violated the security assumptions on the security mechanisms enforced by super app platforms, which can be categorized into compatibility, implementation, trust mode, and vetting issue as follows.

Compatibility Issue. As super apps execute upon different operating systems (Android, Windows, iOS, etc.), while the super apps may have implemented security mechanisms like sandboxing, the security property still are affected by how the underlying systems manage the accessible resources. Thus, issues in permission mechanisms (**T1**) and resource management (**T2**) may still create vulnerabilities in super apps. Hence, the super apps need to pay attention to discrepancies across underlying systems, and to implement additional protection if needed.

Implementation Issue. While super apps implemented various isolation mechanisms to perform access control on miniapps accessing resources, the implementation may suffer from missing (**T3**, missing or programmatically API isolation) or weak (**T11**, weak isolation of tokens) spots that eventually weakens or even voids the mechanism. Hence, super app platforms need to ensure that the implementation is following standardized practices and cover comprehensive aspects of the ecosystem.

Trust Model Issue. While security mechanisms have been implemented to enable access control on untrusted parties, some security assumptions on what can be trusted still contain flaws. For example, while developer identities are vetted by the super app platforms, the platforms still should not trust the developers too much that they will never leak sensitive tokens, as the developers may not be aware of the importance of leak these tokens by mistake (**S9**, **S10**). Also, the super apps should not place excessive trust on miniapps even if they are vetted, as they may contain vulnerabilities not covered by vetting (**T4**, **T6**).

Vetting Issue. The other aspect of the trust issue is that even though super apps adopt strict vetting mechanism and controls the distribution of miniapps, the vetted miniapps still should not be overly trusted. During vetting, malware may actively work against the vetting mechanism, where vetting may not discover all malicious behaviors (**T12**). After vetting, malware still may be able to dynamically load malicious code without being identified by the platforms (**T5**).

	Mobile OS (Native Apps)	Browsers (Web Apps)	Super Apps (Miniapps)
Threats against Frontends			
(T1) Flawed Permission	●	○	●
(T2) Cross-platform Vulnerability	○	○	●
(T3) Hidden API Access	●	●	●
(T4) Cross-miniapp Injection	●	●	●
(T5) Post-vetting Hot Update	◑	○	●
Threats against Backends			
(T6) Identity Confusion	○	○	●
(T7) MasterKey Misuse	◑	◑	●
(T8) Abused Token	◑	◑	●
(T9) Weak Token Isolation	◑	◑	●
(T10) Evasive Malware	●	○	●

Table 4: Comparison of Threats. ●: the threat works against the paradigm; ◑: the threat works against some of the implementations; ○: the threat has not been discovered in the paradigm.

5 Lessons Learned

In this paper, we systematically investigated super app platforms, summarized 13 protection mechanisms (5 at front-end, 8 at back-end), and identified 10 security threats due to violation of three families of security assumptions. By examining the observations summarized in Table 3, and comparing the threats against traditional paradigms in Table 4, in this section, we present 5 lessons learned from the threats in super apps as follows.

5.1 The Inherited Debt of Super Apps

(L1) The old is new again. As shown in Table 4, a majority of threats in super apps can be found in traditional paradigms, such as hidden APIs threats which broadly existed in mobile and web apps. However, these "old" problems have become new again under the context of super app platforms. For instance, while cross-app injection existed in Android systems, the systems do not necessarily have to be responsible for the attack, as the attack happens between two apps that are not hosted by the systems themselves (they could be downloaded from third-party websites), and thus it is partly the users' responsibility to beware of unknown apps. Hence, the Android system informs users of apps from unknown sources but does not restrict their installation. Super apps, however, exclusively host the miniapp packages, and thus have more liability on miniapps targeting this vulnerability, especially when these miniapps access privacy-sensitive data hosted by super apps.

(L2) Every coin has two sides. While super apps have combined the benefits of executing versatile web apps with powerful native functionalities, super apps also face threats generated from both paradigms. These include problems derived from mobile paradigms (e.g., permission, cross-app channel,

hot update, etc.) and from web paradigms (token-related vulnerabilities, API restrictions, and filterings). Hence, the super app platforms have to prudently design the protection mechanism to safeguard the user data stored in the ecosystem.

5.2 Violation of Security Assumption

As shown in Table 3, we have summarized all the security mechanisms, assumptions, and threats due to the violation of these assumptions. When looking into mechanisms based on "Adopted", "Isolated", and "Vetted" assumptions, we may draw different observations on the root cause of the threats.

(L3) Compatibility is an issue. Mechanisms under the Adopted assumption suffer from compatibility issues and trust model change. This includes permission mechanisms which suffer from super app's inconsistency on permission set compared with the underlying system, and sandboxing issues suffered from different mechanisms across platforms (especially between super apps on desktop systems and mobile systems). Hence, the super apps have to check whether the protection mechanisms are implemented to offer the same level of protection on all potential systems the super apps will be executed on.

(L4) Single leak sinks the ship. Mechanisms under the isolated assumption suffer from weak or missing implementation, which voids the isolation mechanism. We have identified four isolation mechanisms, with which one is enforced on accessible API and three is enforced on cloud communication. From the table, we observe that two threats (**T3**, **T9**) are due to super app's implementation. In terms of the threats due to careless developers, they are mainly because developers may leak certain sensitive tokens to front-ends, which can be utilized by attackers to fake the identities (**T7**, **T8**). Hence, the developers have to be properly educated, and mandatory detection of such leakage should be enforced.

(L5) Vetting is not omnipotent. Mechanisms under vetted assumption still need to be constantly supervised for malicious behaviors. We have identified 5 vetting-related mechanisms, where 4 are associated with threats. These threats involve both trust issues and vetting limitations. The major challenge super apps face is that malware actively acts against the vetting mechanism (**T12**), whereas vetting may not cover all potential vulnerabilities or malicious behaviors. Failure to realize so may result in trust issue, as super apps may trust vetted miniapps' behaviors, even though vetting may not reveal the capability for confusion or injection (**T4**, **T6**). Even if a miniapp contains no malicious code during vetting, it still may be updated after being vetted if vulnerabilities allow them to hot-update code (**T5**). Hence, additional mechanisms to supervise the behavior of vetted miniapp need to be enforced.

6 Trade-offs and Open Problems

6.1 Trade-offs

While third-party miniapps are allowed into super app ecosystems, it may still raise contradiction between the interest of super apps and users, especially when the super apps have collected a large amount of user data and open the access to third-party miniapps. While access control is enforced, it still may raise concern due to the *"data sovereignty"*, where super apps may have sole rights to collect, control, dispose, and even delete the data and accounts of super app users. In this section, we discuss three trade-offs (**O**) derived from this dark side of super apps, on privacy issue accompanied by convenience, contradiction between user privacy and the platform's monetization utilizing privacy-sensitive data, and the debate between security and usability. In the end, we will summarize open problems (**P**) that may be addressed by future works based on our findings.

(O1) Privacy and Convenience. While users may experience convenience on seamless and convenient services such as food ordering by allowing miniapps to access personal information such as phone numbers to register accounts and place orders, risks occur when the privacy data of users are shared between miniapps and super apps. Nevertheless, the super app platforms need to seek a trade-off between user convenience and privacy protection, especially when these super apps involve data collected from millions of users.

(O2) Privacy and Monetization. Super apps may be tempted to monetize user data by sharing it with third parties for targeted advertising or other purposes. However, this poses privacy concerns, as users may not be aware of how their data is being used or whether they have control over data sharing. To mitigate these concerns, super apps should adopt transparent data practices, provide clear privacy policies, offer robust data protection measures, and give users control over their data, including the ability to opt out of data sharing or targeted advertising.

(O3) Security and Usability. Balancing security and usability is crucial in the super app ecosystem to optimize user experience while safeguarding user data and privacy. However, this task can be challenging as prioritizing security measures may hinder user experiences. For instance, users may opt for Bluetooth Low Energy (BLE) Secure Connections or Bluetooth Secure Simple Pairing (SSP) protocols to ensure secure authentication and pairing of devices. However, the process of pairing two devices together requires users to interact with the devices, such as entering passkeys and confirming the connection. Imposing these procedures can refrain users from using the super app altogether due to the added inconvenience.

6.2 Open Problems

(P1) Security compliance analysis. In this paper, we have illustrated threats due to implementation issues and compatibility issues (**T1**, **T2**, **T3**, **T9**). As super apps may implement mechanisms differently, it is vital to analyze the security compliance, especially on evaluating whether the super apps offer protection mechanisms at least the same or higher level of security compared with underlying systems. Otherwise, the super app could become a favourable target for attackers to exploit users' devices and cloud-hosted data. Future works could include systematic analysis on existing platforms, and automatic tools for super app and developers to uncover potential vulnerabilities due to such discrepancy.

(P2) Security mechanism standardization. We have revealed that the difference of implementation across super app platforms may introduce vulnerabilities breaking access control (**T3**), or cause one platform to be more vulnerable than the others (**T9**). Future works may include a study on measuring feasible security mechanisms for super app platforms, and a standardized security protection solution that can be deployed by all super app platforms to ensure that the protection implementations are equally stringent.

(P3) Miniapp developer education. Besides the problems due to implementation of super apps, we also found that developers may leak essential but sensitive information to undesired parties (**T7**, **T8**). While the developers are not supposed to be all experts in miniapp security configuration, these developers need to be properly educated on recommended and prohibited practices. Therefore, super apps need to provide more developer-oriented highlight on security configuration in documentations. Future works may include automatic analysis and detection on leakage of privacy- and security-sensitive data that can be directly deployed at developers' ends or the vetting process to prevent vulnerable miniapps from being released to the ecosystem.

(P4) Semantic-aware miniapp vetting. In this paper, we discover that the current vetting mechanisms still face challenges and limitations despite the strict control on the distribution of miniapps (**T4**, **T5**, **T6**, **T10**). Future work may be directed to modeling and formalizing super app specific miniapp malware, and automatic, semantic-aware detection techniques deployable by super apps to significantly thwart these malware from being distributed into the ecosystem.

7 Related Work

Miniapp Security. The security of miniapps within super apps is currently receiving significant attention. For instance, Lu *et al.* [46] uncovered resource management vulnerabilities that allow attackers to steal sensitive user information. Zhang *et al.* [65] developed MiniCrawler to analyze security practices, such as obfuscation usage and security-related API invocations of miniapps. More recently, Zhang *et al.* [64] discovered identity confusion vulnerabilities enabling attackers to exploit high-privileged capabilities. Additionally, Yang *et al.* [63] investigated vulnerabilities cross-miniapp channel in platforms like WeChat and Baidu. Wang *et al.* [57] proposed WeDetector, and identified 11 bugs from 25 real-world WeChat miniapps. Wang *et al.* [37] introduce TaintMini, a comprehensive taint analysis framework to detect collusion attacks among miniapps. Zhang *et al.* [66] investigated the misuse and consequences of cryptographic keys, the "AppSecret", within miniapps. Wang *et al.* [39] discovered previously undisclosed APIs offered by super apps and illustrated their potential for exploitation. Wang *et al.* [38] systematically identified inconsistencies and derived vulnerabilities in WeChat APIs across platforms. In contrast to prior efforts, our work represents the first systematic endeavor to systematize the security mechanisms, threats, and root cause of super apps.

App Ecosystem Security. In recent years, alongside mini-apps, various techniques have emerged to facilitate the development of a streamlined app ecosystem. For instance, Google Instant Apps, as mentioned by Aonzo *et al.* [35], have been introduced to support lightweight applications. Notably, Aonzo *et al.* [35] and Tang *et al.* [51] highlight the vulnerability of Google Instant Apps to be utilized in password-stealing attacks. Super apps are evolved from web browsers. There is a large body of research focusing on browser security such as web extensions [60, 48, 40, 56], which are closely related to our study. For example, SABRE [42] uses in-browser information-flow tracking to detects malicious browser extensions that leaks sensitive information. Hulk [45] dynamically detects malicious browser extensions by monitoring their execution. Expector [60] inspects and identifies browser extensions that involve the advertisements, and then detects the malicious ones.

8 Conclusion

In conclusion, mobile supper apps have revolutionized the mobile app landscape by enabling third-party developers to deploy add-ons within these platforms. This study highlights the security measures implemented by super apps, including permissions, sandboxing, and encryption, to ensure data protection and system integrity. However, challenges such as phishing attacks and balancing revenue generation with user experience remain. User awareness and ongoing research are crucial for addressing these challenges and advancing the super app paradigm. Overall, the super app paradigm offers enhanced functionality and improved user experiences, but further efforts are needed to refine security models and mitigate potential threats.

References

[1] 11+ spectacular kuaishou statistics for 2023. https://webtribunal.net/blog/kuaishou-statistics/#gref.

[2] 74.7 million – the number of users of messaging app zalo in vietnam. https://restofworld.org/stat-of-the-day/messaging-vietnam-zalo/.

[3] Airtel surpasses 10 million unique customers on its 5g network. https://www.airtel.in/press-release/02-2023/airtel-surpasses-10-million-unique-customers-on-its-5g-network.

[4] Alibaba's dingtalk records 600 million users, works to accelerate monetization. https://technode.com/2022/12/29/alibabas-dingtalk-records-600-million-users-works-to-accelerate-monetization/.

[5] Average number of maya users in the philippines in 2020 and 2021. https://www.statista.com/statistics/1279190/number-of-paymaya-users-philippines/.

[6] Binance statistics 2023. https://www.finder.com/binance-statistics.

[7] Careem super app reaches 48 million users. https://waya.media/careem-super-app-reaches-48-million-users/.

[8] Dom tree isolation. https://developers.weixin.qq.com/miniprogram/en/dev/framework/quickstart/#Differences-Between-Mini-Program-Development-and-Web-Development.

[9] File system of wechat. https://developers.weixin.qq.com/minigame/en/dev/guide/base-ability/file-system.html.

[10] Further expanding the paypay ecosystem - future envisioned by z financial. https://www.z-holdings.co.jp/en/strategy/16.

[11] Grab reports first quarter 2023 results. businesswire.com/news/home/20230518005395/en/Grab-Reports-First-Quarter-2023-Results.

[12] Hot update restriction. https://developers.weixin.qq.com/miniprogram/en/dev/framework/runtime/js-support.html.

[13] Jumia reports first quarter 2023 results. https://www.accesswire.com/756438/Jumia-Reports-First-Quarter-2023-Results.

[14] Kakao talk revenue and growth statistics (2023). https://www.usesignhouse.com/blog/kakao-talk-stats.

[15] Line revenue and growth statistics (2023). https://www.usesignhouse.com/blog/line-stats.

[16] M-pesa customer numbers from 2017 to 2022. https://www.statista.com/statistics/1139190/m-pesa-customer-numbers/.

[17] Meituan faces challenge from alipay on its home turf. https://technode.com/2020/04/01/meituan-faces-challenge-from-alipay-on-its-home-turf/.

[18] Mercado pago reaches almost 20 million active users, and debuts in the insurance market. https://labsnews.com/en/news/business/mercado-libre-financial-services-arm-mercado-pago-debuts-in-the-insurance-market/.

[19] Momo statistics and user count. https://expandedramblings.com/index.php/momo-statistics/.

[20] Monthly active users of tencent's wecom* app in china from december 2016 to march 2021. https://www.statista.com/statistics/1242600/china-monthly-active-users-of-wecom/.

[21] New entrants are challenging meituan's grip on china's booming local life services sector. https://technode.com/2023/05/11/new-entrants-are-challenging-meituans-grip-on-chinas-booming-local-life-services-sector/.

[22] Number of global downloads of bytedance's toutiao as of february 2023, by country or region. https://www.statista.com/statistics/1343086/bytedance-toutiao-download-number-by-markets.

[23] Number of monthly active qq users from 4th quarter 2019 to 1st quarter 2023. https://www.statista.com/statistics/1318070/china-tencent-number-of-monthly-active-accounts-of-qq.

[24] Number of monthly active users of baidu app from march 2020 to march 2023. https://www.stat

ista.com/statistics/1315592/baidu-app-monthly-active-users/.

[25] Number of monthly active wechat users from 2nd quarter 2011 to 1st quarter 2023. https://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/.

[26] Paytm statistics and user count. https://expandedramblings.com/index.php/paytm-statistics-facts/.

[27] Scope list of wechat. https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/authorize.html.

[28] Tiktok users worldwide (2020-2025). https://www.insiderintelligence.com/charts/global-tiktok-user-stats/.

[29] Toss statistics and user count. https://expandedramblings.com/index.php/toss/.

[30] Uber claims top spot in indian ride-hailing market. https://techcrunch.com/2020/02/08/uber-claims-top-spot-in-indian-ride-hailing-market/.

[31] Vodacom super-app vodapay now has 3.3 million registered users. https://techfinancials.co.za/2023/05/15/vodacom-super-app-vodapay-now-has-3-3-million-registered-users/.

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

[33] Yandex statistics and user count. https://expandedramblings.com/index.php/yandex-statistics/.

[34] Decoded: WeChat Mini Programs For Cultural Destinations. https://jingculturecommerce.com/decoded-wechat-mini-programs-for-cultural-destinations-part-one/, 03 2020. (Accessed on 04/26/2022).

[35] S. Aonzo, A. Merlo, G. Tavella, and Y. Fratantonio. Phishing attacks on modern android. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1788–1801, 2018.

[36] A. Asmakov. Elon musk wants twitter to be wechat-style 'super app' with payments. https://decrypt.co/100732/elon-musk-wants-twitter-be-wechat-style-super-app-payments. (Accessed on 05/19/2022).

[37] W. Chao, Z. Yue, and L. Zhiqiang. Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis. In *ICSE*.

[38] W. Chao, Y. Zhang, and Z. Lin. One size does not fit all: Uncovering and exploiting cross platform discrepant apis in wechat. In *31st USENIX Security Symposium (USENIX Security 23)*, 2023.

[39] W. Chao, Y. Zhang, and Z. Lin. Uncovering and exploiting hidden apis in mobile super apps. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023.

[40] Q. Chen and A. Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1687–1700, 2018.

[41] W. O. Community. Miniapp architecture design (i). https://developers.weixin.qq.com/community/develop/article/doc/000a4c1620c188f3adf7db9ab5b413, 2019.

[42] M. Dhawan and V. Ganapathy. Analyzing information flow in javascript-based browser extensions. In *2009 Annual Computer Security Applications Conference*, pages 382–391. IEEE, 2009.

[43] Github. jsjs-vm-demo. https://github.com/bramblex/jsjs-vm-demo, 2023.

[44] T. inc. Wechat mini-program's official document (data verification and encryption). https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/signature.html, 2020.

[45] A. Kapravelos, C. Grier, N. Chachra, C. Kruegel, G. Vigna, and V. Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 641–654, 2014.

[46] H. Lu, L. Xing, Y. Xiao, Y. Zhang, X. Liao, X. Wang, and X. Wang. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security*, pages 569–585, 2020.

[47] QPSoftware. Wechat mini program - all you need to know. https://qpsoftware.net/blog/wechat-mini-program-all-you-need-know, 2023.

[48] D. F. Somé. Empoweb: Empowering web applications with browser extensions. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 227–245. IEEE, 2019.

[49] M. Stiltner. The top 6 super apps in asia – and what they reveal about the global trend. https://www.rapyd.net/blog/the-top-6-super-apps-in-asia-and-what-they-reveal-about-a-global-trend/.

[50] D. Talk. Dingding talk mini-program documentation. https://open.dingtalk.com/document/orgapp-server/how-to-call-apis.

[51] Y. Tang, Y. Sui, H. Wang, X. Luo, H. Zhou, and Z. Xu. All your app links are belong to us: understanding the threats of instant apps based attacks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 914–926, 2020.

[52] Techcrunch. Google officially launching chrome extensions next week. https://techcrunch.com/2009/12/05/chrome-extensions-gallery/, 2009.

[53] Tencent. Qq mini-program documentation. https://q.qq.com/wiki/develop/miniprogram/server/open_port/port_use.html.

[54] Tencent. Wechat role-based access control. https://developers.weixin.qq.com/miniprogram/dev/wxcloud/guide/database/security-rules.html.

[55] Tencent. wx.getuserinfo(object object). https://developers.weixin.qq.com/miniprogram/en/dev/api/open-api/user-info/wx.getUserInfo.html, 2023.

[56] R. Wang, L. Xing, X. Wang, and S. Chen. Unauthorized origin crossing on mobile platforms: Threats and mitigation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 635–646, 2013.

[57] T. Wang, Q. Xu, X. Chang, W. Dou, J. Zhu, J. Xie, Y. Deng, J. Yang, J. Yang, J. Wei, et al. Characterizing and detecting bugs in wechat mini-programs. In *Proceedings of the 44th International Conference on Software Engineering*, pages 363–375, 2022.

[58] WeChat. Network restrictions. https://developers.weixin.qq.com/miniprogram/en/dev/framework/ability/network.html, 2021.

[59] WeChat. Regulations on prohibiting the use of javascript interpreter in miniapps. https://developers.weixin.qq.com/community/minihome/doc/0000ae500e4fd0541f2ea33755b801, 2023.

[60] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth, R. Perdisci, and W. Lee. Understanding malvertising through ad-injecting browser extensions. In *Proceedings of the 24th international conference on world wide web*, pages 1286–1295, 2015.

[61] Yandex. miniapp-example. https://github.com/yandex/miniapp-example, 2023.

[62] Yandex. miniapp-example-back-end. https://github.com/yandex/miniapp-example-backend, 2023.

[63] Y. Yang, Y. Zhang, and Z. Lin. Cross miniapp request forgery: Root causes, attacks, and vulnerability detection. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 3079–3092, 2022.

[64] L. Zhang, Z. Zhang, A. Liu, Y. Cao, X. Zhang, Y. Chen, Y. Zhang, G. Yang, and M. Yang. Identity confusion in {WebView-based} mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1597–1613, 2022.

[65] Y. Zhang, B. Turkistani, A. Y. Yang, C. Zuo, and Z. Lin. A measurement study of wechat mini-apps. *ACM SIGMETRICS Performance Evaluation Review*, 49(1):19–20, 2021.

[66] Y. Zhang, Y. Yang, and Z. Lin. 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*, 2023.

A Super App Platforms Investigated

In this paper, we collected 30 super app platforms with the keyword super app as listed in Table 5. These super apps are developed by companies based globally, including those in Asia (China, South Korea, Japan, India, Vietnam, Indonesia, Philippines, Singapore, and Saudi Arabia), Africa (South Africa and Kenya), Europe (Russia), and Latin America (Argentina). Through reverse engineering, we confirmed 20 platforms to be super app platforms with the integration of miniapps from third-party as marked in the blue rows of Table 5, where 4 super apps are implemented with Nebula framework from Alibaba, 1 super app with T7 core from Baidu, 3 super apps with Lynx from Bytedance, 3 super apps

with X5 core from Tencent, and 3 super apps with Android WebView. While we have found similar rendering environments in apps claimed to be super apps, we did not find the entrance to miniapp-alike interfaces or the access is restricted due to regional issue (e.g., Momo only allows registration from Vietnamese phone number). Hence, we did not include these platforms as platforms studied in our paper. Interestingly, we found that some super apps such as YANDEX have created Github repositories related to miniapp demos [61, 62], which may indicate that the miniapp frameworks are under development and testing. Hence, more super apps may emerge after these platforms have finally released the miniapp framework.

Platform	Country	Users (M)	Rendering framework	Has miniapp?	Note
AliPay [17]	China	1,200	Nebula	✔	
DingTalk [4]	China	600		✔	
MPesa [16]	Kenya	52		✔	
VodaPay [31]	South Africa	3		✔	
Baidu [24]	China	657	T7 core	✔	
Huoshan	China	-	Lynx	✔	
Tiktok [28]	China	1,700		✔	
Toutiao [22]	China	209		✔	
QQ [23]	China	597	XWeb w/ X5 core	✔	
WeChat [25]	China	1,300		✔	
WeCom [20]	China	73		✔	
Kuaishou [1]	China	347	Android WebView	✔	
Meituan [21]	China	680	React Native		
Momo [19]	Vietnam	115	React Native		Registration restricted
Ola [30]	India	200			Miniapp entrance not found
Paypay [10]	Japan	55			Registration restricted
Line [15]	South Korea	230		✔	
Toss [29]	South Korea	21			Miniapp entrance not found
Kakao [14]	South Korea	53			Registration restricted
Zalo [2]	Vietnam	75		✔	
Paytm [26]	India	330	Android WebView		Anti-analysis enforced
Airtel Money [3]	India	10			Miniapp entrance not found
Gojek	Indonesia	38			Miniapp entrance not found
PayMaya [5]	Philipine	44			Miniapp entrance not found
Grab [11]	Singapore	33			Miniapp entrance not found
Careem [7]	Saudi Arabia	48			Miniapp entrance not found
Jumia [13]	Nigeria	2			Miniapp entrance not found
Mercado Pago [18]	Argentina	20			Miniapp entrance not found
Yandex [33]	Russia	55	Chromium		Miniapp entrance not found
Binance [6]	China	128	Android WebView	✔	

Table 5: Summary of platforms investigated in this paper.