Article Type: Original research and Brief

The Rise of Miniapps: A New Frontier with Security Challenges in Mobile Apps

Yuqing Yang*†, Chao Wang*, Zhiqiang Lin*, *The Ohio State University, Columbus, Ohio, 43210, USA, †CISPA Helmholtz Center for Information Security, Saarbrücken, Saarland, 66123, Germany

Abstract—Mobile super apps host third party miniapps that reuse platform services (e.g., identity, payments, device and cloud resources) to deliver rich functionality without separate installation. We study this paradigm by first tracing the evolution from built in WebView plus JavaScript bridge to customized miniapp runtimes with store backed distribution and vetting (which balance performance and security), and synthesizing a platform taxonomy. We then derive a threat model and evidence based taxonomy that covers flawed permission scope, cross platform inconsistencies, undocumented privileged interfaces, cross miniapp channel abuse, and key leakage and misuse (with malware that evades vetting and spreads via social features). Finally, we distill practical guidance for platform owners and regulators and we outline open research problems for semantic aware analysis and realistic benchmarking. Our goal is a shared vocabulary and requirements that help harden super app ecosystems while preserving usability and economic benefits.

Index Terms: *H.3.5.e Web-based services, J.8.s Web site management/development tools, K.6.m.b Security*

The super apps have emerged as a popular solution among a large number of IT vendors. Such paradigm enables convenient yet versatile deployment of various services in the form called "miniapp", allowing third-party developers to deliver a variety of services while reusing the components and services within a single super app. These super apps, often described as "like a Swiss army knife", allow users to book tickets, pay for meals, and even make medical appointments within one app. These miniapps, on the other hand, despite being lightweight in a form similar to a packed website, can achieve powerful functionalities by reusing components exposed by the super-apps. For example, a WeChat miniapp can invoke the WeChat Pay function to enable purchasing with just one API.

By integrating miniapps, super apps not only expand their functionality but also open additional revenue streams through cloud services and advertisements. As such, such paradigm has been gaining popularity among various vendors across the world. For instance, WeChat, with over a billion users, hosts more than 4 million miniapps, serves 600 million daily active users, and facilitates transactions exceeding 2.7 trillion CNY [4]. This paradigm has also been adopted by other applications such as Line in Japan, VodaPay in South Africa, and Zalo in Vietnam, which cater to massive user bases.

Despite being an emerging paradigm, the miniapps retain traits from traditional paradigms. The super app vendors such as WeChat, are essentially mobile applications installed on users' devices, managing resources that overlap significantly with those of traditional mobile apps. Meanwhile, miniapps, typically developed using JavaScript, Markup Languages, and Style Sheets, rely on APIs provided by super apps to access host app modules like payment systems, offering users a native-like experience. However, compared with the traditional paradigms, miniapps benefit from deeper integration with device resources and critical super app components. This integration elevates super apps to the level of comprehensive "operating systems", managing both mobile and web resources. These include peripheral devices like Bluetooth, cloud-based services such

Digital Object Identifier 10.1109/XXX.0000.0000000

as databases, and privacy-sensitive data like account details and phone numbers.

Unfortunately, the expanded access to core components and vast user database from third-party introduces significant security challenges. Malicious actors exploiting vulnerabilities in super apps could impact vast user bases and spread malware through social networks, such as group chats and contact lists, causing severe consequences. Recent studies [3], [10], [13], [7], [8], [9] have highlighted persistent vulnerabilities and design flaws in super apps, making them attractive targets for attackers. Further, the super app platforms have been targeted by malware and malicious developers who attempt to cause privacy and financial losses to the users utilizing the convenient distribution of malicious miniapps among the built-in social network.

Motivated by these challenges, we present the first systematic study on the security of the miniapp paradigm. Through reverse engineering and literature analysis, we identify and dissect super app architectures, reveal key threats, identify root causes, and propose best practices to mitigate risks and strengthen the ecosystem against exploitation. Our work aims to provide a comprehensive understanding of miniapps and their significance, thereby bringing attention from the community to work together on strengthening the security and privacy of this emerging paradigm.

Super App in a Nutshell

What is a Super App

A super app is a comprehensive, multifunctional application platform that serves as a single entry point for accessing a diverse range of services. Unlike traditional standalone applications, super apps enable third-party developers to integrate their services within its ecosystem. This integration fosters a vibrant and versatile environment where users can seamlessly interact with various functionalities—such as messaging, shopping, payment, and transportation—without needing to switch between multiple applications. The ability to support third-party integration is a key defining feature of a super app, as it opens the door to a dynamic and expanding ecosystem of services. With this definition, for example, WeChat is a typical super app, but Uber is not though it offers multiple services such as food delivery in addition to ride hailing since it does not offer the integration of 3rd-party services.

Building on this capability, miniapps are introduced as an integral part of the super app ecosystem. A miniapp is a lightweight package developed by third-party developers that is submitted to the super app platform, which is then distributed into end users' super apps. Unlike standalone applications, miniapps cannot be executed alone because they rely on the execution environment, delegated resources, and exposed components from the super app. This dependency enables third-party developers to create and deliver services efficiently within the super app's ecosystem, eliminating the need for users to download or manage separate applications. In short, the super app acts as a host, providing the necessary infrastructure, user base, and integration platform, while miniapps contribute specialized functionalities that enhance the overall ecosystem. This interdependence underscores the super app's role as a versatile platform, with its ability to support miniapps serving as a key criterion for distinguishing it as a super app rather than a standalone app or miniapp platform. Since different super apps may implement miniapps in varying ways and use diverse mechanisms, it can be challenging to establish a universal definition. Therefore, we define a super app (and the miniapps running atop it) based on the following criteria: (1) it integrates a generic cross-platform execution environment, (2) it delegates super-app-managed resources to third-party developers, and (3) it exposes super app components to facilitate miniapp functionalities.

1) Integration of generic execution environment. A key feature of miniapp platform is the integration of a generic, cross-platform execution environment. To be more precise, a WebViwe supported by a JavaScript engine (e.g., V8) is commonly adopted by super app platforms. This solution enables developers to create miniapps using web technologies like JavaScript, HTML, and CSS, without worrying about cross-platform compatibility or learning a new type of language. This abstraction simplifies development and ensures consistent performance across diverse mobile platforms.

2) Delegation of super-app-managed resources. Super apps are super not only because of the functionality, but also because of the huge users these vendors provide services to, as seen in WeChat (1.2 billion users) and AliPay (the largest payment app in China). Over time, these platforms accumulate substantial user data, such as account details, delivery addresses, and phone numbers. These resources are delegated to miniapps, enabling functionalities like "one-click login" using a user's phone number as an identifier. These information, when delegate to miniapp developers, can further extend the services miniapps can provide significantly.

3) Exposure of super app components for miniapps. To facilitate seamless integration, miniapp platforms expose core components via JavaScript APIs, allowing miniapps to invoke functionalities such as payments, authentication, and social sharing. For instance, WeChat integrates its payment system via WeBank, enabling users to perform transactions, invest, or deposit funds directly within the app.

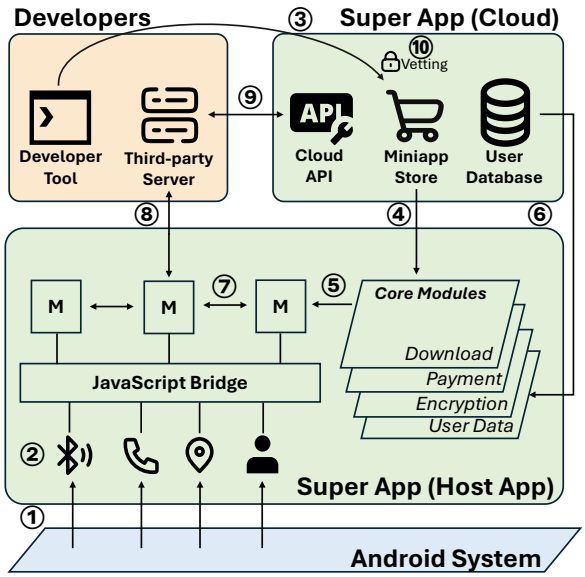

Figure 1: Overview of the key components of super app ecosystem. The box with "M" represents miniapps being executed.

This model has been a key driver of miniapp paradigm adoption, particularly in regions with advanced mobile payment infrastructure, such as East Asia. This further enables innovative e-commerce models, such as live shopping, as exemplified by platforms like TIKTOK and KUAISHOU.

Terminology and definition: Based on the discussion above, we summarize the terminology of super apps and miniapps separately as follows. **Super apps** are the mobile apps that integrates execution engine for third-party code packages, delegate resources managed by the mobile application, and expose app modules for these code packages to reduce the complexity of development. **Miniapps** are the code packages developed by third-party developers and submitted to the super apps, which resides in and depends on a specific super app to function. The relationship between super apps and miniapps is similar to the relationship between Operating System and applications, where the former provides execution environment and resources, and the latter provides versatile services utilizing these resources. In the rest of the paper, we will focus mainly on the super apps, with miniapps being integral components when assessing the security mechanisms and pitfalls.

Resource Management in Super Apps

With years of development, super apps have essentially become a hybrid system equivalent to an Operating System, involving resource management at user end, data delegation and service invocation at cloud, along with a built-in miniapp store to distribute the miniapp packages. As illustrated in Figure 1, the paradigm involves three parties. The super app has the host app installed on the user's device (e.g., the WECHAT app on user Alice's smartphone), and the super app cloud which facilitates communication between the host app on the user's side and the server on the developer's side. The user manages the device (blue box). And the developers has their own server as well as devices for developing miniapps (orange box).

This paradigm involves complex interactions, as denoted by numbered steps. First, as a standalone mobile app, the host app requests permission to access local resources from the underlying system, such as Bluetooth and GPS location (①). These permissions allow miniapps to access these resources via JavaScript APIs provided by the super app (②). To deploy a miniapp, a developer acquires the developer tool and submit their packed miniapps to the official miniapp store (③), which can be accessed by users. When a user clicks on a miniapp, the miniapp package is download from the miniapp store (④). Once downloaded, the miniapp can be executed with access to the local resources granted by the host app. In addition to the local resources on the user's device, miniapps can also access super app's components such as payment components (⑤), as well as resources, such as account details and phone numbers. These data are downloaded and processed by the components of miniapps for encryption to achieve high confidentiality against eavesdroppers (⑥). Furthermore, miniapps can interact with other miniapps installed on the device (⑦), and communicate with their back-end servers (⑧). In addition to this, the super apps incorporate cloud API gateways for developer servers to invoke cloud services and access user-related information such as ID and tokens (⑨).

The super app cloud plays a critical role in this ecosystem, particularly for token-based cloud authentication (⑨). Given the sensitive nature of the resources involved, super apps enforce mandatory vetting of miniapp packages before submission (⑩). Since miniapps can only be submitted to the super app's own miniapp market, this mandatory vetting process ensures that only secure, malware-free miniapps are available to users.

Super Apps vs. Traditional Apps

While super apps share similarities with web and mobile apps, they are different in platform design, developer interaction, and user experience. Table 1 highlights the differences using examples such as Android (mobile OS), Chrome (web browser), and WECHAT (super app).

Platforms. Super apps manage a wider spectrum of resources, and exercise strict control over miniapp distribution to prevent malware. This is particularly for the third-party app codes. Android app is loosely managed because

applications can be downloaded and installed from various app markets. In Chrome, the code of the front-end of a web app is distributed to the browser, and is considered as being partially managed by the browser. However, there is not a centralized management nor vetting on such front-end code, and thus we mark it partial management. In miniapps, the package can only be fetched via the single built-in app market, and can only be installed through official markets, with no side-loading permitted, which is more comprehensively managed. Also, super apps manage sensitive user data—such as accounts and personal information—exclusively, unlike mobile OS or browser platforms that delegate such management to external entities.

Developers. Developing miniapps is streamlined compared to traditional web apps. Developers can leverage APIs for accessing super app components, such as payments and cloud databases, without needing a separate back-end server. Further, the API support open for developers are much richer than those for web apps. For instance, we extracted the interface declaration file from Android framework repository for Android APIs, DefinitelyTyped repository for chrome APIs, and api-typings repository for WeChat miniapp APIs, separately. This centralized model simplifies development, reduces costs, and ensures cross-platform compatibility.

End Users. For users, miniapps provide an "install-free" experience akin to opening a web page. Compared with other paradigms, the super apps has a proprietary app store hosting miniapps, and every time a miniapp is launched, it is automatically updated to the newest version. Unlike traditional web apps, users can trust that miniapps are vetted, as super apps maintain exclusive control over their distribution. In addition, super apps enforce strict limit on storage consumption to as low as 10 MB to ensure that miniapps are light-weight. These ensures both security and convenience while minimizing storage and maintenance overhead.

The Evolution of Super Apps

The concept of super apps and miniapps has been evolving for many years, progressing from earlier desktop browsers and their extensions, finally to the sophisticated architectures we see today. As shown in Figure 2, we notice that the development and optimization of the miniapp paradigm have been far from straightforward, demanding a careful balance of complex trade-offs between security and execution efficiency.

(P-I) Browsers with Extensions. Prior to the emergence of miniapps, browsers such as *Chrome* and *Opera* began providing APIs for third-party developers to access services exposed by their browser kernels in 2009. These extensions are essentially code packages written in JavaScript and HTML, along with a manifest file specifying configurations. With an extended API set provided by the browsers, these extensions can perform many tasks such as providing VPN service or block the advertisements, marking the precedent of the mobile miniapp paradigm.

(P-II) Vendor hub with JavaScript Bridge. When mobile payment becomes popular, giant mobile IT vendor begin to integrate additional services inside their applications in the form of a built-in webpage, such as ride hailing services. For example, shortly after WeChat promoted the new built-in payment feature provided by WeBank, a wallet hub is embedded in the app, where users can seamlessly use Didi to book for rides. As such, these apps gradually becomes "super" by transferring into a service hub, where users can access not only services provided by the super app vendors themselves, but also for various collaborating service providers.

On top of the built-in services, it is natural for users to open links inside the super apps, which opens a built-in browser, where users naturally may need to perform complex tasks, such as payment. Instead of allowing users to navigate outside the super apps, the super apps seek a solution to enable invocation to the super app's payment components from these website. This attempt marks the birth of JavaScript bridge, which sets foundation for building the contemporary miniapp runtime. The so-called JavaScript (JS) bridge built by WeChat is an interface layer between the JS code and the underlying Android code. This solution is similar to the Web Browsers' support of extensions, where a layer of JavaScript bridge connects the JavaScript code with app core functionality. As such, online shopping websites can directly launch WeChat payment for retail. However, during this procedure, user experience may still be hindered by efficiency of execution between the JavaScript layer and the Android layer. Consequently, users may notice prolonged delay in web rendering as well as the risk of malicious behaviors such as phishing. These eventually resulted in the birth of miniapp runtime in the next phase.

(P-III) Super apps with native webview. To allow third-party integration of miniapps, the easiest way is to utilize the native webview components of Android with JavaScript Bridge layers, which connects the miniapp code to the super-app components. Vendors including Zalo and Line implement such architectures to support execution of miniapps. Miniapps for both platforms are developed with generic webapp framework, i.e., the React Native framework. For the super apps with such architecture, opening a miniapp is similar to opening a webpage in the browser, except that the miniapp is distributed from the official CDN only (e.g., h5.zdn.vn for Zalo).

Hosts	Mobile OS (Native Apps)	Web Browsers (Web Apps)	Super Apps (Miniapps)
Example Platform	Android	Chrome	WeChat
Platforms			
Managed resources			
Local Resources			
System Resources?	●	◑	●
Third-party App Codes?	○	◑	●
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 (4,534)	Poor (459)	Medium (1,345)
Compatible across Platforms?	○	●	●
Backend?	◑	●	◑
Centralized Vetting?	●	○	●
Users			
Install-free?	○	●	●
App Store?	●	○	●
Storage Consumption?	High ($\leq$ 200MB)*	Low ($\leq$ 5MB)	Low ($\leq$ 10MB)
Update?	Server-based	Client-based	Server-based
Performance?	High	Browser-specific	Super-app-specific
Offline Loading?	High	Low	Medium

Table 1: Comparison between different hosts and their supported apps. "●": full support; "◑": partial support; "○": no support. The data supporting the columns of "API Support" and "Storage Consumption" is based on publicly available official documents and repositories. *: The resource file of Android app can be up to 1.5 GB.

(P-IV) Super apps with miniapp runtime. In response to the need to optimize the performance and security of third-party web access, WeChat started to implement a customized miniapp runtime based on the JavaScript Bridge by separating the execution and rendering of the webpages into multiple threads to maximize the multi-core capability of mobile devices. Meanwhile, WeChat builds an integrated miniapp store, where developers can submit their "web pages" in a proprietary package (`.wxapkg`). These packages, after being vetted by the platform for security screening, are released to end devices of users upon request. Consequently, by seizing the distribution channel and creating a customized execution engine, the platform strikes a balance between performance and security. Nowadays, such a paradigm is being adopted by an increasing number of vendors across the world. As illustrated in Figure 2, with the debut of miniapp in 2017, many mobile application vendors immediately followed the step, including ByteDance (2017), AliPay (2018), and Baidu (2018). Interestingly, AliPay even embedded their miniapp framework for vendors outside China, including MPesa (Kenya, 2020) and VodePay (South Africa, 2021), pushing the paradigm global-wise.

We observe that super apps maintained by the same company may reuse super app frameworks to support execution and rendering of miniapps. As shown in Figure 2, there are X5 from Tencent, Lynx from ByteDance, and Nebula from Alibaba. We identify these name by reverse engineering the applications and identifying the modules responsible for miniapp execution and rendering. As a result, these names are found in the corresponding module, indicating the codename for the frameworks. It is also worth to note that the super app paradigm is still evolving,

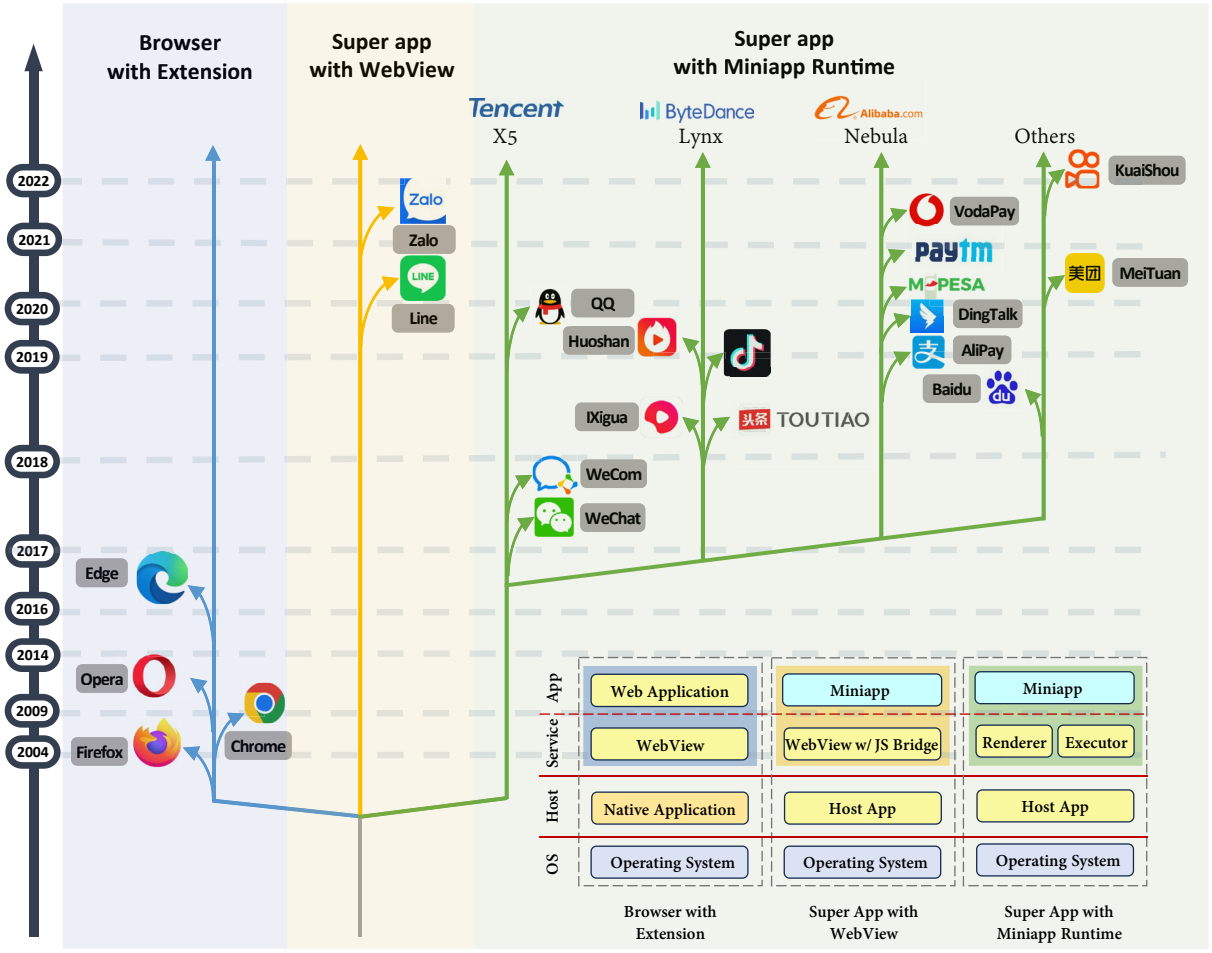

Figure 2: The taxonomy of super app platforms. The architectures of each cluster are shown at the bottom right, where resources in boxes with dotted lines are optional.

with new super apps emerging constantly. Also, there are a large amount of platforms providing super-app-alike services, such as UBER, GOJEK, and GRAB. These platforms provide multiple services merely from the first-party, which is similar to the "Vendor Hub" phase of super apps, but still bear potential to transform into miniapp platforms in the future. However, in this paper, we focus solely on the platforms allowing third-party service integration via third-party miniapps, and thus they are excluded from the scope of this paper.

Threats

The miniapp paradigm introduces unique opportunities for innovation but also exposes significant security challenges. These threats involve two types: vulnerabilities from architectural or implementation flaws, and threats posed from evasive malware. This section examines both categories in detail, highlighting specific issues and their implications for the super app ecosystem.

Threats from Vulnerabilities

Vulnerabilities in super apps often stem from inconsistencies in permission enforcement, platform-specific security gaps, and improper access control mechanisms. These flaws expose users and developers to risks that can be exploited by malicious actors.

(T1) Flawed Permission Scope [3]. Super apps manage a broader range of resources than traditional apps, including user information and account details. To prevent resource abuse from miniapps, super apps enforce additional permissions. However, these permissions are not always consistent with the underlying operating systems. For example, Lu *et al.* [3] found that while Android requires location permissions (`ACCESS_COARSE_LOCATION` and

ACCESS_FINE_LOCATION) to access Wi-Fi lists, the `wx.getWifiList` API in WeChat bypassed such requirements, which may result in over-collection of user privacy data and privilege escalation.

(T2) Cross-Platform Inconsistencies [7]. Miniapps feature cross-platform compatibility offered by super apps, but variations in security policies, implementation, and updates across platforms can create exploitable gaps. For instance, APIs for sensors like accelerometers, compasses, and gyroscopes are often platform-specific. The absence of critical APIs, such as `makeBluetoothPair` on iOS, prevents Bluetooth devices from distinguishing between trusted and untrusted sources, exposing users to Man-in-the-Middle attacks.

(T3) Exploitable Privileged APIs [8]. To prevent abuse from miniapps, sensitive APIs such as `eval()` are restricted to mitigate risks. However, undocumented privileged APIs may remain accessible, lacking proper access controls. For instance, in WeChat, attackers have exploited hidden APIs like `captureScreen` to take screenshots and send them to remote servers, `getLocalPhoneNumber` to extract users' phone numbers. Such vulnerabilities enable miniapps to steal sensitive user data or perform malicious activities undetected.

(T4) Cross-Miniapp Channel Hijacking [10], [15]. Miniapps often cross-communicate to perform complex tasks, such as working with a payment miniapp to facilitate the purchase of a mini-app. However, inadequate access control and isolation in these cross-miniapp communication channels can expose the ecosystem to exploitation, enabling malicious miniapps to hijack legitimate channels and compromise other miniapps.

(T5) Key Misuse and Abuse [13]. Miniapps rely on token-based protocols to verify the identities of web requests and users when accessing cloud resources. These protocols use a master key, "AppSecret," generated by the super app and assigned to miniapps. Although these keys are intended to remain confidential, studies have revealed widespread misuse and leakage in miniapp packages. Such exposed keys can cause unauthorized activities including account hijacking (unauthorized access to user accounts), promotion abuse (manipulating promotional campaigns for personal gain), and service theft (using paid services without authorization).

(T6) Vague Identity Isolation [12]. Super apps utilize domain allowlists to regulate web page access, ensuring that miniapps can only interact with trusted sources. However, Zhang *et al.* [12] identified a vulnerability called "identity confusion", where privilege boundaries between miniapps and web domains are blurred. For example, an unprivileged miniapp may include a privileged web domain, and vice versa. This confusion enables adversaries to manipulate their identities, masquerading as trusted and privileged entities to bypass access control.

Threats from Malware

In addition to vulnerabilities, super apps face threats from malware attempting to evade detection. These threats exploit weaknesses in vetting mechanisms, identity verification, and social dynamics within the ecosystem.

(T7) Anti-Vetting Techniques [11]. Malicious miniapps continuously adapt to evade detection despite the implementation of rigorous vetting mechanisms. Common evasion strategies include code obfuscation, dynamic code loading, and embedding evasive behaviors that conceal malicious intent during the vetting process. Additionally, developers have employed custom interpreters to execute post-vetting updates, enabling malicious functionality to be introduced after the miniapp has been approved. By masquerading as benign applications to pass initial vetting, these miniapps can later transform into malware with minimal modifications, such as a small code update or a remotely fetched flag. This approach facilitates the stealthy distribution of malware within the ecosystem, posing significant risks to users and platform integrity.

(T8) Social Engineering and Phishing [2]. Malware takes advantage of the social features embedded in super apps, such as friend lists, group chats, and social sharing mechanisms, to deceive users and manipulate them into performing unintended or harmful actions. For instance, malicious miniapps may impersonate legitimate miniapps, exploiting users' trust to steal credentials, payment information, or other sensitive data. Additionally, malware often launches fake promotions or scams, enticing users with promises of rewards, discounts, or exclusive offers to lure them into sharing personal information or clicking on harmful links.

The impact of these attacks is particularly worrisome due to the interconnected nature of super apps. Malware can rapidly propagate through users' social networks, spreading malicious links or fraudulent miniapps to friends, groups, and communities with minimal user intervention. This amplifies the scale and reach of the attack, potentially compromising a large number of users within a short time. Furthermore, the trust-based environment of super apps makes users less susceptible to these attacks, as interactions often occur within familiar social circles.

On top of that, exploiting the social aspect can enable the propagation of more types of malicious behaviors, such as exploiting advertising models. For instance, some miniapps cheat advertisement systems by forcing users to share links with friends, masquerading as legitimate

miniapps, or acting as "portal miniapps". These portals funnel user traffic to other miniapps, artificially inflating click counts and generating higher ad revenue. Such deceptive tactics undermine platform integrity and require continuous monitoring and mitigation efforts.

Mitigation

Best Security Practice

By summarizing and evaluating the security mechanisms adopted by the super app platforms, we extract common practices that are beneficial to improving the overal security of super apps, which includes standardization of security design, elevating developer education via documentation, performing comprehensive identity verification, as well as enforcing interactive and self-adaptive vetting.

Security design standardization. In this paper, we have illustrated threats due to implementation issues and compatibility issues (**T1**, **T2**, **T3**). As super apps may implement mechanisms differently, and underlying systems may differ it is vital to ensure that the security design is both aligned with all compatible underlying system. This involve both validation on the design of permission mechanism and storage management across systems. Furthermore, the super apps need to collaborate on a standardized set of security mechanism that utilize the best practice across each platforms [1].

Developer education. Ensuring the security of the ecosystem is never the sole work for the platforms alone, as a number of vulnerabilities are rooted in unawareness of security mechanisms from the developers. For instance, developers commonly misplaces trust on the platforms, thinking that their miniapp front-end packages are protected and can be trusted. Unfortunately, these packages are ultimately deployed to the front-end and can be manipulated by malicious users. Certain platforms such as WeChat attempt to mitigate this issue by explicitly educating the developers not to leak sensitive keys and code snippets to the front-end packages, on top of the underlying mechanisms for verification, which helps to reduce the attack channel available for malicious parties.

Comprehensive identity verification. In this paper, a number of threats (**T4** and **T6**) are caused due to weak verification on the identity of miniapps and back-end domains, enabling malicious parties to impersonate the official parties to inflict losses. As such, strict verification needs to be enforced prior to transmitting messaging across either front-end or back-end channels.

Interactive and self-adaptive vetting. As many other web platforms, the threat from malware may eventually disrupt the healthiness of the ecosystem (**T7** and **T8**). Given the evasive and adversarial nature of malware developers, mererly adopting fixed mechanism is not enough to tackle this threat. As such, it is vital to deploy interactive vetting mechanisms. On top of vetting miniapps from the platforms' perspective, the super apps need to encourage users and developer community to share cases of newly identified malware, which provides valuable insights to be added to the malware detection mechanism that gradually improve the security of the ecosystem in a self-adaptive way.

Open Problems

Performing Fine-grained Program Analysis and Benchmarking. Despite existing works developing analysis tools to aid vulnerability discovery [10], [6], [14], the current research falls short upon dynamic analysis solutions. However, given the dynamic nature and the fragmented and proprietary landscape of super apps, the community still calls upon a dynamic execution testbed for identifying and confirming security issues among miniapps. For instance, miniapps may actively attempt to hide malicious traces in the code whereas executing their malicious behavior dynamically. Further, the acquisition and utilization of permission may vary depending how miniapps are executed. All of these scenarios call for a powerful and accurate program analysis tool set to facilitate the detection. Unfortunately, while super app platforms have internal tools to test uploaded miniapp packages, these tools are generally kept private. As such, efforts from the open-source community leading towards a set of analysis and miniapp testing framework can significantly benefit the community of miniapp and super app security.

Performing Semantic-aware Miniapp Vetting. We have shown that the current vetting mechanisms still face challenges and limitations despite the strict control on the distribution of miniapps [11]. 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.

Standardizing the Security Mechanisms. On top of discrepancies of underlying operating systems, the difference of implementation across super app platforms may also introduce vulnerabilities breaking access control in API invocation and miniapp communication. Future works may include a study on measuring feasible security mechanisms for super app platforms, and a standardized security protection solution deployable by all super app platforms to ensure that the protection implementations are equally stringent.

Discussion

Why Threats Exist?

In this paper, we identified 8 threats in the emerging miniapp paradigm. These threats are caused by vulnerabilities in design or malware exploitations that the platforms need to tackle. In general, the root cause of these issues involve compatibility, implementation, and trust model issue:

Compatibility Issue. As miniapps are executed in different operating systems (Android, Windows, iOS, etc.), the security mechanisms may perform differently across platform. Consequently, vulnerabilities in permission mechanisms (**T1**) and resource management (**T2**) may still create exist in super apps. To address this, the super apps need to accommodate discrepancies across underlying systems, and to implement additional protection if needed.

Implementation Issue. While mechanisms from existing paradigms are deployed, the implementation can be incomplete (**T3**) or weak (**T4**), which eventually undermines or even voids the security mechanisms. Hence, super app platforms need to ensure that the implementation is following standardized practices and cover comprehensive aspects of the ecosystem.

Trust Model Issue. As miniapp execution is complex, the boundaries between parties and the trust models becomes vague. From the developer's perspective, they should not place excessive trust on the platform regarding protection. Otherwise, they may leak critical information in the code packages (**T5**). From the platform's perspective, the super apps should not trust miniapps even if they are vetted, as they may contain vulnerabilities not covered by vetting (**T6**), or perform malicious activities even if they are vetted (**T7**, **T8**).

Balancing the 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. As such, these trade-offs need to be considered when designing mitigation and countermeasure mechanisms. These includes trade-offs between privacy and monetization, security and usability, as well as centralization and openness.

Privacy and monetization. Super apps allow third-party developers to access sensitive user data to operate, or even allow miniapps to monetize directly by embedding advertisements from the platform. As such, it is paramount to ensure the users are acknowledged of the use of sensitive data and secure the transmission of these data to third-party miniapps. Additionally, it is also vital to make sure that the monetization involving these privacy data is not abused. However, it remains challenging to determine whether a certain access to sensitive data is necessary and appropriate. For example, while permission system exist, miniapps in many super apps only need to ask for the permission once, and can then get the user data for the rest of the time. Moreover, we discovered that different super apps have contradictory view on whether a certain type of data need to be protected by permissions. For instance, TIKTOK requires miniapps to first require for `scope.clipboard` permission before accessing Clipboard data [5], whereas WECHAT does not require permissions prior to accessing or setting clipboard data. Considering the potential risk of abuse on user privacy data, super app platforms need to come up with a consensus on data protection and permission system to prevent attackers from utilizing such discrepancy between super apps to abuse the data.

Security and usability. As super apps attract malicious developers who want to abuse the data acquisition, the super app platforms have been in an eternal arm race against these bad actors. Nevertheless, enforcing too complicated security mechanisms may raise complaints from developers and users of inconvenience to use the miniapp, which instead keeps users and developers from the super apps. For example, WECHAT had changed the access to user information multiple times to fight against malware who abuses the user information acquisition mechanism. In the beginning, the access to user information need to ask for the permission `scope.userinfo`. However, many miniapps abuse the mechanisms that prompts the user of accessing user info every time, regardless of whether the miniapps needs to access the user information. To address the users' concerns, the platform later added an additional mechanism, prohibiting the miniapps from actively prompt the user of accessing user information, and the information access can only be triggered when the user clicks on a certain button. Nevertheless, some miniapp started to refuse providing services to end users without forcing the user to click the button even though acquiring the user's avatar and nickname is not mandatory for the services, and the platform again changed the mechanism. Consequently, due to the existence of bad miniapp actors and constant change to balance the security and convenience, complaints have been building up in the forum from developers about efforts need to make to refactor the code.

Centralization and Openness. The Super App paradigm is a typical centralized platform, with the platform exclu-

Platform	Country	Category	Users (M)*
WeChat	China	Social	1,260
Alipay	China	Payment	1,300
QQ	China	Social	569
Tiktok	China	Entertainment	639
Meituan	China	Food delivery	770
Line	Korea / Japan	Social	178
KakaoTalk	Korea	Social	54
Baidu	China	Entertainment	632
Zalo	Vietnam	Payment	70
Grab	Indonesia	Food delivery	46
VodaPay	South Africa	Payment	4.8
Slack	USA	Collaboration	42
Teams	USA	Collaboration	320
Snapchat	USA	Social	800

Table 2: Super app platforms worldwide. *: the amount of users is fetched via publicly available data for the platforms hosting miniapps.

sively controlling the vetting, distribution, and execution of miniapps. In a sense, it enables strict security mechanism to avoid exploitations abusing the weak verification of identities, such as enforced back-end domain verification, and developer identity vetting. However, this also creates discrepancy across platforms, making it challenging for developers to integrate miniapps for multiple platforms. Moreover, this makes many security mechanisms implicit from the developers, causing a variety of security concerns due to unawareness and misplaced trust on the platform. For example, developers commonly regard their miniapp safe as they are exclusively hosted on the super app, so as to leave processing of sensitive encrypted data at the front-end. Unfortunately, the front-end miniapp packages can be easily obtained by malicious users, and the processing can be hijacked, inflicting losses to the miniapp developers. As such, striking a balance between centralization for strict security mechanisms and openness for educating developers is vital in ensuring the overal safety of the user data and miniapps.

Geographical Distribution of Super App

In this paper, we particularly focused on the super apps that meet three criteria: integrates a generic execution environment, delegates resources managed by super apps, exposes components for the third-party developers. As shown in Table 2, the distribution of super apps still illustrate a geographical characteristic, with Chinese and Asian companies taking a lead, and middle-east companies following. A potential explanation to this phenomenon is user preferences between convenience, usability, and privacy. While a centralized and integrated service may create a convenient one-step experience, privacy issue may emerge as the operator of the super app platform gains access to a wide spectrum of personal information. Meanwhile, integrating an execution engine inside an application inheritedly creates performance issue, limiting the complexity of the miniapps.

"Super Apps" Outsde Mobile Ecosystem

However, we would like to make it clear that there are multiple ways to define a super app. In a broader sense, if we look into more similar application architectures that, on top of its core functionalities, allow third-party developers to embed services into the application. These services can either involve code distributed to the front-end or expose no code components on the user side, which includes many US- and European- based apps. For example, applications such as Visual Studio Code and Atom support extensions for developer to edit their project more conveniently, and Chatbot-based services in apps such as Slack and Discord have been widely used to provide versatile services, even allowing users to play games in the form of text. Moreover, ChatGPT released their own plugin system to connect the large language model with various services providers, allowing them to provide customized services such as booking flight tickets. These different application scenarios naturally involves novel challenges and security problems, which calls for future research in this community.

Conclusion

In conclusion, mobile super apps have transformed the app ecosystem by enabling third-party developers to integrate add-ons seamlessly. Despite robust security measures such as permissions, sandboxing, and encryption, challenges like phishing attacks and balancing revenue generation with user experience persist. Addressing these requires heightened user awareness and ongoing research to enhance security and usability. While super apps offer unmatched functionality and convenience, refining security models and mitigating emerging threats are essential. A proactive approach to safeguarding user trust and ensuring platform integrity will be critical to their continued success.

References

1. W. Consortium. Miniapps | working groups. https://www.w3.org/groups/wg/miniapps/, 2025.
2. X. Deng, M. Zhang, X. Dong, and X. Hu. Detect counterfeit mini-apps: A case study on wechat. In Z. Lin and L. Xing, editors, *Proceedings of the ACM Workshop on Secure and Trustworthy Superapps, SaTS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 1–10. ACM, 2024.

3. 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.
4. QPSoftware. Wechat mini program - all you need to know. https://qpsoftware.net/blog/wechat-mini-program-all-you-need-know, 2023.
5. TikTok. User authorization. https://developer.open-douyin.com/docs/resource/zh-CN/mini-app/develop/api/other/user-authorization/, 2023.
6. C. Wang, R. Ko, Y. Zhang, Y. Yang, and Z. Lin. Taintmini: Detecting flow of sensitive data in mini-programs with static taint analysis. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, 2023.
7. C. Wang, Y. Zhang, and Z. Lin. One size does not fit all: Uncovering and exploiting cross platform discrepant {APIs} in {WeChat}. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6629–6646, 2023.
8. C. Wang, 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*, pages 2471–2485, 2023.
9. C. Wang, Y. Zhang, and Z. Lin. Rootfree attacks: Exploiting mobile platform's super apps from desktop. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, pages 830–842, 2024.
10. 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.
11. Y. Yang, Y. Zhang, and Z. Lin. Understanding the miniapp malware: Identification, dissection, and characterization. In *32nd Network and Distributed Systems Security Symposium (NDSS)*, 2025.
12. 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.
13. 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.
14. Z. Zhang, J. Du, W. Diao, and J. Wu. Minible: Exploring insecure BLE API usages in mini-programs. In *Proceedings of the ACM Workshop on Secure and Trustworthy Superapps, SaTS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 18–22. ACM, 2024.
15. Z. Zhang, Q. Hou, L. Ying, W. Diao, Y. Gu, R. Li, S. Guo, and H. Duan. Minicat: Understanding and detecting cross-page request forgery vulnerabilities in mini-programs. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 525–539, 2024.

Dr. Yuqing Yang is a PostDoctoral Researcher at CISPA Helmboltz Center for Information Security, 66123 Saarbrücken, Germany, and the Invited Expert of the W3C Miniapp Working Group. His research interests include 1) security analysis for vulnerability and malware detection, 2) security modeling of sensitive and malicious behavior of applications, and 3) scalable measurement and evaluation on applications and platforms for emerging web and mobile platforms. He received his Ph.D. in Computer Science and Engineering from the Ohio State University. Contact him at yuqing.yang@cispa.de.

Chao Wang is a PhD candidate at The Ohio State University. Prior to that, he obtained his Master of Science degree in Computer Science and Engineering from the Ohio State University. His research interest lies in mobile security and web security, in particular mobile Super/Mini App security and JavaScript program analysis. He is also a DevOps engineer focused on code development and server operations. Contact him at wang.15147@osu.edu.

Dr. Zhiqiang Lin is a Distinguished Professor of Engineering, and the Director of Institute for Cybersecurity and Digital Trust (ICDT) at The Ohio State University. His research interests center around systems and software security, with an emphasis on (1) developing automated binary analysis techniques for vulnerability discovery and malware analysis, (2) hardening the systems and software from binary code rewriting, virtualization, and trusted execution environment (TEE), and (3) the applications of these techniques in Mobile, IoT, Cloud, and Edge Computing. He has published over 170 papers, many of which appeared in the top venues in cybersecurity. He received his Ph.D. in Computer Science from Purdue University. Contact him at zlin@cse.ohio-state.edu.