TrustedDomain Compromise Attack in App-in-app Ecosystems

Zhibo Zhang
Fudan University
Shanghai, China
zhibozhang19@fudan.edu.cn

Zhangyue Zhang
Fudan University
Shanghai, China
23210240095@m.fudan.edu.cn

Keke Lian
Fudan University
Shanghai, China
kklian20@fudan.edu.cn

Guangliang Yang
Fudan University
Shanghai, China
yanggl@fudan.edu.cn

Lei Zhang
Fudan University
Shanghai, China
zxl@fudan.edu.cn

Yuan Zhang
Fudan University
Shanghai, China
yuanxzhang@fudan.edu.cn

Min Yang
Fudan University
Shanghai, China
m_yang@fudan.edu.cn

ABSTRACT

Emerging app-in-app ecosystems (e.g., WeChat) provide a lightweight and efficient WebView-based runtime for mini-apps, which frequently load rich web content from remote servers and access sensitive resources via APIs provided by the super-apps (a.k.a. the app-in-app frameworks). Inspired by the content security policy (CSP), super-apps enforce a domain-based allowlist to prevent mini-apps from loading untrusted and malicious web content.

In this paper, we observe that the domain-based allowlist mechanism is unreliable in app-in-app ecosystems because it assumes all web pages under the allowlist domain are trusted. To demonstrate such weakness, we propose a novel attack called TrustedDomain Compromise (TDC) Attack, along with two interesting attack vectors, through which attackers can manipulate unsafe domains or URLs to bypass the allowlist check and launch phishing attack or abuse runtime APIs. Thereafter, we conduct the first empirical study on the TDCAttack in the real-world app-in-app ecosystems. Specifically, we investigate the underlying reasons for the failure of the allowlist mechanism and propose an automated analysis framework for identifying TDCAttacks in real-world mini-apps. Our experiment shows that popular app-in-app ecosystems including WeChat, Alipay, and Baidu are all vulnerable to the TDCAttack. Further, we have identified 26 exploitable real-world mini-apps.

CCS CONCEPTS

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

KEYWORDS

App-in-app; Code Injection Attack; Allowlist; Security Analysis

ACM Reference Format:
Zhibo Zhang, Zhangyue Zhang, Keke Lian, Guangliang Yang, Lei Zhang, Yuan Zhang, and Min Yang. 2023. TrustedDomain Compromise Attack in App-in-app Ecosystems. In *Proceedings of the 2023 ACM Workshop on Secure and Trustworthy Superapps (SaTS '23), November 26, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3605762.3624430

1 INTRODUCTION

Recently, the app-in-app paradigm has gained significant popularity. With its array of diverse mini-apps, it provides users access to distinct features and functionalities without necessitating them to exit the super-app. These mini-apps dynamically load remote web content into a WebView-based mini-app runtime [5], possessing the capability to access sensitive super-app user data and system resources via the dedicated APIs provided by the super-apps. Therefore, for the purpose of ensuring the security of the app-in-app ecosystem, it is crucial to guarantee that the web content loaded by the mini-app should be benign and trusted.

Inspired by the practical and effective web security policy, i.e., Content Security Policy (CSP) [3], the app-in-app ecosystem employs a domain-oriented *allowlist* mechanism against the loading and injection of potentially harmful web content in mini-apps. To enable the *allowlist* protection in a mini-app, its developers define the allowlist of the web domains whose content is treated as secure. In runtime, this allowlist is respected and enforced by the corresponding super-app. In particular, when a mini-app accesses a URL, the super-app intercepts the request of the URL and only permits if it matches the allowlist. In this way, even if a remote attacker targets on a mini-app through web and mobile attacks, e.g., content-injection or phishing deeplink [32], allowlist can prevent malicious content from being loaded into the mini-app.

As discussed above, the allowlist security protection plays a vital role in safeguarding the security of mini-apps. However, it is built on the unreliable assumption that all web pages under each domain (denoted as domain assets) listed in the allowlist should be benign and safe, which is increasingly proving to be insufficiently reliable within today's app-in-app ecosystem. In particular, mini-app developers typically construct the domain allowlist based on the URLs their mini-apps commonly access. The quantity and security of these domain assets are often inadequately ensured.

In light of these, we propose *TrustedDomain Compromise Attack*, i.e., TDCAttack, in the app-in-app ecosystem and identify two distinct attack vectors: (i) allowlist domain abusing e.g., subdomain takeover; and (ii) insecure domain assets, e.g., XSS and open redirection. We demonstrate that even when the super-app rigorously enforces the allowlist validation, mini-apps still suffer from TDCAttack. Attackers, by manipulating unsafe domains or domain assets, can achieve (i) phishing attacks by loading pages within the (protected) mini-app context, and (ii) even executing malicious code. More severely, attackers can further abuse the runtime APIs provided by the super-app, i.e., gaining the unauthorized access to sensitive super-app data and system resources.

Our work. In this paper, we conduct the first attempt and present an empirical study of the security of allowlist deployments across different super-apps. Through this study, we aim to explore the following research questions:

RQ1: *What and how many domains are included in the allowlist of a given mini-app?*

RQ2: *Which vulnerabilities can be exploited to launch TDCAttack in various super-apps?*

RQ3: *How does TDCAttack affect real-world mini-apps?*

A significant challenge in solving **RQ1** is that the allowlist configurations of mini-apps are often hardly retrieved. This is mainly because the allowlist is typically stored in the super-app server (i.e., mini-app developers need to submit their allowlist to the super-app for make it effective). The allowlist content is a black box for us. One important insight is that the allowlist content can be reflected by the mini-app's source code. Therefore, we propose a static analysis-based approach against mini-app code to automatically learn the corresponding allowlist content. Moreover, we find that during allowlist matching, wildcard is frequently used. Thus, we extend the obtained allowlist using automated subdomain discovery techniques [4]. As a result, we vet the security of 11838 mini-app in total, including 4,446 mini-apps from WeChat, 3,946 from Alipay, 3,446 from Baidu. We successfully extract 81,978 allowed domains.

To solve **RQ2**, we find there exist semantic gaps between vulnerable domain assets and practical TDCAttack. Specifically, not all web vulnerabilities can be exploited to launch TDCAttack. This is because TDCAttack targets the front-end users of mini-apps and attackers are confined to launch attacks by manipulating links that users click on. Moreover, super-apps may implement customized defenses, resulting in the efficacy of a vulnerability varying across different super-apps. To solve this problem, we investigate the threat models of allowlist mechanism in the high-profile super-apps, i.e., WeChat, Alipay, and Baidu. We carefully analyze the top web vulnerabilities listed by OWASP [2], and check the vulnerabilities that may be exploited for TDCAttack. Overall, we finally identify four types of exploitable vulnerabilities for TDCAttack. Although WeChat and Aliapy enforce their customized defense mechanisms, they are still vulnerable to TDCAttacks. Based on these findings, we further analyze the extracted domain assets and find that 140 mini-apps are potentially vulnerable to TDCAttack.

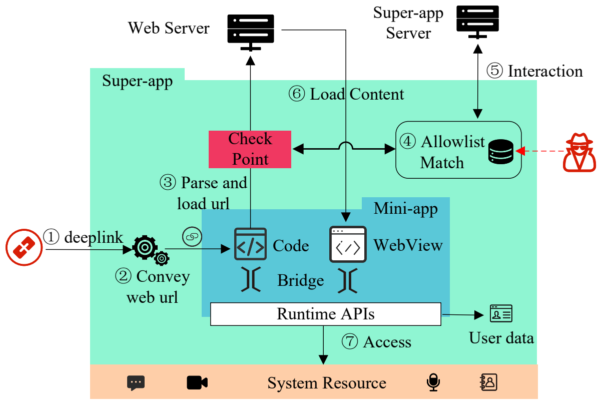

Figure 1: Overview of allowlist mechanism.

To solve **RQ3**, we verify whether the identified vulnerable domain assets can be successfully loaded by the corresponding mini-apps. Specifically, we conduct a static cross-context data-flow analysis from the mini-app's entry points (i.e., onLoad lifecycle functions in JS files) from the URL loading operations (i.e., src assignment in HTML files) to analyze the process of external input parsing. Based on the analysis result, we automatically wrap the vulnerable URLs in a template-based testing triggered by the 'adb' command. The testing results are monitored by hooking relevant WebView callbacks. Finally, we verify 26 exploitable TDCAttack, and deliver case studies to demonstrate their security impacts, including phishing, privacy leakage, and privilege escalation.

Contributions. We sumarize our contributions below.

- We conduct the first security analysis of the allowlist mechanism within the app-in-app ecosystem and propose a novel security issue of TDCAttack.
- To assess its security hazards in the real world, we design and implement an automated analysis framework for identifying vulnerable mini-apps susceptible to TDCAttack.
- Through an evaluation encompassing 11,838 mini-apps, we successfully identify 26 exploitable mini-apps and confirm various security consequences, including phishing, privacy leakage, and privilege escalation.

2 UNDERSTANDING TDCATTACK

In this section, we first give an overview of the allowlist mechanism in the app-in-app ecosystem. Then we present the details of TDCAttack, with two discovered attack vectors.

2.1 Overview of allowlist mechanism

By reverse engineering the allowlist mechanism implementation in three most popular super-apps, including WeChat, Alipay, and Baidu, we generalize their workflows as illustrated in Figure 1.

When the user clicks on a deeplink specifying the super-app protocol and a mini-app ID, the super-apps loads the mini-app from its server into the WebView-based runtime. Furthermore, the web URL can be conveyed to the mini-app in a key-value pair query string, which will be parsed and loaded by the mini-app. Before successfully loading the web content, the super-app performs a security check on the URL based on the domain allowlist configured by the mini-app developer. Only URLs that match the allowlist are permitted to load. Once loaded, the web content can invoke runtime APIs provided by the super-app, which can access the user data (e.g., saved user token, phone number) and system resources.

Furthermore, we categorize the allowlist mechanism into the following two types based on the location of the allowlist security checks:

- App-Side Check: This check is performed on the client-side by super-app before a mini-app loading a web URL. Super-app intercepts such loading request, and match target URL with corresponding allowlist configuration. This configuration is distributed dynamically from the super-app server.
- Server-Side Check: This check is performed on the super-app server-side. In this way, the super-app sends the URL to be loaded as a request parameter to its server and then determines whether to allow loading based on the response.

2.2 Threat Model

In this section, we describe the threat model adopted in TDCAttack. Specifically, the mini-apps are benign and can receive and load external URLs. We consider the remote web attackers as the TDC attackers. They aim to abuse critical functionalities or steal sensitive data in the mini-apps by exploiting vulnerabilities in their trusted domain assets. The attackers have the capability to conduct domain allowlist extraction and domain assets vulnerability scanning on the target mini-app within their local environment. Then they craft malicious URLs with attack payloads and send them to the victim user. When the victim user clicks the URLs, they can evade the allowlist checks and the embedded malicious code can be successfully executed inside the target mini-app's runtime.

2.3 TDCAttacks

The effectiveness of the allowlist mechanism depends on the security and reliability of domain assets included in the allowlist. However, the security of these assets is determined by the website developers, beyond the control of mini-app developers. TDCAttack exploits the design flaw created when mini-app developers incorrectly place their security trust in website developers. By investigating the top OWASP web vulnerabilities [2], we identified 4 types of vulnerabilities, as shown in Table 1, that can be used for TDCAttack from two attack vectors as follows:

Atatck Vector #1: The allowlisted domains can be abused by attackers. This vector consists of CWE-79 and CWE-601:

- **Subdomain takeover**: An attacker can gain control over a subdomain that was previously associated with a legitimate service but is no longer in use or properly configured. This can happen if the subdomain's DNS (Domain Name System) record points to a service that has been deactivated, expired, or moved, leaving it vulnerable to takeover by attackers.
- **Expired domain**: When a domain's registration expires and its owner doesn't renew it, attackers can preempt the expired domain and host malicious content. If the mini-app's allowlist isn't updated promptly, URLs under the attacker-controlled domain can bypass the allowlist security check.

Attack Vector #2: Exploitable vulnerabilities in allowlisted web pages. This vector consists of CWE-16 and CWE-672:

- **XSS**: If an allowlisted URL contains an XSS vulnerability, attackers can inject malicious scripts into the web page. When this compromised web page is loaded into the mini-app's WebView, the injected malicious code is executed, allowing the attacker to utilize the mini-app's identity to invoke runtime APIs provided by the super-app.
- **Open redirect**: If an allowlisted URL has an open redirection vulnerability, attackers can exploit it to craft malicious links that redirect users to their controlled web pages. This attack can succeed because the allowlist checks solely focus on preventing untrusted web resources from being loaded into the WebView and do not affect the web page routing within the WebView.

2.4 Related Work

Mini-app Studies. Recent studies reveal the model of mini-app ecosystems, and their various advantages in different aspects of social life, including health, education, government, and marketing [8, 9, 11, 16, 19, 22, 23, 25, 35]. Additionally, some studies provide program analysis technique by leveraging as dynamic analysis [12, 17], and static analysis [15, 24]. In the domain of research on attacks and defense in app-in-app ecosystems, a few pioneering studies [18, 29, 31, 32, 34] have delved into the defense mechanisms and vulnerabilities of these ecosystems. Specifically, Lu et al. [18] and Zhang et al. [31] investigate the permission inconsistency problem, and Zhang et al. [32] proposed the novel identity confusion flaws in protecting runtime APIs. Zhang et al. [34] and Yang et al. [29] measure the security prevalence of information leakage. As a comparison, our paper focuses on the prevalent injection vulnerabilities and their root causes to the allowlist mechanism, which has not been extensively studied before.

Content Security. There exists several techniques that support dynamic content loading and execution in platforms such as browsers, cross-platform apps, and hybrid apps. Numerous studies [13, 14, 20, 26–28, 30] discussed the threats and attacks against code integrity. Richards et al. [20], Weichselbaum et al. [26], and Yue et al. [30] target the web applications running in web browsers. Jin et al. [14], Xiao et al. [27], and Jin et al. [13] target the cross-platform apps and hybrid apps, who can steal sensitive information or abusing system resources. While TDCAttacks belong to this class of security threats, our paper presents different contributions from existing research because the defense model in app-in-app ecosystems is different from the previous work, and bringing new challenges and threats. Our paper present the first study on the root cause of TDCAttacks and their prevalence. Our proposed analysis framework can suits for various app-in-app ecosystems despite their different allowlist implementation.

3 TDCATTACK: AN EMPIRICAL STUDY

In this section, we present the first framework for analyzing the security of real-world mini-app allowlist deployments. Figure 2

serial number	name	CWE	Security introduction	Security outcome
1	Cross-site scripting	79	malicious scripts are injected into trusted websites.	code injection
2	Open redirect	601	websites allows a user to control a redirect to another URL.	malicious page load
3	Subdomain takeover	16	registering an existing domain name to control the domain.	control domain
4	Expired domain	672	the owner has not renewed by the expiration date.	control domain

Table 1: 4 types of vulnerabilities that could be exploited for TDCAttack

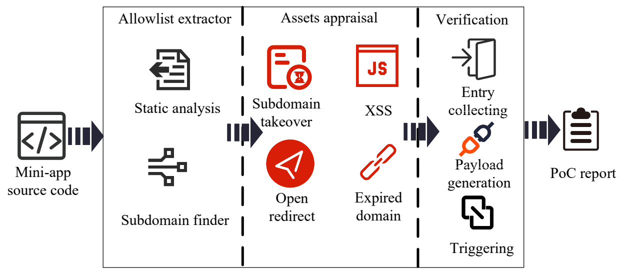

Figure 2: Overview of the methodology.

shows the overview of our methodology, comprising the following three steps:

3.1 Step I: Allowlist Extractor

In this step, we design an algorithm to identify domains that are configured in the allowlist of a given mini-app. The high-level idea is that we can extract the loaded URLs from the mini-app's source code and then employ them as the seeds for expansion via subdomain discovery techniques. The insight here is that the domains of URL accessed during the normal operation of a mini-app are included within the allowlist. We observe that in many cases the domain allowlist checks enforced in the mini-apps are coarse. For instance, the function "endswith()" is frequently utilized for string comparison. Thus, the subdomain assets of these URLs can also pass the allowlist checks.

Specifically, we perform static data-flow analysis on the source code of a given mini-app, which can be obtained leveraging MiniCrawler [33]. As illustrated in algorithm 1, we begin by identifying all URL loading statements (i.e., loadpoints) in the mini-app, based on their invocation patterns learned from developer documentation (Line 4-5). Then we backwardly trace the source value of loaded URLs and save them in a domain list denoted as S_{dom} (Line 6-8). This analysis is implemented on TaintMini [24]. Furthermore, due to the domain matching operation is often broad, e.g., using the `endswith()` API, we extend the S_{dom} to E_{dom} (Line 12-15) using OneForAll [4], a popular open-sourced subdomain finder tool. Through this process, we can ultimately derive a comprehensive and accurate representation of the domains that might be included in the allowlist.

3.2 Step II: Assets Appraisal

In the second step, we identify unsafe domains and domain assets that suffer from the aforementioned two attack vectors. Specifically, for the domains identified in step I, we perform security

Algorithm 1: Infer allowlist from mini-app source code

```
Input: mini-app source code $s$
Output: Inferred allowlist domains $E_{dom}$
1  Initialize $S_{dom}$, $E_{dom}$ as empty sets;
2  // Use static data-flow analysis to trace the URLs loaded in
     the web-view component;
3  for each WebView in s do
4      loadpoints ← collect(WebView);
5      for each loadpoint in loadpoints do
6          url ← backtrace(loadpoint);
7          $S_{dom} \leftarrow S_{dom} \cup \{extract - domain(url)\}$;
8      end
9  end
10 // Use a sub-domain finder to expand the URLs in $S_{dom}$;
11 for each domain in $S_{dom}$ do
12     subdomains ← subdomain-finder(domain) ;
13     $E_{dom} \leftarrow E_{dom} \cup$ subdomains;
14 end
15 return $E_{dom}$
```

analysis with the help of several popular web vulnerability detection tools, including Xray [7] for detecting websites vulnerable to XSS and open redirect, Aquatone [1] for identifying domains vulnerable to subdomain takeovers, and whois [6] for detecting expired domains. As previously mentioned, super-apps may implement customized defenses against web vulnerabilities such as XSS. To understand the potential attack vectors within a specific app-in-app ecosystem, we perform penetration testing within our custom-built mini-apps for WeChat, Alipay, and Baidu. In detail, we manually configure a vulnerable website and try to load the URL in the mini-app. If the payload is successfully loaded and the embedded JavaScript code is executed, we classify the super-app as susceptible to a specific attack.

3.3 Step III: Security Impact Verification

In the third step, we use template-based dynamic testing to verify whether a mini-app can be impacted by vulnerable domain assets. Specifically, we first identify an exploitable path in target mini-apps through a cross-context static analysis from the mini-app's lifecycle function (e.g., `onLoad`) to the URL loading points, to analyze the process of external input parsing (typically key-value pairs). Then we extract the processed keys and combine the exploits created in the last step to generate the attack payloads in a form like scheme://host/path?appId=[appID]&page=[page]?[key]

Step	Consuming time
Allowlist Extractor	3 min per mini-app
Assets Appraisal	2.5 min per domain
Verification	32 s per mini-app

Table 2: Efficiency of our security analysis framework

Step	Result	TP(%)	FP(%)
Allowlist Extractor	81978 domains	97.8	2.2
Assets Appraisal	1526 vulnerable urls	91.7	8.3
Verification	26 vulnerable mini-apps	100	0

Table 3: Effectiveness of our security analysis framework

Vulnerability name	Platform		
	WeChat	Alipay	Baidu
Cross-site scripting	√	★	√
Open redirect	×	√	√
Subdomain takeover	√	√	√
Expired domain	√	√	√

Table 4: Effectiveness of 4 attacks on mainstream super-apps. Symbol "√" means the super-app is vulnerable, "×" means the super-app has customized protection, "★ means the iOS version of super-app is vulnerable.

=[exploits]. We use the Android Debug Bridge (adb) to launch the target mini-apps carrying the generated payload. Besides, we implement a feedback monitor by instrumenting relevant WebView callbacks, which enables us to determine whether the attack payloads are successfully executed or not.

4 EVALUATION

In this section, we present the overall analysis results, including efficiency and effectiveness of our analysis framework, and verified security impacts on popular super-app platforms.

4.1 Efficiency & Effectiveness

Table 2 shows the efficiency of each step. We performed our evaluation on a laptop with an Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz processor and 24G bytes RAM, Windows 11 Operating System. During the experiment, the tool can log the consuming time for each step.

Table 3 lists the results of each step in our security analysis, including the true positive rate, and false positive rate. Specifically, we analyzed 11,838 mini-apps in total, including 4,446 mini-apps from WeChat, 3,946 from Alipay, and 3,446 from Baidu. As a result, we successfully extracted 60,761 domains from mini-apps' source code and extended them to 81,978 domains. In the security-oriented experiments, we found problematic allowlisted domain assets in 140 mini-apps, including expired domain names in 7 mini-apps, XSS vulnerability in 135 mini-apps, and open redirect vulnerability in 3 mini-apps. We randomly selected 10 mini-apps from each super-app platform (30 in total) and manually verified their detection results. We found that the true positive rate in each step is high, including 97.8% in the first step, 91.7% in the second step, and 100% in the third step. For the false positives, we further analyzed their root causes and the details are presented below.

In Step I (allowlist extractor), false positives primarily arise due to version updates. Specifically, our mini-app dataset includes versions prior to December 2022, and their allowlist configurations may have been altered in subsequent releases. For instance, a mini-app might modify its third-party advertiser, rendering certain domains obsolete in newer versions. In step II (assets appraisal), the false positives are caused by the limited trigger condition under the mini-app context and the inaccuracy of the adopted vulnerability detection tool. For instance, XSS vulnerabilities can be exploited by crafting payload in HTTP query, header, and body in web attacks. However, the attack vectors in mini-apps are limited and the XSS payload can only be embedded in HTTP query strings. This type of false positive can be filtered easily. Besides, the adopted web vulnerability detection tool can introduce false positives too.

Furthermore, we manually verified their attack consequences in 3 popular super-apps. Table 4 shows the impact of different types of vulnerabilities across them. In detail, all of the super-apps are vulnerable to TDCAttack. Although Alipay enforces the XSS protection inside its customized WebView kernel, its iOS version that uses WKWebView is still vulnerable to our attack.

4.2 Case Study

Now, we illustrate one real-world case study to demonstrate how a TDC attack is discovered and performed.

Case Study: Exploit Alipay mini-app through XSS vulnerability. Take 'mini-appA', an online car quotation platform in Alipay, to illustrate the process of TDCAttack and the security consequences. First, the allowlist extractor successfully discovers 20 loaded URLs from its source code and expands them to 705 domains. Then, we assess the security of these domains by feeding them to OneForAll [4]. As a result, we find a subdomain 'sub.benign.com' contains a webpage with XSS vulnerability. That is, malicious code can be injected into the mini-appA's runtime when *https://sub.benign.com?source=[XSS-payload]* is loaded. Hence, an attacker can craft a phishing link like this:

```
https://ds.alipay.com/?scheme=alipays://platformapi/startapp?appId=
    ↪ [mini-appA]&page=pages/webview?h5_url=
    ↪ https://sub.benign.com?source=[XSS-payload]
```

The above link can be easily distributed to victim users through social media such as Twitter and Sina Weibo. When the victim user clicks it, the "mini-appA" will start and load the `h5_url` (i.e., https://sub.benign.com?source=[XSS-payload]), executing malicious code in the XSS-payload. Note that, as "mini-appA" saves the user's login token in its storage, the malicious code can steal the saved user token by leveraging the `getStorage` runtime API, leading to account takeover and information leakage.

We verify the attack on Android and iOS respectively. We find that although Alipay has implemented security checks in its customized WebView kernel for mitigating XSS attacks, they only function on the Android system, leaving iOS users exposed to TDCAttack. Note that, this case is hardly detected through manual auditing. It is because the vulnerable URL under sub.benign.com is only designed for PC users and is not supposed to be used and loaded in “mini-appA“. However, security analysis framework can successfully uncover it.

5 DISCUSSION

The goal of this study is to present the TDC attack surface on the mini-apps allowlist mechanism, and we mainly consider popular vulnerabilities listed in the top OWASP web vulnerabilities. Beyond these, there are many interesting and new domain vulnerabilities such as image-hosting domain [10] that can be used for exploiting the vulnerable domain assets and launching TDC attacks. We leave covering these cases for future research.

Contermeasures. To prevent TDC attacks, we propose several mitigation strategies based on our analysis. First, super-apps and mini-apps should enforce a standard and atomic allowlist check mechanism such as Content Security Policies [21]. They should carefully choose the trusted domain assets following the strict definition of scheme, host, paths, and even content types. Second, the mitigation of TDC attacks can also benefit from real-time code integrity protection, as proposed in other cross-platform apps [27].

6 CONCLUSION

In this paper, we introduced the novel TrustedDomain Compromise Attack observed in real-world app-in-app ecosystems. We classified TDCAttack into two primary attack vectors, enabling attackers to manipulate unsafe allowlisted domains and URLs to bypass security checks. Such attacks can lead to phishing or the abuse of sensitive runtime APIs, including access to storage files of victim mini-apps. Furthermore, we conduct the first empirical study of TDCAttack in the real-world app-in-app ecosystems. By developing a security analysis framework, we measured the TDCAttacks in mini-apps within three prominent super-apps. Our experiments demonstrate that our framework effectively detects mini-apps susceptible to TDCAttack, maintaining a low rate of false positives. Notably, we identified 26 exploitable real-world mini-apps. We are confident that our contributions will significantly enhance the security design and implementation of app-in-app ecosystems.

ACKNOWLEDGMENTS

We would like to thank our shepherd Yue Zhang and the anonymous reviewers for their insightful comments that helped improve the quality of the paper. This work was supported in part by the National Key Research and Development Program (2021YFB3101200), National Natural Science Foundation of China (62172104, 62172105, 61972099, 62102093, 62102091). Yuan Zhang was supported in part by the Shanghai Rising-Star Program under Grant 21QA1400700 and the Shanghai Pilot Program for Basic Research - FuDan University 21TQ1400100 (21TQ012). Min Yang is the corresponding author, and a faculty of Shanghai Institute of Intelligent Electronics & Systems and Engineering Research Center of Cyber Security Auditing and Monitoring.

REFERENCES

[1] 2019. *AQUATONE - A Tool for Domain Flyovers.* Retrieved July 20, 2023 from https://github.com/michenriksen/aquatone#installation
[2] 2021. *OWASP Top 10 - 2021.* Retrieved July 20, 2023 from https://owasp.org/Top10/
[3] 2023. *Content Security Policy (CSP).* Retrieved August 9, 2023 from https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
[4] 2023. *OneForAll.* Retrieved July 20, 2023 from https://github.com/shmilylty/OneForAll
[5] 2023. *WebView.* Retrieved August 9, 2023 from https://developer.android.com/reference/android/webkit/WebView
[6] 2023. *whois | Kali Linux Tools.* Retrieved July 20, 2023 from https://www.kali.org/tools/whois/
[7] 2023. *xray.* Retrieved July 20, 2023 from https://github.com/chaitin/xray
[8] Xin Chen, Xi Zhou, Huan Li, Jinlan Li, and Hua Jiang. 2020. The value of WeChat as a source of information on the COVID-19 in China. *Preprint]. Bull World Health Organ* 30 (2020).
[9] Ao Cheng, Gang Ren, Taeho Hong, Kichan Nam, and Chulmo Koo. 2019. An exploratory analysis of travel-related WeChat mini program usage: affordance theory perspective. In *Information and Communication Technologies in Tourism 2019: Proceedings of the International Conference in Nicosia, Cyprus, January 30–February 1, 2019.* Springer, 333–343.
[10] Pei Chen Xiaojing Liao Guoyi Ye Geng Hong, Mengying Wu and Min Yang. 2023. Understanding and Detecting Abused Image Hosting Modules as Malicious Services. In *2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23).* ACM.
[11] Lei Hao, Fucheng Wan, Ning Ma, and Yicheng Wang. 2018. Analysis of the development of WeChat mini program. In *Journal of Physics: Conference Series*, Vol. 1087. IOP Publishing, 062040.
[12] Jiajun Hu, Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2023. *w*Test: WebView-Oriented Testing for Android Applications. *arXiv preprint arXiv:2306.03845* (2023).
[13] Xing Jin, Xunchao Hu, Kailiang Ying, Wenliang Du, Heng Yin, and Gautam Nagesh Peri. 2014. Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 66–77.
[14] Zihao Jin, Shuo Chen, Yang Chen, Haixin Duan, Jianjun Chen, and Jianping Wu. 2023. A Security Study about Electron Applications and a Programming Methodology to Tame DOM Functionalities.. In *NDSS.*
[15] Wei Li, Borui Yang, Hangyu Ye, Liyao Xiang, Qingxiao Tao, Xinbing Wang, and Chenghu Zhou. 2023. MiniTracker: Large-Scale Sensitive Information Tracking in Mini Apps. *IEEE Transactions on Dependable and Secure Computing* (2023).
[16] Qinzhen Liang and Chengyang Chang. 2019. Construction of teaching model based on WeChat Mini-Program. *International Journal of Science* 16, 1 (2019), 54–59.
[17] Yi Liu, Jinhui Xie, Jianbo Yang, Shiyu Guo, Yuetang Deng, Shuqing Li, Yechang Wu, and Yepang Liu. 2020. Industry practice of javascript dynamic analysis on wechat mini-programs. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering.* 1189–1193.
[18] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang Wang. 2020. Demystifying resource management risks in emerging mobile app-in-app ecosystems. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications Security.* 569–585.
[19] Qianhui Rao and Eunju Ko. 2021. Impulsive purchasing and luxury brand loyalty in WeChat Mini Program. *Asia Pacific Journal of Marketing and Logistics* 33, 10 (2021), 2054–2071.
[20] Gregor Richards, Christian Hammer, Brian Burg, and Jan Vitek. 2011. The eval that men do: A large-scale study of the use of eval in javascript applications. In *ECOOP 2011–Object-Oriented Programming: 25th European Conference, Lancaster, Uk, July 25-29, 2011 Proceedings 25.* Springer, 52–78.
[21] Sid Stamm, Brandon Sterne, and Gervase Markham. 2010. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web.* 921–930.
[22] Yiling Sui, Tian Wang, and Xiaochun Wang. 2020. The impact of WeChat app-based education and rehabilitation program on anxiety, depression, quality of life, loss of follow-up and survival in non-small cell lung cancer patients who underwent surgical resection. *European Journal of Oncology Nursing* 45 (2020), 101707.
[23] Zhenya Tang, Zhongyun Zhou, Feng Xu, and Merrill Warkentin. 2022. Apps within apps: predicting government WeChat mini-program adoption from trust–risk perspective and innovation diffusion theory. *Information Technology & People* 35, 3 (2022), 1170–1190.

[24] Chao Wang, Ronny Ko, Yue Zhang, Yuqing Yang, and Zhiqiang Lin. 2023. TAINT-MINI: Detecting Flow of Sensitive Data in Mini-Programs with Static Taint Analysis. In *Proceedings of the 45th International Conference on Software Engineering*.

[25] Feilong Wang, Lily Dongxia Xiao, Kaifa Wang, Min Li, and Yanni Yang. 2017. Evaluation of a WeChat-based dementia-specific training program for nurses in primary care settings: A randomized controlled trial. *Applied Nursing Research* 38 (2017), 51–59.

[26] Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. 2016. Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1376–1387.

[27] Feng Xiao, Zheng Yang, Joey Allen, Guangliang Yang, Grant Williams, and Wenke Lee. 2022. Understanding and Mitigating Remote Code Execution Vulnerabilities in Cross-platform Ecosystem. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2975–2988.

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

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

[30] Chuan Yue and Haining Wang. 2009. Characterizing insecure JavaScript practices on the web. In *Proceedings of the 18th international conference on World wide web*. 961–970.

[31] Jianyi Zhang, Leixin Yang, Yuyang Han, Zixiao Xiang, and Xiali Hei. 2023. A Small Leak Will Sink Many Ships: Vulnerabilities Related to mini-programs Permissions. In *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 595–606.

[32] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. 2022. Identity confusion in {WebView-based} mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX Security 22)*. 1597–1613.

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

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

[35] Kaina Zhou, Wen Wang, Wenqian Zhao, Lulu Li, Mengyue Zhang, Pingli Guo, Can Zhou, Minjie Li, Jinghua An, Jin Li, et al. 2020. Benefits of a WeChat-based multimodal nursing program on early rehabilitation in postoperative women with breast cancer: a clinical randomized controlled trial. *International journal of nursing studies* 106 (2020), 103565.