Cross Miniapp Request Forgery: Root Causes, Attacks, and Vulnerability Detection

Yuqing Yang
The Ohio State University
yang.5656@osu.edu

Yue Zhang
The Ohio State University
zhang.12047@osu.edu

Zhiqiang Lin
The Ohio State University
zlin@cse.ohio-state.edu

ABSTRACT

A miniapp is a full-fledged app that is executed inside a mobile super app such as WeChat or SnapChat. Being mini by nature, it often has to communicate with other miniapps to accomplish complicated tasks. However, unlike a web app that uses network domains (*i.e.*, IP addresses) to navigate between different web apps, a miniapp uses a unique global `appId` assigned by the super app to navigate between miniapps. Unfortunately, any missing checks of the sender's `appId` in a receiver miniapp can lead to a new type of attacks we name it cross-miniapp request forgery (CMRF). In addition to demystifying the root cause of this attack (*i.e.*, the essence of the vulnerability), this paper also seeks to measure the popularity of this vulnerability among miniapps by developing CmrfScanner, which is able to statically detect the CMRF-vulnerability based on the abstract syntax tree of miniapp code to determine whether there are any missing checks of the `appIds`. We have tested CmrfScanner with 2,571,490 WeChat miniapps and 148,512 Baidu miniapps, and identified 52,394 (2.04%) WeChat miniapps and 494 (0.33%) Baidu miniapps that involve cross-communication. Among them, CmrfScanner further identified that 50,281 (95.97%) of WeChat miniapps, and 493 (99.80%) of Baidu miniapps lack the `appID` checks of the sender's mini-apps, indicating that a large amount of miniapp developers are not aware of this attack. We also estimated the impact of this vulnerability and found 55.05% of the lack of validation WeChat miniapps (7.09% of such Baidu miniapps) can have direct security consequences such as privileged data access, information leakage, promotion abuse, and even shopping for free. We hope that our findings can raise awareness among miniapp developers, and future miniapps will not be subject to CMRF attacks.

CCS CONCEPTS

• **Security and privacy** → **Web application security**; **Mobile and wireless security**.

KEYWORDS

Mobile security; mobile super apps; miniapp security; input validation

CCS '22, November 7–11, 2022, Los Angeles, CA, USA

ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560597

ACM Reference Format:
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 (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560597

1 INTRODUCTION

Increasingly, mobile super apps from WeChat [34], Baidu [38], and TikTok [28] to recently Snapchat [4] all have been exploring new ways to utilize their huge user base and extend their functionalities to make more profit. One such way is through a novel paradigm called miniapp, debuted by WeChat in 2017 [2], in which a light-weight and full-fledged app is executed inside a JavaScript engine created (or virtualized) by the host app [42]. With easy access and install-less features, along with a myriad of daily-life services ranging from ride-hailing to online-shopping, the miniapp paradigm has rapidly gained momentum among users, bringing a huge amount of opportunities for both super app vendors and third-party developers. For example, with a massive monthly active users of miniapps reaching up to 1.29 billion [5] as in the first quarter of 2022, and WeChat miniapps alone have accumulated a total transaction value of 2.72 trillion RMB in 2021 [6].

Compared to traditional web apps or native mobile apps, miniapps are featured with easy development and distribution. Specifically, unlike web apps that typically require developer maintained back-end servers, miniapps do not require a mandatory back-end from developers [9], and instead there is a set of well-encapsulated APIs to allow easy access to super app maintained back-end data (*e.g.*, user name, gender, and home address) and system resources (*e.g.*, Bluetooth, GPS, and cameras) [32]. In addition to being distributed through app stores (much like how native mobile apps are distributed), miniapps can also be distributed by utilizing the super app user's social network via chat messages, user's moment posts, or even just a QR code, once the miniapp has passed the vetting by the super app. Using a new miniapp is only one click away [42], and there is no need to install or uninstall it (*i.e.*, install-less). As such, the miniapp paradigm has created a triple-win situation: developers reduce their development cost, the users have convenient access to a variety of services, and the platform gains popularity, user's stickiness, and more profit [34, 42, 45].

On the other hand, similar to traditional web apps or native apps, miniapps often also need to work together to complete a sophisticated task. For example, a shopping miniapp may need to communicate with a payment miniapp with additional information such as the order ID and price information to finish a purchasing transaction. Such a cross-communication is particularly important

for miniapps to enrich their functionalities, since a single miniapp usually has fewer functionalities when compared with native apps due to its size limit (*e.g.*, the size of a WeChat miniapp cannot exceed 12 MB [8]). Additionally, the miniapps may not only communicate with their back-ends as in traditional web apps, but also need a similar cross-miniapp communication in their front-ends as in native apps, much like the traditional inter-process communication (IPC). Therefore, the super app has invented such a communication that resembles the Android Intent mechanism, to allow a miniapp to redirect to other miniapps. Meanwhile, when a miniapp initiates a redirection request to another one, it can set specific data to be transmitted to others by setting a special native-app-intent-alike JSON object called `extraData` [19].

However, during the cross-communication (including both the back-end channel and the front-end channel), given privacy-sensitive data transmitted and the privileged capabilities the communicating miniapps (particularly the receiver miniapps) may have, the super app needs to add strict security policies and mechanisms to protect the data during the transmission and consumption. As such, super apps such as WeChat have enforced the mandatory HTTPS protocol for miniapps to communicate with their back-ends. However, the enforcement for the front-end secure communication is missing (*e.g.*, no integrity check of the origin of the sender miniapp). Meanwhile, we also find that the communication is often miniapp-specific, and hard to be secured from the super app's perspective. For instance, a shopping miniapp needs to be notified when the payment miniapp has finished processing its payment. However, when the shopping miniapp receives the feedback from the payment miniapp, it is the responsibility of the shopping miniapp instead of the super app to check whether the feedback is indeed from an expected miniapp. As such, it is the miniapp's responsibility to check the message authenticity and integrity.

Therefore, any missing check of the identity (*i.e.*, the `appId`) of the sender miniapp at a receiver miniapp (particularly for those receiver miniapps with privileged capabilities) can allow attackers to forge fake requests and inject them into the receiver miniapp. We name such an attack Cross-Miniapp Request Forgery (CMRF), which can lead to various security breaches. For instance, as we have demonstrated in this paper, the attacker can achieve "shopping for free" by injecting a payment success message to trick a shopping miniapp into believing that the payment has been succeeded; the attacker can hack into arbitrary webcams to see the video streams; and the attacker can also login into arbitrary users' accounts by injecting a fabricated account identifier for promotion abuse.

To understand the prevalence and impact of CMRF attacks among miniapps, we have developed CmrfScanner, a static analysis tool to detect the vulnerabilities among miniapps, by detecting whether there are any missing checks of the source miniapp's ID in cross-miniapp communication based on the abstract syntax tree of the miniapp code. Our evaluation with 2,571,490 WeChat miniapps and 148,512 Baidu miniapps revealed a worrying situation: 95.97% of the cross-communicated WeChat miniapps, and 99.80% of the cross-communicated Baidu miniapps have failed to perform the security checks, indicating that a large number of miniapp developers are not aware of our CMRF attacks. While the attacker can inject arbitrary messages into the receivers that do not perform `appId` checks, those receivers may not have permissions to perform privileged actions (*e.g.*, controlling devices or buying products). As such, we have also estimated the potential impact of the vulnerabilities based on the use of mission critical APIs in the victim miniapps. Among these miniapps that do not perform `appId` checks, our API-based impact analysis estimates that nearly 55.05% wechat miniapps and 7.09% baidu miniapps can have direct security consequences.

Resources	Protection P	Protection I	Protection V	Protection A	Permissions (if applicable)	Example APIs
Front-end Guarded Resources						
Location	✓				scope.userLocation	`getLocation`
Audio	✓				scope.record	`RecorderManager.start`
Bluetooth	✓				scope.Bluetooth	`getBluetoothDevices`
Camera	✓				scope.camera	`CameraContext.takePhoto`
Media	✓				scope.writePhotosAlbum	`saveImageToPhotosAlbum`
WeRun	✓				scope.werun	`getWeRunData`
UserInfo	✓				scope.userinfo	`getUserProfile`
Address	✓				scope.address	`chooseAddress`
Invoice	✓				scope.invoice	`getInvoice`
InvoiceTitle	✓				scope.invoiceTitle	`getInvoiceTitle`
File		✓			N/A	`FileSystemManager.saveFile`
Data Cache		✓			N/A	`setStorageSync`
Back-end Guarded Resources						
Payment			✓		N/A	`requestPayment`
AccountInfo			✓		N/A	`getAccountInfoSync`
Coupon			✓		N/A	`openCard`
PhoneNumber			✓		N/A	`getPhoneNumber`
Socket				✓	N/A	`createTCPSocket`
HTTPS				✓	N/A	`wx.request`

Table 1: Summary of resources and their protection mechanism. P: Permission, I: Isolation; V: Vetting, A: Allowlisting.

The APIs provided by the super apps, a miniapp can access rich resources (*e.g.*, location and Bluetooth). Since a large body of these resources are privacy sensitive, the host app as well as the underlying OS must protect the resources properly. Note that in the rest of this paper, we use the host app to denote the app that provides the run-time environment to the miniapps,

Contributions. In short, we make the following contributions:

- **Novel Attacks** (§3 and §7). We are the first to study the security issues of cross-miniapp communication, and discover the CMRF attacks, which allow attackers to inject a forged request to a vulnerable miniapp leading to various security consequences such as privileged data access, information leakage, or even shopping for free.
- **Efficient Detection** (§4). To identify the vulnerable miniapps, we present CmrfScanner, an open-source, static data flow analysis based tool to analyze the AST nodes of the miniapps and detect the uses of the requests to identify those miniapps without the validation checks of the `appID`s.
- **Empirical Evaluation** (§6). With CmrfScanner, we have identified 50,281 (95.97%) vulnerable miniapps out of 52,394 miniapps from the WeChat market, and 493 (99.80%) vulnerable miniapps out of 494 miniapps that used the cross-miniapp channel from the Baidu market. Our API-based impact analysis further estimates that 55.05% of these WeChat miniapps and 7.09% Baidu miniapps can have direct security impact.

2 BACKGROUND

2.1 The Resource Management of Miniapps

and the super app to denote the whole app including both the miniapp and the host app. At a high level, the protection can be classified into permission-based, isolation-based, allowlist-based, and vetting-based. As shown in Table 1, based on where the resources are located, we describe how they are guarded in the following.

- **Front-end Guarded Resources.** A front-end resource can be protected by either the permission mechanism from both the host apps and the underlying OS, or just the isolation mechanism from the OS. To be more specific, the front-end of a super app can use the permission mechanism to check whether a given miniapp has the corresponding permissions when accessing a protected resource. For instance, WeChat provides access for miniapps to both the OS-level resources (*e.g.*, location, Bluetooth) and information collected from the users (*e.g.*, mailing address) [1]. Meanwhile, the host apps can also use the isolation mechanism provided by the OS to protect resources. For example, the host apps can create an isolated space for each miniapp to prevent its files (*e.g.*, configuration files) from being accessed by other unauthorized apps or miniapps.
- **Back-end Guarded Resources.** The back-end of the super app also need to protect the resources from being accessed by unauthorized apps. To this end, the super-apps will first verify the authenticity and the corresponding permissions of the developers, and then grant them the access once they have passed the checks. For example, API `requestPayment` [7] is not available to individual developers but to enterprise developers. To use such APIs, the enterprise developers need to submit their business licenses to the super app providers for the vetting, and can only use those APIs when they obtain the approval. Additionally, the super apps' back-end can use allowlist to protect the network resources [3]. Specifically, the miniapps cannot access arbitrary web domains, but only a list of domains that are trusted by their super apps (*e.g.*, websites that do not use HTTPS are not allowed to be accessed by miniapps). Sometimes, the miniapp developers can require the super apps to expand the allowlist by providing the domains that they attempt to access for approval.

2.2 Cross-miniapp Communications

Similar to mobile apps or web apps, the miniapps can also communicate with each other to complete a sophisticated task. Particularly, a miniapp can send a cross-miniapp request by redirecting to another miniapp identified by its miniapp ID (e.g, `appID` in WeChat, and `AppKey` in Baidu; note that we will just use `appID` in the rest of the paper) assigned by the super apps, which is similar to sending an Android Intent in Android apps [16] or initiating a URL request in web apps. In the following, we describe its detailed workflow.

As shown in Figure 1, two miniapps need to go through the three stages with eight steps to achieve their cross-communication: (*i*) request creation and sending, where a sender miniapp first creates a request and invokes the communication API, (*ii*) channel establishment, where the host app receives the request, and creates a shared-memory-alike channel to save the request for the receiver miniapp to consume, and (*iii*) request receiving, where the receiver fetches the request from the channel. Next, we explain each step in greater details (we use WeChat as an example in this section, while other super apps such as Baidu have similar implementations):

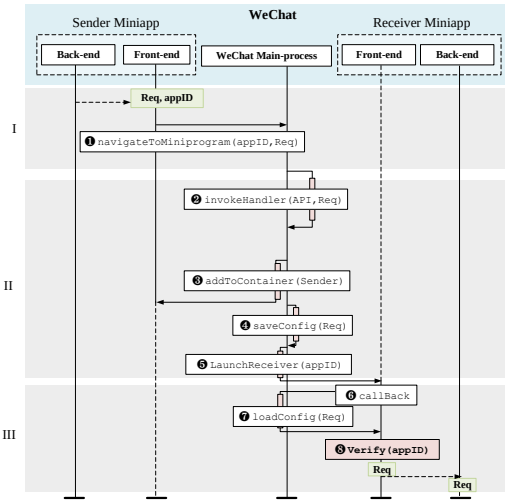

Figure 1: Illustration of Cross-MiniApp Communication

(I) **Request Creation and Sending.** Assume a shopping miniapp needs to communicate with a payment miniapp to complete a payment transaction. After the user presses the purchase button, the shopping miniapp first needs to get the price and then generate an order ID, and then invokes the inter-miniapp communication API (*e.g.*, `wx.navigateToMiniProgram`) (**Step ❶**). For example, as shown in Figure 2, the communication API has three parameters: (*i*) the receiver's miniapp ID (*i.e.*, `appId` "`wx2d495bf4b2abdecef`"), (*ii*) the path URL that the sender attempts to send the requests (note that a receiver can have more than one path URL, each with different functionalities), and (*iii*) the request, typically in the format of `extraData`. As shown in Figure 2, the sender miniapp uses `extraData` to transfer the price and the order ID. Specifically, `extraData` is a JSON object that contains the name of the request and the value of the request (*e.g.*, `extraData:{Price:price}` in line 8).

(II) **Channel Establishment.** When the cross communication API is invoked from a miniapp, it first notifies the host app of the ongoing communication request (**Step ❷**). For example, when WeChat receives this request, it pushes the sender into a run-time stack (called `AppBrandRuntimeContainer`), which maintains a few miniapps that have been launched so far (**Step ❸**). In particular, when a miniapp in the run-time stack enters an inactive state with its running resources unreleased (WeChat does not allow a miniapp to run in the background), WeChat will release the resources of the miniapp located at the bottom of the stack if the miniapp stays inactive for more than 5 minutes. This is because the host app itself (*e.g.*, WeChat) is still a native app, which has relatively limited resources when compared with an Operating System (*e.g.*, Android). It cannot allow a miniapp to continue to launch (or run) or consume unlimited resources. As such, creating a shared-memory-alike space for the two miniapps (which are two independent processes) to communicate is a viable option. The creation of such a shared memory is simple, and WeChat just encapsulates the request

```
// sender (shopping miniapp) ID: wxd7c977843ebe7a64
    submitOrder: function(){
    price = self.getPrice();
    tt.navigateToMiniProgram({
      appId: "wx2d495bf4b2abdecef",
      path: "paymentpage",
      extraData: {
        Price: price
        orderID: orderid,
      }
    });
  }
  onLaunch(o){
    var e=this;
    o.referrerInfo && (e.globalData.paymentState
      = o.referrerInfo.extraData.paymentState) &&
      (e.globalData.couponCode
      = o.referrerInfo.extraData.couponCode)
    if(e.globalData.paymentState == "Success")
    {
        shiptheProducts() //ship the products
    }
    saveCouponCode(e.globalData.couponCode)
  }
```

Figure 2: Code snippet of a (vulnerable) sender miniapp

```
// receiver (payment miniapp) ID: wx2d495bf4b2abdecef
  var e=getApp();
  onLaunch(o){
    o.referrerInfo && (e.globalData.price
      = o.referrerInfo.extraData.Price) &&
      (e.globalData.appId
      = o.referrerInfo.appId)
    e.globalData.orderID = o.referrerInfo.extraData.orderID
  }
  Pay:function() {
    var price = e.globalData.Price
    wx.requestPayment({price, ...}) //pay the order
    if(e.globalData.appId == "wxd7c977843ebe7a64"){
        e.globalData.coupon = 'MYCOUPON'
    }else{
        e.globalData.coupon = null
    }
  }
  wx.navigateBackMiniProgram({
  extraData: {
  paymentState: 'Success',
  couponCode: e.globalData.coupon
}
```

Figure 3: Code snippet of a (secure) receiver miniapp

into an object called `AppBrandInitConfig`, then saves this object for future references (**Step ❹**). Subsequently, WeChat launches the receiver miniapp based on the `appId` specified by the sender (**Step ❺**).

(III) **Request Receiving.** When the receiver miniapp is launched, it first invokes a callback function to fetch the request as shown in Figure 3. The callback function can be `onShow` or `onLaunch`, which will be executed every time when the UI launches (**Step ❻**). The callback function ultimately retrieves the saved `AppBrandInitConfig`, and returns the requests to the receiver running in the foreground (**Step ❼**). Since the sender configures parameters with different names (*e.g.*, "`Price`" and "`orderID`"), the receiver then uses these names to fetch the parameters of interest. The receiver can also check the source of the request by comparing `appId` with its expected one to ensure that the request is from an expected miniapp (**Step ❽**, which is highlighted). For example, as shown on lines 13–17 of Figure 3, the payment miniapp and the shopping miniapp could have collaborations, *e.g.*, the payment miniapp may give some coupons to the user if the payment request is from the portal of the intended shopping miniapp (*e.g.*, whose `appId` is "`wxd7c977843ebe7a64`"). Note that similar to the sender that can send a request to the receiver, the receiver can also send a request back to the sender using the cross-miniapp communication API `wx.navigateBackMiniProgram`. In our example, the receiver notifies the sender of the payment statues and the coupon, and at this moment the receiver now becomes the sender and the original sender becomes the receiver. The workflow is similar to the API `navigateToMiniProgram`, and therefore its detail is omitted for brevity.

3 THE CMRF ATTACKS

Having explained the workflow of cross-miniapp communication, we now present cross-miniapp request forgery (CMRF) attacks, where an attacker uses a malicious miniapp to inject forged request into a receiver miniapp that does not enforce security checks. In particular, we provide details of the CMRF attacks, including the threat model (§3.1), the vulnerabilities (§3.2), and the workflow of the attack (§3.3).

3.1 Threat Model and Scope

Assumptions. As illustrated in Figure 1, there are various parties involved in a cross-miniapp communication such as the host app (*e.g.*, WeChat), the front end of both the receiver and sender miniapps, and their back-ends if any. To make the attack more focused, we first assume that there is no tampering with the host app code. Similarly, we assume the code integrity of the miniapp front-end, and there is no static modification against them either. Additionally, we assume that the back-ends of the miniapps are also trusted, since they are typically out of the reach of attackers (unless they have vulnerabilities to allow unauthorized access, which is out of our focus). The communication between front-end and back-end is also trusted, since this is typically secured via encryption (*e.g.*, HTTPS). Finally, we assume that the phone is trusted (no rooting), but the sender miniapp itself could be untrusted.

The Attacker's Capabilities. The CMRF attack can succeed for at least two reasons. First, the attacker does not have to publish the malicious miniapp in the market (no vetting at all) and only use it in the development environment. Note that currently WeChat does not differentiate the testing environment and production environment, miniapp developers can test their miniapps to cross-communicate with any other miniapps of interest on the market. Fundamentally, WeChat cannot stop this behavior, as developers must have the capabilities to test the cross-communication between miniapps (*e.g.*, testing a shopping miniapp with an officially vetted payment miniapp). Second, even when attackers submit a malicious miniapp for the vetting, WeChat still cannot guarantee to scrutinize this malicious app, as the cross-communication is miniapp-specific and it is completely up to the miniapp receiver to check the `appId`s, not by the vetting.

Scope. The miniapp paradigm has been supported by multiple super apps, including WeChat, TikTok, Baidu, Alipay, and SnapChat. For proof of concept, we focus on the miniapps crawled from WeChat and Baidu for two reasons. First, being the pioneer of the miniapp paradigm, WeChat has more than two million miniapps, and its user base has reached 1.2 billion [27]. Although Baidu do not have as many miniapps as WeChat, it also has more than

150,000 miniapps [42], which have more miniapps than many other vendors (*e.g.*, Snapchat, which has less than 100 miniapps at the time of this writing). The attacks against those platforms are likely to have more impacts compared to others. Second, WeChat, and Baidu have the cross-miniapp channel to allow the two miniapps to communicate, while many recently launched super apps, such as Kakao from South Korea and Grab from Singapore [40], currently do not support this feature. Finally, we would like to note although we particularly focus on WeChat and Baidu in this paper, there are other super apps subject to CMRF attacks, as described in §8.

3.2 The Vulnerability, Root Cause, and Impact

Based on our threat model and also on how two miniapps cross-communicate, we can clearly see that the `appId` is so fundamental in cross-miniapp communication, and is used by the two APIs: `navigateToMiniProgram` and `navigateBackMiniProgram` to decide the target of the destination miniapp. It works similarly to IP addresses in network communication. Since `appId` is assigned and managed by the super apps, theoretically, attackers cannot pretend to be either the sender or the receiver. However, while `appId` cannot be forged by attackers, developers themselves can make mistakes, which can put the miniapps (particularly those receivers) in grave danger: as alluded earlier, it is the developers' responsibility to ensure that the request consumed by the receiver miniapp is from a trusted sender miniapp. *Any consumption of the requests without* `appId` *checks could lead to CMRF attacks, where an attacker can inject a forged request for malicious purposes.* For example, as shown in §2.2, we can notice that the shopping miniapp (Figure 2) is vulnerable since there is no `appId` check, whereas the payment miniapp (Figure 3) is secure since there is the check.

However, similar to buffer overflows in which not all overflow can lead to control flow hijack attacks, not all lack of `appId` checks can have security impact. For example, if the receiver miniapps do not access any protected resources (as discussed in §2), and only have unprivileged capabilities (*e.g.*, inspecting the width and the height of the window and knowing the version of OS), the attackers do not need to exploit the cross-miniapp-channel at all, as those unprivileged capabilities are publicly accessible to all the miniapps. As such, for CMRF vulnerability to be impactful, it also requires that the victim receiver miniapps to have accesses to the protected resources (*e.g.*, Bluetooth, payment, or location).

3.3 The Workflow of CMRF Attacks

Whenever a receiver miniapp does not check the `appId` of the sender miniapp, this miniapp could be vulnerable to CMRF attacks. To launch such an attack, the attacker can first statically analyze the format of the request by reverse engineering the victim miniapps, then forge a corresponding request with a malicious payload with the same format, and finally inject the forged request to the victim to trick it into believing that the request is from a "trusted" sender and consequently perform any privileged operations. Similarly, once the request has been successfully injected, the receiver may also send back sensitive information to the original sender, and thus the sender can also steal the sensitive information from the victim receiver. Therefore, as illustrated in Figure 4, there could be at least

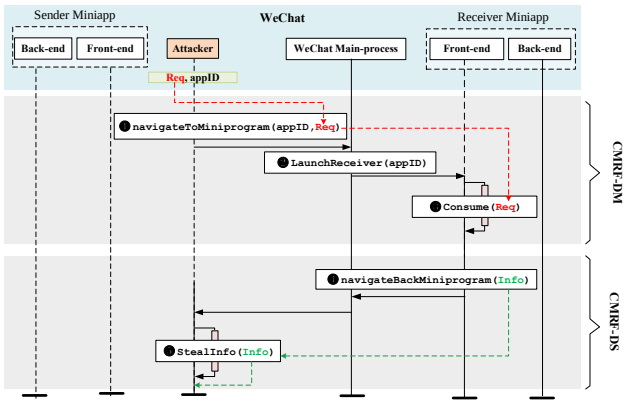

Figure 4: How to Launch an CMRF Attack.

two types of attack consequences depending on the direction of the communication: manipulating the data of the victim by injecting the fake requests (forward direction), or stealing sensitive information (*e.g.*, user credentials) from the victim (backward direction). Next, we explain each type of these attacks in greater detail:

(I) CMRF for Data Manipulation (CMRF-DM). The shopping miniapp shown in Figure 2 is vulnerable since there is no `appId` check. Although in this example, the shopping miniapp is the sender, it will work as the receiver and need to receive the payment status from the intended payment miniapp after the payment has been made. Therefore, an attacker can create a fake payment miniapp and navigate back to the victim shopping miniapp with a `paymentState` "`Success`" message.

To be more specific, we assume that an attacker runs the shopping miniapp on his or her own smartphone. The attacker first selects a product and presses the `purchase` button. Then, the shopping miniapp will invoke the payment miniapp and wait for the payment to be completed. Everything works fine up to this point. However, instead of finishing the normal payment procedure, the attacker terminates the payment miniapp and runs a fake payment miniapp on his or her own development environment, then uses the fake miniapp to create a forged request (which contains the `paymentState` "`Success`"), and sends it to the victim receiver through `navigateToMiniprogram` (Step ❶). This can be easily achieved by waiting for more than 5 minutes such that the shopping miniapp will be killed. The attacker can then use `navigateToMiniprogram` to invoke the shopping miniapp with the forged request.

On the other hand, the shopping miniapp will always use the callback function `onLaunch` to receive the requests from other miniapps regardless of whether or not it is invoked through `navigateBackMiniprogram` or `navigateToMiniprogram`. As such, the host app will launch the receiver miniapp (*e.g.*, the shopping miniapp) based on the `appId` specified by the attacker (Step ❷). Since the shopping miniapp does not check whether the `appId` is from the the intended miniapp (such a check should have been placed before line 18 in Figure 2), the request sent by the malicious miniapp will be processed as usual (Step ❸), and therefore the attacker can achieve "shopping for free" attacks.

(II) CMRF for Data Stealing (CMRF-DS). This type of attack is built upon CMRF for the injection of fake requests, where the

attacker goes a few steps further by receiving requests sent from the receiver. This time, the receiver, *e.g.*, the payment miniapp, is the victim which leaks sensitive data, *e.g.*, the coupon code (line 14 in Figure 3), to the attacker, which should not be leaked because it is supposed to be distributed to the user who initiated the payment. While it has checked the `appId` of the sender miniapp of the request (starting from line 13), we assume that the checking process is missing from now on to demonstrate our attack principle.

Specifically, we assume that the attacker has already used steps ❶-❸ via `navigateToMiniapp`, and now the attacker's miniapp is switched offline. Next, since we assume that the receiver does not check the `appId`, it will then believe that the sender is legitimate and will respond to the sender in future communication. In Step ❹, the receiver finishes the request and subsequently sends the response (which contains the coupon) to the sender by `navigateBackMiniProgram`, which will notify the host app, which will further launch the original attacker's miniapp. Finally, the attacker can use the callback function `onShow` or `onLaunch` to steal sensitive information from the victim receiver (Step ❽). For instance, in our example, the attacker obtained the `coupon`. Additionally, note that if the malicious miniapp is installed onto the user's mobile, it will have severe security consequences (*e.g.*, stealing the location data to track the users).

4 VULNERABILITY DETECTION

Since the lack of `appID` check is the root cause of the CMRF vulnerability (similar to the lack of bounds check is the root cause of buffer overflow vulnerability), we have to identify the missing checks in the miniapp code in order to identify the vulnerability. In this section, we present CmrfScanner, a JavaScript static analysis tool that utilizes the abstract syntax tree (AST) reconstructed from miniapp code to scan for vulnerable miniapps. Again, a miniapp is subject to our CMRF attacks when (*i*) it uses the cross-miniapp channel and (*ii*) it does not check whether the request is from a trusted sender (*i.e.*, lack of `appID` check). Therefore, the key idea for CmrfScanner is to first detect whether there is the use of the cross-miniapp request (§4.1), and if so, CmrfScanner then further detects whether there are missing `appID` checks (§4.2).

4.1 Detecting Uses of Cross-miniapp Requests

A miniapp vulnerable to CMRF attack must use a cross-miniapp request. To detect whether a miniapp contains such a request, we can inspect the framework level APIs and their parameters since miniapps must use standard APIs to receive the requests (and these APIs cannot be hidden). More specifically, for WeChat, we know it will wrap the information from a sender miniapp in an object called `referrerInfo`, containing `appId` and extra data of the sender. All of the cross-miniapp messages will involve the members of `referrerInfo.extraData.*`, where the `*` represents a specific member of `extraData`, *e.g.*, `paymentStatus`. As such, we can first inspect the object to which a variable or member expression is referring, and then match the aforementioned string to confirm the use of the cross-miniapp request. Additionally, from the receiver's perspective, it needs to use the startup parameter from different life-cycle functions (*e.g.*, `onLaunch` or `onShow`) to fetch the cross-miniapp

```
1  //code fragment of a vulnerable miniapp
2  "wxd7c977843ebe7a64"==(
3    e.referrerInfo.appId?e.referrerInfo.appId:"")
4    && checkPayStatus(param).then(function(a){...})
5
6  }
```

Figure 5: Code snippet of complex ID check

requests, and get the information from the sender miniapp (*e.g.*, the `appId` and `extraData`).

Therefore, a straightforward way to detect whether there is the use of the cross-miniapp request is to match the string of `referrerInfo.extraData.*`. However, it is well-known that variables can be assigned to other variables and variables can have aliases. In general, the alias may appear in either the assignments or function invocations. Specifically, (*i*) the instance of `referrerInfo` may be re-assigned to a local variable (*e.g.*, `a`), and if so, the developer-defined variables are visited through expressions such as "`a.extraData.*`". (*ii*) The instance of `referrerInfo` can be passed to a function as a parameter (*e.g.*, `func(a)`, where `a` is an alias of `referrerInfo`). As such, we have to design an alias-aware static-analysis algorithm to detect the use of cross-miniapp requests. In the following, we describe our algorithm in greater detail.

(I) Identifying the Uses of the Requests Across Assignments. To identify the uses of the request across direct assignments is simple: if the value of `referrerInfo` or `referrerInfo.extraData` is assigned to a specific variable, CmrfScanner then records that variable and keeps tracking whether the variable is assigned to other variables or not. If so, it also records other variables. Also, when detecting that the right-hand side (RHS) of a recorded variable (which is originally assigned from `referrerInfo.extraData`, or `referrerInfo`) is fetched by the miniapp (*e.g.*, `a.extraData.paymentStatus`, where the variable `a` is the recorded variable), it concludes that the request is used. For example, assume "`var a = referrerInfo.extraData`", CmrfScanner will record "`a`" first. Later, when observing "`var b = a`", it will record "`b`" as well. Finally, the miniapp fetches the developer-defined variable (*e.g.*, `paymentStatus`) from `b` (*e.g.*, `b.extraData.paymentStatus`), it then identifies the use of the request across different assignments.

(II) Identifying the Uses of the Requests Across Function Invocations. To identify the uses of the request across function invocations is more sophisticated, as it involves the resolution of data flows across caller, callee and the passed parameters (*e.g.*, `a.extraData.paymentStatus` can be passed as a parameter). Theoretically, we can follow the same practice by recording and tracking the variables and the parameters of functions that are originally re-assigned from `referrerInfo.extraData` (or `referrerInfo`), and iterating the each statement of the functions to confirm whether or not the parameters are assigned to other variables, or fetched by the miniapps. However, this may introduce an inefficiency issue when the callee is a utility function that is being called multiple times (which means that we need to traverse the same function node multiple times). Moreover, such a solution may cause the analyzer to enter a dead loop if the function call graph contains cycles. This is because there could be functions calling each other recursively (*e.g.*, where a function keeps invoking another function until a specific condition is satisfied). In dynamic analysis, the analyzer tool can terminate the program when it finds the condition satisfied. However,

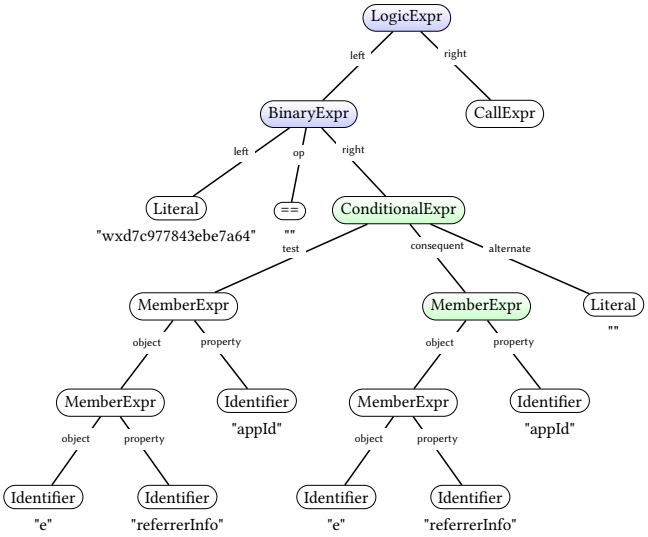

Figure 6: The AST Tree of the code snippet of Figure 5, where the green nodes involve `appId` and blue nodes indicate in which the check is performed.

we are performing static analysis, and it is likely that the analysis may never end (*e.g.*, in dead loops) when the functions call each other recursively, since the analyzer tool does not know whether the condition is satisfied. As such, we need to find a solution that makes the analysis more efficient and solves dead loops.

To improve the efficiency, we propose not to reanalyze an analyzed function multiple times, but to store a call graph to record the caller with an index of the parameter of interest (the parameter is originally assigned from `refererInfo.extraData`). In particular, for each function involving these parameters of interest, we scan each of the parameters individually and record the analyzed results for future reference (*i.e.* whether the miniapp fetches any requests from these parameters of interest). Such a practice can avoid the analyzer entering dead loops as well, since for each of the functions being analyzed, we will not process any functions that do not have the parameter of interest involved, although the functions call each other recursively.

4.2 Detecting the Missing Checks

Having identified the use of a cross-miniapp request, we next need to see whether there are missing checks of `appID`s. While the simple string matching may be able to handle simple comparisons (*e.g.*, `appId == "wxd7c977843ebe7a64"`), it cannot process the comparisons that are as complex as shown in Figure 5, in which nested logical expressions and conditional expressions have separated the comparison operators and the access to the sender's `appId`. Fundamentally, if there is an `appId` check, a receiver miniapp must fetch the sender miniapp's `appId` from the object `referrerInfo.appId`, which is set by the WeChat framework, and then compare with a string, a variable, or an array of `appId`s, and so on.

Therefore, the miniapp `appID` checking will always involve logical or binary expressions such as `==` and `in`, where the `appId` is involved in *either side of the expression* (*e.g.*, `appId` and the variable being checked can even be located on the same side of the expressions such as "`if (["wxd7c977843ebe7a64", "wxd3cd37823e137a53"].indexOf(referrerInfo.appId) !== -1)`"). Particularly, we first identify binary expressions under these conditions (*e.g.*, `If`, `For`, `While`, and logical expressions) because all these checks on a given set of `appId`s are located in binary expressions. Next, we traverse both the left and right hand sides of the binary expression, and if either side contains a variable that is resolved to `refferer Info.appId` (*i.e.*, the variable originally assigned from `refferer Info.appId`), we identify the sender `appId` of this conditional branch as checked.

For example, as shown in the AST tree in Figure 6, which is the tree associated with the example in Figure 5 (lines 2 – 4), the Member Expression that involves `appId` Identifier (marked green) is in the right branch of Binary Expression marked blue, whose left branch is the Literal of the compared `appId` `"wxd7c977843ebe7a64"`. Furthermore, since we have already collected multiple paths that involve the uses of the requests, we perform the detection on each of those paths to confirm whether there are missing checks. For a specific miniapp, as long as we have confirmed one path that involves the uses of the request without checks, we conclude that this miniapp is vulnerable to CMRF attacks.

5 IMPLEMENTATION

CMRF Attack Implementation. We have implemented the proof-of-concept (PoC) attack code by following the developer documentation of the super apps (*e.g.*, Baidu, WeChat) as well as reverse engineering of the targeted miniapps. At a high level, since the CMRF attacks are often receiver miniapp-specific, we first need to reverse engineer the victim miniapps of interest (*e.g.*, those receivers without the security checks, and have the capabilities to access sensitive resources) to understand its logic and decide which specific attack (*e.g.*, CMRF-DS or CMRF-DM) can be launched accordingly. For example, the shopping miniapp may be subject to shopping for free attack, where the attackers can inject fake payment status (the format of the payment status messages can be identified through reverse engineering). With the knowledge obtained from the reverse engineering of the victim miniapps, we then created the corresponding fake request and injected the message through `navigateToMiniProgram` (CMRF-DM), or obtained the transmitted messages from the victim (CMRF-DS).

CmrfScanner **Implementation.** We have implemented our CmrfScanner, whose source code has been made available at https://github.com/OSUSecLab/CMRFScanner. We did not build it from stractch and instead we built it based on the open source tool `DoubleX` [22]. Specifically, since our analysis involves domain-specific knowledge and complex AST parsing rules, we used `DoubleX` due to its easy AST traversal APIs and value analysis components. We also modified `DoubleX` to enable the parsing of JS files in miniapp and used domain knowledge for function entry identification, `appID` check identification, and use of the cross miniapp request data.

6 EVALUATION

In this section, we present the evaluation result. To crawl the miniapp for the testing by CmrfScanner, we used our open source MiniCrawler [48] to download the miniapps from WeChat appstore. We obtained 2,571,490 WeChat miniapps in total, which

Category	No Use		Checked		Vulnerable	
	# app	%total	# app	%	# app	%
WeChat						
Business	131,078	5.1	81	8.07	923	91.93
E-learning	10,271	0.4	4	5.19	73	94.81
Education	240,077	9.34	184	3.72	4,756	96.28
Entertainment	29,442	1.14	140	33.02	284	66.98
Finance	3,509	0.14	6	6.67	84	93.33
Food	114,675	4.46	332	8.07	3,780	91.93
Games	88,056	3.42	10	2.09	469	97.91
Government	31,432	1.22	33	9.02	333	90.98
Health	27,716	1.08	37	5.44	643	94.56
Job	21,773	0.85	16	7.02	212	92.98
Lifestyle	394,493	15.34	269	4.23	6,092	95.77
Photo	9,039	0.35	3	4.41	65	95.59
Shopping	989,498	38.48	743	2.56	28,304	97.44
Social	20,671	0.8	6	2.99	195	97.01
Sports	15,980	0.62	69	22.48	238	77.52
Tool	261,467	10.17	122	3.72	3,161	96.28
Traffic	35,412	1.38	53	9.28	518	90.72
Travelling	10,524	0.41	5	3.62	133	96.38
Uncategorized	83,983	3.27	0	0.0	18	100.0
Total	2,519,096	97.96	2,113	4.03	50,281	95.97
Baidu						
Automobile	356	0.24	0	0.0	2	100.0
Business	5,201	3.5	0	0.0	113	100.0
Charity	2	0.0	0	0	0	0
E-commerce	96	0.06	0	0	0	0
Education	1,378	0.93	0	0.0	3	100.0
Efficiency	10,852	7.31	0	0.0	1	100.0
Entertainment	195	0.13	1	11.11	8	88.89
Finance	45	0.03	0	0.0	2	100.0
Food	123	0.08	0	0	0	0
Government	282	0.19	0	0.0	5	100.0
Health	2	0.0	0	0	0	0
Information	1,736	1.17	0	0.0	6	100.0
IT tech	113	0.08	0	0	0	0
Lifestyle	1,818	1.22	0	0	0	0
Medical	97	0.07	0	0	0	0
News	4	0.0	0	0	0	0
Post service	163	0.11	0	0	0	0
Real estate	1,510	1.02	0	0	0	0
Shopping	116,093	78.17	0	0.0	327	100.0
Social	205	0.14	0	0	0	0
Sports	145	0.1	0	0	0	0
Tool	46	0.03	0	0	0	0
Traffic	226	0.15	0	0.0	1	100.0
Travelling	1,473	0.99	0	0	0	0
Uncategorized	5,857	3.94	0	0.0	25	100.0
Total	148,018	99.67	1	0.2	493	99.8

Table 2: The statistics of miniapps w.r.t app categories

consume 6.29 TB disk storage. We also extended MiniCralwer to allow it to download miniapps from Baidu market, with which we also collected 148,512 Baidu miniapps (consuming 81 GB disk storage). CmrfScanner was tested with these data sets using three desktop PCs running Ubuntu 18.04 with i7-7700 CPU and 16 GB memory each. In this section, we first present the effectiveness of CmrfScanner with these miniapps (§6.1), and then report its efficiency (§6.2).

6.1 Effectiveness

To understand how popular a miniapp is subject to CMRF attacks, we tested CmrfScanner with our crawled data set. The detailed detection results are reported in Table 2. In total, there are 2,519,096 (97.96%) WeChat miniapps and 148,018 (99.67%) Baidu miniapps that never involve cross-communication and therefore are not vulnerable to CMRF attacks at all. However, surprisingly, for the WeChat miniapps that use cross-communication, 50,281 (95.97%) of them fail to perform the checking of the source miniapp's `appId`. Similarly, among all 494 Baidu miniapps that use the cross-communication, there are 493 (99.80%) miniapps that do not perform the check. This indicates that regardless of the providers, miniapps that fail to validate the `appID` are ubiquitous, and the vast majority of developers are never aware of our CMRF attacks. Also, interestingly, we find that some developers who have added the checks still leave some execution paths unprotected, which still make their miniapps vulnerable albeit some checks had already been performed: 620 and 4 miniapps from WeChat and Baidu, respectively, fall into this category. In the following, we first quantify the correctness of CmrfScanner by inspecting its false positives and false negatives (§6.1.1), and then dive into deeper the detected vulnerable miniapps (§6.1.2).

6.1.1 ***False Positive and False Negative Analysis.*** Although DoubleX proved to be a powerful tool in inferring the value of variables, and domain-specific efforts have been made in CmrfScanner to improve the accuracy, there are certain conditions under which false positives or false negatives may occur. We note that failure to identify appID-check (*e.g.*, caused by aliasing) will make our CmrfScanner mistakenly identify miniapps that checked appID as vulnerable, causing false positives. Similarly, failure to detect the use of cross-miniapp message will make CmrfScanner fail to identify vulnerabilities, causing false negatives, since the use of the check means not vulnerable.

Given that there is no ground truth, to verify whether there are any false positives and false negatives of our result, we sampled 100 miniapps from the miniapps identified as vulnerable and checked to verify the false positives, and sampled 100 miniapps from the miniapps identified as not involving extra data to verify false negatives. We plan to release a set of the vulnerable miniapps including these manually labeled 200 miniapps to support open science, which can serve as the ground truth for any follow-up research.

False Positives (FPs). We found no FPs among these detected 100 vulnerable miniapps, and each of them is indeed vulnerable according to our manual inspection of the miniapp code. Interestingly, during our inspection, we found that many miniapps were developed from certain templates, where the vulnerability in templates resulted in their vulnerabilities. For instance, we identified that there are multiple mini-apps sharing the same folder structure and the same sequences of code (which can be used to fingerprint the type of templates). In the end, we found 68 miniapps generated by three different templates that are vulnerable among 100 sampled miniapps, which consists of 68% sampled miniapps. This shows that code generated from the template is a serious problem in miniapp security.

False Negatives (FNs). Among the 100 miniapps that are not identified vulnerable, we found 2 false negatives, making the FN rate to 2%. Both of these false negatives occur because DoubleX failed to infer the object referrered to by a member expression. As the value analysis pointing to `extraData` returns `None`, CmrfScanner will not find a variable matching `extraData`, this marking it as not using cross-miniapp channel. Thus, a false negative occurs because the miniapp actually uses the cross-miniapp channel and there is the lack of `appId` check.

6.1.2 ***Insights of the Vulnerable Miniapps.*** Next, we seek to draw a few insights from these large volumes of identified vulnerable miniapps, such as whether there are any correlations between the vulnerabilities with (*i*) the app category and (*ii*) the app ratings, and (*iii*) the potential accessed resources and the categorizations of

the attacks, and (*iv*) how they cross communicate (*e.g.*, `navigateTo` or `navigateBack`).

(I) Distribution between Vulnerable Miniapps and Their Categories. Since the miniapp categories reflect the types of services a miniapp provides, they could partially reflect the potential attacks to which a miniapp may be vulnerable. This result is reported in Table 2. We can see that among the 19 WeChat categories and 20 Baidu categories, the top-hit categories with vulnerable miniapps are shopping, lifestyle, and tool (regardless of their host apps).

Intuitively, the shopping miniapps often provide e-commerce services, where products are ordered, purchased, and delivered. It is not a surprise to see that the shopping miniapps often have to communicate with other miniapps. The lifestyle miniapps, however, mainly provide convenient services for daily lives ranging from hair salon to SPA and massage. Similar to shopping miniapps, these lifestyle miniapps also are highly possible to involve online purchases and membership management. As such, the impact of the CMRF attack against these miniapps could be concerning, as the attackers may be able to forge orders or membership information to perform an impersonation attack or bypass the payment to get products or services for free.

(II) Distribution between Vulnerable Miniapps and Their Ratings. On top of the miniapp categories, we also classify miniapps with respect to their popularity. For WeChat miniapps, unlike Baidu miniapps which have number of clicks in the metadata, they do not contain such information, and instead we can only use other data to approximate this. In particular, we can use their ratings directly from the metadata [48] to approximate the popularity of the apps, since WeChat miniapp ratings are only available to the public when a sufficient number of users have rated them. The ratings could be an important dimension in understanding the miniapp ecosystem, since they reflect not only the popularity of the miniapps, as the rating of less popular miniapps will not be shown to the public (there is no number of clicks for each miniapp in WeChat available metadata), but also the quality of the services a miniapp provides. For Baidu miniapps, we collected the number of clicks (which can be extracted from the metadata) to reflect its popularity, as a click likely indicates an install. Also, note that there is no rating available in Baidu miniapps.

As shown in Table 3, we found that the ratings for many of the WeChat miniapps are not shown, which means that they are less popular to have enough users to rate them. However, among the rated miniapps, we found that miniapps with higher ratings are less likely to have more vulnerability. For example, among the rated vulnerable miniapps, the ratio of the popular vulnerable miniapps (*e.g.*, 95.73%) is slightly less than that of non-popular vulnerable apps (100%). Interestingly, we observed slightly different results from the Baidu miniapps in that the majority of vulnerable miniapps have a total click count from 1k to 100k (79.72%), which is in the middle of the total popularity span. However, given that these miniapps have a large amount of users, the impacts of our CMRF attacks can still be catastrophic.

(III) Distribution between Vulnerable Miniapps and Resource Access. Although theoretically attackers can inject any forged request into receivers that do not perform the `appID` checks, the inject request may not have any security impacts, particularly when the receiver miniapps themselves do not have privileged capabilities to access sensitive resources. Therefore, it is also necessary to quantify the security impact of the lack of `appID` checks. While there are many ways to do so, a lightweight approach is to use the accessed resources as a proxy (by scanning the mission-critical APIs and permissions that used by those vulnerable miniapps) to estimate the impacts of the attacks. Note that the permission list and the resource of attacker's interest can be found in Table 1, and the permissions and APIs (*e.g.*, `getLocation`) can be recognized through scanning the configuration file (similar to Android) or the code of the miniapps (the uses of privileged APIs).

Popularity	No Use		Checked		Vulnerable	
	# app	%total	# app	%	# app	%
WeChat						
5.0	4,831	0.19	9	4.27	202	95.73
[4.5,5.0)	48,486	1.89	82	5.04	1,545	94.96
[4.0,4.5)	23,794	0.93	18	2.93	597	97.07
[3.5,4.0)	12,222	0.48	3	2.03	145	97.97
[3.0,3.5)	8,342	0.32	2	2.25	87	97.75
[2.5,3.0)	4,375	0.17	2	6.67	28	93.33
[2.0,2.5)	2,014	0.08	0	0.0	3	100.0
[1.5,2.0)	693	0.03	0	0.0	2	100.0
[1.0,1.5)	166	0.01	0	0	0	0
Not Scored	2,414,173	93.88	1,997	4.02	47,672	95.98
Total	2,519,096	97.96	2,113	4.03	50,281	95.97
Baidu						
[10M,100M)	7	0.0	0	0.0	1	100.0
[1M,10M)	93	0.06	0	0.0	1	100.0
[100K,1M)	2,456	1.65	0	0.0	29	100.0
[10K,100K)	45,495	30.63	1	0.5	200	99.5
[1K,10K)	62,151	41.85	0	0.0	193	100.0
[100,1K)	25,868	17.42	0	0.0	46	100.0
[10,100)	8,196	5.52	0	0.0	10	100.0
[1,10)	3,752	2.53	0	0.0	13	100.0
Total	148,018	99.67	1	0.2	493	99.8

Table 3: The statistics of miniapps w.r.t. their popularity. The popularity is measured by the ratings for WeChat miniapps, and number of clicks for Baidu's.

To this end, we scanned the vulnerable miniapps again, to identify the mission critical APIs and permissions to quantify the potential security impact of CMRF attacks. As reported in Table 4, based on the categories of attacks those miniapps are subject to, we can notice in total that there are 28,326 WeChat miniapps and 22 Baidu miniapps that are subject to CMRF-DM, and 1,541 WeChat miniapps and 20 Baidu miniapps that are subject to CMRF-DS, respectrively. In addition, we also found that over 28,843 (55.05%) WeChat miniapps involved at least one type of protected resources, and many of them involve some type of resources that are specifically sensitive, such as user location and payment. Meanwhile, we noticed that the distributions of the miniapps involving mission critical APIs of Baidu is different from that of WeChat (similarly, the distributions of the use of cross-miniapp channel of Baidu, 0.33%, and WeChat, 2.04%, are also different). There are 19 (3.85%) and 15 (3.04%) miniapps involving network and payment, respectively. As such, we believe that this problem is currently widely spread across mission critical miniapps with high privileged capabilities.

(IV) The Cross-Communication Statistics in the Vulnerable Miniapps. While we do not need to identify the sender miniapp when determining the vulnerabilities at the receiver side (since we only need to analyze the code of the receiver miniapp to confirm the vulnerability), we still would like to explore the relationship between senders and receivers, *e.g.*, which miniapp redirects to or back to a vulnerable miniapp. As such, we first recover the redirection

Attack	Resources	WeChat		Baidu	
		#	%	#	%
CMRF-DM	bluetooth	98	0.19	0	0.00
	card	13,139	25.08	0	0.00
	location	9,029	17.23	2	0.40
	media	2,898	5.53	2	0.40
	socket	3,614	6.90	5	1.01
	tcp	15,890	30.33	19	3.85
	sports	0	0.00	0	0.00
CMRF-DS	address	195	0.37	1	0.20
	bluetooth	193	0.37	0	0.00
	camera	31	0.06	0	0.00
	datacache	1506	2.87	20	4.05
	file	274	0.52	3	0.61
	https	1,539	2.94	20	4.05
	invoice	13	0.02	1	0.20
	location	593	1.13	10	2.02
	media	1,154	2.20	6	1.21
	payment	1381	2.64	15	3.04
	socket	325	0.62	3	0.61
	userinfo	1,364	2.60	10	2.02
	sports	8	0.02	0	0.00
Cross-communication/Total		52,394	2.04	494	0.33
Privileged/Vulnerable		28,843	55.05	35	7.09

Table 4: Distribution between vulnerable miniapps and accessed resources

	Navigate to			Navigate back		
	Categories					
	# vuln	# edges	%	# vuln	# edges	%
Business	23	41	5.56	97	97	1.31
E-learning	4	4	0.97	1	1	0.13
Education	48	232	11.59	227	227	7.43
Entertainment	9	42	2.17	11	11	1.35
Finance	3	10	0.72	6	6	0.32
Food	4	5	0.97	480	480	0.16
Games	12	54	2.9	8	8	1.73
Government	20	56	4.83	45	48	1.79
Health	25	42	6.04	52	52	1.35
Job	6	52	1.45	15	15	1.67
Lifestyle	58	1,274	14.01	340	340	40.81
Photo	2	19	0.48	8	8	0.61
Shopping	65	195	15.7	946	947	6.25
Social	6	13	1.45	11	11	0.42
Sports	1	1	0.24	10	10	0.03
Tool	98	326	23.67	174	177	10.44
Traffic	14	25	3.38	68	69	0.8
Travelling	3	7	0.72	7	7	0.22
Uncategorized	0	0	0.0	0	0	0.0
Total	401	2,398	96.86	2,506	2,514	80.27
	Categories					
5.0	4	5	0.97	4	5	0.16
[4.5,5.0)	92	325	22.22	92	325	10.41
[4.0,4.5)	51	203	12.32	51	203	6.5
[3.5,4.0)	12	51	2.9	12	51	1.63
[3.0,3.5)	6	30	1.45	6	30	0.96
[2.5,3.0)	2	3	0.48	2	3	0.1
[2.0,2.5)	0	0	0.0	0	0	0.0
[1.5,2.0)	0	0	0.0	0	0	0.0
[1.0,1.5)	0	0	0.0	0	0	0.0
Not Scored	234	1,781	56.52	234	1,781	57.05
Total	401	2,398	96.86	2,506	2,514	80.27

Table 5: Statistics of vulnerable WeChat miniapps with resolved `appId`s and their communication directions and data.

target specified with the `appId` in the `navigateToMiniProgram`, and then generate a redirection graph where the nodes are miniapps (indexed by `appId`s) and edges are pointed from the sender to the receiver miniapp. To this end, we parse the AST tree and collect the `appId` property to obtain the `appId` values. Although there are many cases in which the `appId` is dynamically set in WeChat, we were still able to obtain 2,907 `appId`s (and they are just constant strings), and eventually connect them with 4,912 edges. We were not able to obtain many concrete values of the `appID`s for Baidu miniapps since most of them are dynamically distributed from servers.

In particular, as shown in Table 5, we categorize the involved vulnerable WeChat miniapps based on how a sender initiates the cross-miniapp communication and how a receiver processes it. Please note that Baidu miniapps are not included, since we only identify a few miniapps that have hard coded the receivers' `appId`s in the `navigateToMiniProgram`. While such practice will not affect the vulnerability detection, our tool cannot resolve the relationship between the sender and the receiver (only dynamic analysis can resolve this).

As reported in Table 5, we found that although the vulnerable miniapps involving '`navigationTo`' is much less than that of '`navigationBack`' (401 vs. 2,506), the involved redirection edges is close to that of '`navigationBack`' (2,398 versus 2,514). This indicates that there are extremely popular vulnerable miniapps to which are being redirected by hundreds of miniapps, which consequently generates hundreds of edges involving a single vulnerable miniapp.

6.2 Efficiency

Finally, we quantify the execution time of CmrfScanner. Since its analysis involved data flow analysis, which is time consuming, we first filtered out those miniapps that do not involve any cross-app communication (*i.e.*, `navigateToMiniProgram` and `navigateBackMiniProgram`) by inspecting the used APIs, resulting in 52,394 WeChat miniapps and 494 Baidu miniapps from the 2,571,490 Tencent miniapps and 148,512 Baidu miniapps, respectively. Then, CmrfScanner performed the analysis with these miniapps based on the usage of `extraData`. The evaluation was carried out on a desktop with 24 threads in parallel, and it took 9 days to finish analyzing all the WeChat miniapps and less than 12 hours to finish analyzing all the Baidu miniapps, where the average cost of time to analyze a single miniapp is 48.61 and 43.45 seconds for Tencent and Baidu miniapps, respectively.

7 SECURITY CASE STUDIES

In this section, we perform a few case studies to understand the impacts of our CRMF attacks. At a high level, the attacks will be application specific depending on how the exchanged data is used. Although our analysis can narrow down the miniapps that access the sensitive resources down to 28,878 (28,843 wechat miniapps, and 35 Baidu miniapps), we still have to inspect the semantics of these data of the vulnerable miniapps to design the corresponding concrete attacks. While this is generally a challenging task, we can leverage the variable names in the miniapp code to reflect the semantic meaning, since miniapp is developed using JavaScript and the variables can be easily parsed. To this end, we developed a simple plugin to walk through the AST again, to collect the variable names related to the cross-miniapp messages (particularly extraData) received by vulnerable miniapps. Note that although the miniapps are usually heavily obfuscated, the variable names of the request will not be obfuscated as the receiver will need to fetch the messages based on the variable names.

In total, we identified 2,412 meaningful variable names, and these variable names are examined by three experienced security researchers to see whether there are any security concerns. In the end, as shown in Table 6, we categorized the identified variables with respect to their semantic meanings and present the top-5 hit

Category	Variable Name	Vulnerable w/o Check	Vulnerable w/ Incomplete Check	Total
Payment Info	for_pay_back	355	0	355
	payStatus	313	2	315
	pay	178	9	187
	isPay	118	0	118
	isLecturePay	115	0	115
Order Info	orderId	132	11	143
	orderInfo	80	0	80
	order_id	42	0	42
	jtOrderId	36	3	39
	hpj_jsapi_order_id	21	0	21
Phone Number	mobile	6,627	7	6,634
	phone	53	0	53
	userPhone	8	0	8
	phoneNumber	6	1	7
	partnerMobile	2	0	2
Promotion Info	cardId	25	0	25
	user_coupon_id	2	0	2
	couponCode	1	0	1
	coupon_id	1	0	1
	coupon_no	1	0	1
Device Info	deviceID	2	0	2
	uuid	2	0	2
	deviceId	1	0	1
	devicenum	1	0	1
	UUID	1	0	1

Table 6: Top-five cross-miniapp variable names that may involve security concerns with respect to affected types of resources.

variable names in the corresponding category that can involve severe consequences if the cross-communicated message is forged. As shown in the table, these involved variables can be classified into the following five categories according to the names:

- **Payment Info**: Payment Info includes payment scene, type, and the result of payment, which could be fetched or injected by attackers to shop for free.
- **Device Info**: Device info includes the information obtained from the (IoT) devices, such as the `deviceID`, `devicenum`, or `room_id`. If the device info is forged, the attacker can launch similar impersonation attacks.
- **Promotion Info**: Promotion Info usually includes users' membership cards, member coupons and promotions that can be exploited by attackers to abuse the promotion provided by shops.
- **Order Info**: Order Info includes order IDs and information that could be modified by attackers including prices and product amounts.
- **Phone Number**: Phone Number is crucial to users in miniapp paradigm in that most miniapps utilize this as identifier and even account for users to log in. If utilized by attackers, attacker may hijack arbitrary account for any illicit purposes.

In this section, due to page limit, we just demonstrate concretely how to launch shopping for free (§7.1), device manipulation (§7.2), and promotion abuse attacks (§7.3).

7.1 Shopping for Free Attacks

As discussed, many shopping miniapps may not develop their own payment mechanism but instead use the third-party payment miniapps. However, if a shopping miniapp does not check the source of the information about the payment status sent in `extraData`, the attacker may send a success status to a certain order and shop it for free. As shown in Figure 7, the miniapp Xixiu Group Purchase Backend retrieves the payment status information at line 4,

```
1  //app.js
2  get_pay_info: function(e) {
3  var t = this;
4    if (... e.referrerInfo.extraData.for_pay_back && this.waitForPayBack) {
5      this.waitForPayBack = !1, wx._hideLoading();
6      var r = e.referrerInfo.extraData;
7      "2" == r.pay_status && (this.broadcastUpdate(), "function" == typeof
       ↪  this.payBackSuccess && this.payBackSuccess()),
8      "3" != r.pay_status && "4" != r.pay_status || wx._showAlert({
9        content: "payment failed",
10       success: function() {
11         "function" == typeof t.payBackFail && t.payBackFail();
12       }
13     });
14   }
15 },
```

Figure 7: Vulnerable Code (Xixiu Group Purchase Backend) that is Subject to Shopping for Free Attacks.

which first checks whether the `for_pay_back` and `waitForPayBack` flag is true to ensure that the current miniapp has a pending payment and receives a cross-miniapp request for payment result information. Then, the miniapp assigns the `extraData` to `r` (line 7), and checks if `pay_status` is 2 (which indicates the payment is successfully processed). However, as the miniapp does not check the `appID` of the source miniapp, an attacker can launch this vulnerable miniapp, make an order, hang the redirected payment and inject a message with `pay_status` set to 2 with the attacker's own malicious miniapp. As the miniapp will treat the order as a success, the attacker can buy products for free. We have verified this attack in our controlled environment, and it succeeded.

7.2 Devices Manipulation Attacks

We noticed that many IoT miniapps are using the exchanged `deviceID` to identify a device. However, the miniapps should validate the device ID sent in cross-miniapp channel is from trusted users to avoid privacy concern and financial losses. However, as shown in Figure 8, a miniapp named Suyuan Webcam from Suyuan Surveillance, a company that produces surveillance cameras, first retrieves `deviceID` from `extraData` (and sets `globalData` to the corresponding `deviceID` to preserve the device ID) in a conditional expression. If the device ID is NULL, it goes into the alternative branch at the end of line 3, where the miniapp shows a dialog saying "live device not found". If the device ID is set, in `loadLive` at line 14, it sends a request to the backend to fetch the video streaming URL of the device ID and opens a webview to watch the live video of the camera. However, as the source `appID` is not verified, an attacker can easily use his/her own miniapp and inject or even enumerate device ID to see the streaming of the camera.

7.3 Promotion Abuse Attacks

In miniapp paradigm, the mobile phone number is very important, since it is usually used to uniquely index a user's account in miniapps. For instance, in a Baidu miniapp named Aurora Vision as shown in Figure 9, the miniapp first parses the value of `extraData.coupon` to `e` and then creates an object containing the merchant ID, user ID, and the coupon parameters after parsing the coupon content as URI. However, since the attacker can manipulate the data by sending crafted coupon info via his/her own miniapp, the attacker can collect the benefit of the targeted offer distributed in the form of coupon. Interestingly, we also noticed that many of

```
//  app.js
onLaunch: function(t) {
  console.log(t), a.String.isBlank(t) ||
  ↪ a.String.isBlank(t.referrerInfo.extraData.deviceId) ?
  ↪ this.globalData.data.deviceId = null : this.globalData.data =
  ↪ t.referrerInfo.extraData, wx.showModal({
    title: "Hint",
    content: "Live device not found", ...
    });},
// index/index.js
loadLive: function() {
  wx.request({
    url: "https://***.com/device/getVideoUrlByDeviceId",
    data: {
      deviceId: e.globalData.data.deviceId
    },
    method: "get",
    success: function(o) {...}})}
```

Figure 8: Vulnerable Code (Suyuan Webcam) that is subject Webcam Manipulation Attacks

```
onShow: function(t) {
  if (t.referrerInfo && "{}" !== s()(t.referrerInfo) && "{}" !==
  ↪ s()(t.referrerInfo.extraData) && t.referrerInfo.extraData.coupon) {
    var e = t.referrerInfo.extraData.coupon,
    n = this.getUserInfoFromCacheSync() || "";
    Object(l.k)({
      merchantId: this.globalData.merchantId,
      couponParams: encodeURI(e),
      userId: n.sid || "",
      vs: "V3"
    }).then((function(t) {
      console.log("Coupon res", t)
    })).catch((function(t) {
      console.log("Coupon err", t)
    }))
  }
}
```

Figure 9: Vulnerable Code (Aurora Vision) from Baidu, which is Subject to Promotion Abuse Attacks (involves payment)

those vulnerable miniapps used the exact the same code to receive user phone numbers, and those miniapps are likely to be produced by the same template. We further discovered that some of these apps were generated from online miniapp template generation services (*e.g.*, Youzan, Yudian, and Qingzhou). These miniapps generators work in the "What You See Is What You Get" fashion, which allows miniapp developers to easily customize their miniapps by modifying specific text or figures without any additional programming efforts. Since these generators have the default packed resources and code, a developer could use them directly without any modifications. As such, if one of these miniapps is found to be subject to our attacks, all miniapps generated by this template will be similarly vulnerable.

8 DISCUSSION

Generality and Practicality of CMRF Attacks. While we have only evaluated CMRF on two platforms, *i.e.*, WeChat and Baidu, we believe our findings are general and CMRF attacks are practical in many super apps. First, as shown in Table 7, we notice that similar to WeChat and Baidu, there are a lot of super apps that have provided the cross-app communication channel, and their implementations are quite similar: only Baidu has a different name for their cross-miniapp API. In fact, most of the APIs offered by the super apps have the same or similar names (*e.g.*, `navigateToSmartMiniProgram` in Baidu and `navigateToMiniProgram` in other platforms). Second, our CMRF attacks require the receiver miniapps to access sensitive resources in order to have security impact. We notice that the sensitive resources provided by different platforms are similar and that they all used similar mechanisms to protect those resources (although some resources are not provided on some platforms), albeit the fact that some platforms provide more resources for miniapp, such as WeChat. For example, all the platforms have accesses to the users' locations, and those locations are guarded by the permission mechanism. Third, the most effective way of defending against the CMRF attack is to validate the `appID` and enforce the security check before the receiver consumes the request. However, not all developers will perform the checks, given their varied backgrounds and skill sets, leading to their miniapps subject to CMRF attacks. Finally, as discussed in §3.1, our attacks do not require the malware to be released on the market, and the attackers do not need to bypass the vetting enforced by the platforms, which has also increased the practicality of the attacks.

Limitations and Future Work. Our study is not perfect. First, during our evaluation, we attempt to recover the miniapp cross-communication map. However, not of all the appIds can be recovered and therefore we cannot resolve all the pairs of `navigateToMiniProgram` and `navigateBackMiniProgram`. For example, we noticed that only a few Baidu miniapps whose appIDs can be resolved. To really resolve the values of appIds, we may have to develop a value set analysis to dynamically compute them.

Second, we only studied CMRF against the cross-miniapp channel in this paper, and we believe that CMRF attacks only scratch the tip of iceberg. There could be multiple other attacks that exploit the cross-miniapp-channel. For example, from the attacker's perspective, similar to the native apps, we envision there may be collusion attacks through `navigateToMiniProgram` or `navigateBackMiniProgram`. More specifically, since a mini-app needs permission to access sensitive information (*e.g.*, location information, gender, places of residence, and phone numbers), a malicious mini-app with the access to sensitive resources can potentially leak the obtained sensitive data to other mini-apps without the required permission via the cross mini-app communication.

Finally, we only detect the vulnerability, and we believe that super app vendors can also offer some mitigation mechanisms (*e.g.*, by encrypting the channel so that attackers cannot perform the forgery). We leave the investigation of such mitigation in future work.

Ethics and Responsible Disclosure. We have followed community practices to avoid potential harms to developers or users: we only carried out the proof of concept attacks on our own accounts, devices, and miniapps. Also, in our case studies, instead of directly injecting payloads to or collecting sensitive data from the miniapps of other developers, we only inspect their logic to confirm vulnerabilities. Meanwhile, we have reported all our findings including the list of the vulnerable miniapps to Tencent in October 2021, and Baidu in April 2022. Tencent and Baidu both have acknowledged and confirmed our attacks, and notified us that they will also keep the third party developers posted, since the CMRF attacks have to be patched by the third-party developers. Additionally, Tencent has showed great interests of our CmrfScanner, and asked us for more technical details (*e.g.*, algorithms and implementations), which have been presented in this paper.

Super App	Vendors	AppID	Sending Request APIs	Location	Audio	Bluetooth	Camera	Multi-Media	Sport	UserInfo	Address	Invoice	File	Data Cache	Payment	AccountInfo	Coupon	PhoneNumber	Network
QQ	Tencent	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
WeChat	Tencent	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
WeCom	Tencent	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Baidu	Baidu	AppKey	navigateToSmartProgram, navigateBackSmartProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓			✓	✓
Taobao	Alibaba	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓			✓	✓
Alipay	Alibaba	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓			✓	✓
Tiktok	Bytedance	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓
JINRI Toutiao	Bytedance	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓
Watermelon Video	Bytedance	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓
Pipixia	Bytedance	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓
Toutiao Lite	Bytedance	appId	navigateToMiniProgram, navigateBackMiniProgram	✓	✓		✓	✓		✓	✓	✓	✓	✓				✓	✓

Table 7: Comparison of API implementations and resource accessed across different super apps.

9 RELATED WORK

Studies on Miniapp and Its Security. Since the mini-app is a novel paradigm, only few works have been made torwards this direction and most of them focus on understanding its architecture and applications. For example, Hao et al. [26] studied the system architecture and key technologies used by the WeChat miniapps. Regarding the applications, miniapps can be used on healthcare [43, 49], transportation [14], online shopping [36], education [13, 31, 41]. Today, there are millions of miniapps, and most of them are obfuscated, as reported by MiniCralwer [48].

Several attacks have also been discovered against the miniapps. For example, Lu et al. [32] studied the resource management vulnerabilities of mini-apps, and designed a few attacks to allow malicious mini-apps to collect the sensitive data provided from the host apps. Most recently, Zhang el al. [47] uncovered a novel identity confusion vulnerability caused by multiple reasons such as timing, frame, or URL parsing in an array of super apps and demonstrated concrete attacks with this vulnerability, such as bypassing security patches. Different from the existing works, our study focused on both the attacks and vulnerability detection of the novel CMRF attacks caused by missing checks in cross mini-app channel.

Cross-app Security. The cross-app security issues were first discovered in the web security domain, where two web apps with weak authentication can be subject to various attacks such as XSS [21, 23], CSRF [17], postMessage abuse [37], cross-domain requests (CORS) attacks [12, 29] and login CSRF [10]. Later, a set of detection frameworks (*e.g.*, with deep learning to help discriminate CSRF requests [11, 39], or property graph and program analysis [35]) and defenses (e.g, [24, 25]) were proposed to understand and defend against these attacks.

Cross-app security also exists in mobile apps (*e.g.*, [18, 20, 30, 44, 46]). For example, Chin et al. [15] investigated the inter-app communication channel and proposed a detection tool using app permission and intent contents as a detection proxy. In terms of the hijacking of cross-app communication channel such as Android Intent, Lu et al. [33] also proposed an approach to use static analysis to detect suspicious apps. On top of cross-app channels, Wang et al. [44] inspected more channels such as schema and webacessing channels, and showed that a wide range of attacks may work against these channels due to the lack of origin-based protection. Xing et al. [46] studied the cross-app resources sharing channels in MAC OS X and iOS, and discovered that unauthorized cross-app resource access attacks (XARA) can work against these devices. Li et al. proposed the Cross-App WebView Infection (XAWI) vulnerability [30] against the web views, which powers a series of multi-app, colluding attacks.

Our CMRF attacks are different compared to these existing works. First, our attacks exploit the cross-miniapp channel that uniquely exists in mini-app paradigm, and the workflow of this channel is completely different from both URL redirection in web apps, and intent mechanism in Android. Second, being mini by nature, the miniapps are heavily relied on the cross-miniapp channel to complete sophisticated tasks, and given the rich APIs provided by their super apps, the miniapps can now access a large range of sensitive resources (*e.g.*, invoice, sport data, user account info), and these access can cause severe security consequences if not properly implemented.

10 CONCLUSION

We have presented a novel cross-miniapp request forgery (CMRF) attack, which is caused by the missing checks of the sender's miniapp ID in a receiver miniapp. To understand how popular the miniapps are vulnerable to our attacks, we developed CmrfScanner, which has identified 52,394 (2.04%) WeChat miniapps and 494 (0.33%) Baidu miniapps that have used the cross-miniapp communication channel, and 50,281 (95.97%) of WeChat miniapps and 493 (99.80%) of Baidu miniapps are subject to CMRF attacks. Among all those miniapps that do not validate `appIDs`, there are 28,843 WeChat miniapps and 35 Baidu miniapps that have the privileges to access the sensitive data and can lead to severe security consequences such as shopping for free, promotion abuse, and device manipulations. Finally, we have disclosed our findings to the corresponding vendors, which have confirmed our attacks.

ACKNOWLEDGEMENTS

We would like to thank Chao Wang for his valuable insights on the security mechanism of super apps. We also thank anonymous reviewers for their invaluable comments. This research was supported in part by NSF awards 1834215 and 2112471. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and not necessarily of the NSF.

REFERENCES

[1] Authorization | weixin public doc. https://developers.weixin.qq.com/miniprogram/en/dev/framework/open-ability/authorize.html. (Accessed on 09/05/2022).

[2] China's tencent takes on the app store with launch of mini programs for wechat. https://techcrunch.com/2017/01/09/wechat-mini-programs/. (Accessed on 08/05/2022).

[3] Network | weixin public doc. https://developers.weixin.qq.com/miniprogram/en/dev/framework/ability/network.html. (Accessed on 09/05/2022).

[4] Snap minis. https://minis.snapchat.com/.

[5] Wechat active users worldwide 2022 | statista. https://www.statista.com/statistics/255778/number-of-active-wechat-messenger-accounts/. (Accessed on 09/05/2022).

[6] Wechat revenue and usage statistics (2022) - business of apps. https://www.businessofapps.com/data/wechat-statistics/. (Accessed on 09/05/2022).

[7] wx.requestpayment. https://developers.weixin.qq.com/miniprogram/en/dev/api/payment/wx.requestPayment.html. (Accessed on 09/05/2022).

[8] The total size of all subpackages of a Mini Program cannot exceed 12 MB. https://developers.weixin.qq.com/miniprogram/en/dev/framework/subpackages.html, 06 2020. (Accessed on 04/30/2022).

[9] Allison. Wechat mini-programs 2020: What your brand should know about this daily-life essential. https://daxueconsulting.com/wechat-mini-programs-2020-report/, 2020.

[10] Adam Barth, Collin Jackson, and John C Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88, 2008.

[11] Stefano Calzavara, Mauro Conti, Riccardo Focardi, Alvise Rabitti, and Gabriele Tolomei. Mitch: A machine learning approach to the black-box detection of csrf vulnerabilities. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 528–543, 2019.

[12] Jianjun Chen, Jian Jiang, Haixin Duan, Tao Wan, Shuo Chen, Vern Paxson, and Min Yang. We still {Don't} have secure {Cross-Domain} requests: an empirical study of {CORS}. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1079–1093, 2018.

[13] Xin Chen, Xi Zhou, Huan Li, Jinlan Li, and Hua Jiang. The value of wechat as a source of information on the covid-19 in china. *Bull World Health Organ*, 2020.

[14] Ao Cheng, Gang Ren, Taeho Hong, Kichan Nam, and Chulmo Koo. An exploratory analysis of travel-related wechat mini program usage: affordance theory perspective. In *Information and Communication Technologies in Tourism 2019*, pages 333–343. Springer, 2019.

[15] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, page 239–252, New York, NY, USA, 2011. Association for Computing Machinery.

[16] Breno Dantas Cruz and Eli Tilevich. Intent to share: enhancing android inter-component communication for distributed devices. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 94–104. IEEE, 2018.

[17] Philippe De Ryck, Lieven Desmet, Wouter Joosen, and Frank Piessens. Automatic and precise client-side protection against csrf attacks. In *European Symposium on Research in Computer Security*, pages 100–116. Springer, 2011.

[18] Wenrui Diao, Xiangyu Liu, Zhe Zhou, Kehuan Zhang, and Zhou Li. Mind-reading: Privacy attacks exploiting cross-app keyevent injections. In *European Symposium on Research in Computer Security*, pages 20–39. Springer, 2015.

[19] WeiXin Public Doc. wx.navigatetominiprogram. https://developers.weixin.qq.com/miniprogram/dev/api/navigate/wx.navigateToMiniProgram.html.

[20] Mohamed Elsabagh, Ryan Johnson, Angelos Stavrou, Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin. FIRMSCOPE: Automatic uncovering of Privilege-Escalation vulnerabilities in Pre-Installed apps in android firmware. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2379–2396, August 2020.

[21] Benjamin Eriksson, Pablo Picazo-Sanchez, and Andrei Sabelfeld. Hardening the security analysis of browser extensions. In *ACM Symposium On Applied Computing (SAC)*, 2022.

[22] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. Doublex: Statically detecting vulnerable data flows in browser extensions at scale. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1789–1804, 2021.

[23] Jeremiah Grossman, Seth Fogie, Robert Hansen, Anton Rager, and Petko D Petkov. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

[24] BB Gupta, Shashank Gupta, S Gangwar, M Kumar, and PK Meena. Cross-site scripting (xss) abuse and defense: exploitation on several testing bed environments and its defense. *Journal of Information Privacy and Security*, 11(2):118–136, 2015.

[25] Shashank Gupta and Brij Bhooshan Gupta. Cross-site scripting (xss) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of System Assurance Engineering and Management*, 8(1):512–530, 2017.

[26] Lei Hao, Fucheng Wan, Ning Ma, and Yicheng Wang. Analysis of the development of wechat mini program. In *Journal of Physics: Conference Series*, volume 1087, page 062040. IOP Publishing, 2018.

[27] Tencent. Inc. 55+ wechat statistics - 2022 update. https://99firms.com/blog/wechat-statistics/#gref.

[28] MANSOOR IQBAL. Tiktok revenue and usage statistics (2020). https://www.businessofapps.com/data/tik-tok-statistics/, 2020.

[29] Sebastian Lekies, Nick Nikiforakis, Walter Tighzert, Frank Piessens, and Martin Johns. Demacro: Defense against malicious cross-domain requests. In *International Workshop on Recent Advances in Intrusion Detection*, pages 254–273. Springer, 2012.

[30] Tongxin Li, Xueqiang Wang, Mingming Zha, Kai Chen, XiaoFeng Wang, Luyi Xing, Xiaolong Bai, Nan Zhang, and Xinhui Han. Unleashing the walking dead: Understanding cross-app remote infections on mobile webviews. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 829–844, 2017.

[31] Qinzhen Liang and Chengyang Chang. Construction of teaching model based on wechat mini program. *International Journal of Science*, 16(1):54–59, 2019.

[32] Haoran Lu, Luyi Xing, Yue Xiao, Yifan Zhang, Xiaojing Liao, XiaoFeng Wang, and Xueqiang 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.

[33] Long Lu, Zhichun Li, Zhenyu Wu, Wenke Lee, and Guofei Jiang. Chex: Statically vetting android apps for component hijacking vulnerabilities. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, page 229–240, New York, NY, USA, 2012. Association for Computing Machinery.

[34] Natalie Lui. Wechat mini programs: The complete guide for business. https://www.dragonsocial.net/blog/wechat-mini-programs/, 2020.

[35] Giancarlo Pellegrino, Martin Johns, Simon Koch, Michael Backes, and Christian Rossow. Deemon: Detecting csrf with dynamic analysis and property graphs. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1757–1771, New York, NY, USA, 2017. Association for Computing Machinery.

[36] Qianhui Rao and Eunju Ko. Impulsive purchasing and luxury brand loyalty in wechat mini program. *Asia Pacific Journal of Marketing and Logistics*, 2021.

[37] Sooel Son and Vitaly Shmatikov. The postman always rings twice: Attacking and defending postmessage in html5 websites. In *NDSS*, 2013.

[38] Business statistics. 90 baidu statistics and facts. https://expandedramblings.com/index.php/baidu-stats/, 2020.

[39] Marius Steffens and Ben Stock. Pmforce: Systematically analyzing postmessage handlers at scale. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 493–505, 2020.

[40] Mark 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/.

[41] Yiling Sui, Tian Wang, and Xiaochun Wang. 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*, 45:101707, 2020.

[42] W3C. Miniapp standardization white paper. https://w3c.github.io/miniapp/white-paper/, 2020.

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

[44] Rui Wang, Luyi Xing, XiaoFeng Wang, and Shuo 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.

[45] wechatwiki.com. Wechat data, insights and statistics: user profile, behaviours, usages, market trends. https://wechatwiki.com/wechat-resources/wechat-data-insight-trend-statistics/, 2019.

[46] Luyi Xing, Xiaolong Bai, Tongxin Li, XiaoFeng Wang, Kai Chen, Xiaojing Liao, Shi-Min Hu, and Xinhui Han. Cracking app isolation on apple: Unauthorized cross-app resource access on mac os˜ x and ios. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 31–43, 2015.

[47] Lei Zhang, Zhibo Zhang, Ancong Liu, Yinzhi Cao, Xiaohan Zhang, Yanjun Chen, Yuan Zhang, Guangliang Yang, and Min Yang. Identity confusion in webview-based mobile app-in-app ecosystems. In *31st USENIX Security Symposium (USENIX Security'22)*, 2022.

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

[49] Kaina Zhou, Wen Wang, Wenqian Zhao, Lulu Li, Mengyue Zhang, Pingli Guo, Can Zhou, Minjie Li, Jinghua An, Jin Li, et al. 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*, 106:103565, 2020.