Analysis of Information Flows and Privacy in the WeChat Ecosystem (Citizen Lab Report)

Mona Wang, Pellaeon Lin, and Jeffrey Knockel

Key findings

● This work performs the first analysis of WeChat's tracking ecosystem. Using reverse engineering methods to intercept WeChat's network requests, we identified exactly what types of data the WeChat app is sending to its servers, and when.

● We found that granting permissions such as precise geolocation affects the type and amount of data transmitted, including geolocation data and hardware serial numbers.

● We found that the most fine-grained activity tracking data is sent during Mini Program execution. All Mini Programs, and thereby their users, are enrolled in the tracking program, meaning that a large amount of users' app activity is sent to WeChat and not just the developers of the app themselves.

● We identify disclosure gaps with WeChat's privacy policy, which implies that only third-parties collect usage data related to Mini Programs, when, in fact, WeChat also collects this data.

● Some important features within WeChat, such as Advanced Search and Channels, are not governed by WeChat's own Privacy Policy. Instead, they are governed by Weixin's Privacy Protection Guidelines. The WeChat Privacy Policy states that these "third-party" services are "operated by Weixin". Usually, the Weixin Privacy Protection Guidelines apply in whole to users signing up with Chinese phone numbers. Because of this, a user's data might be subjected to a worse protection than the user thinks.

Introduction

With over 1.2 billion monthly active users, WeChat is the most popular messaging and social media platform in China and third in the world. According to some market research, network traffic from WeChat made up 34% of Chinese mobile traffic in 2018. WeChat has in many ways monopolized messaging in China, making it necessary for individuals in China to use. WeChat has also evolved beyond simply messaging. People commonly use WeChat as a social media platform to share updates with contacts, as a platform for conducting financial transactions, and also as a platform for downloading and using other programs, referred to as “Mini Programs.” WeChat has not only become the default way to contact people in China, but its ecosystem also encompasses many other necessities of daily life, like performing financial transactions or calling taxis. Many inside and outside China, therefore, use WeChat out of necessity. Besides individuals in China, diaspora populations, family members, journalists, International activists, diplomats, people who do business in China, and just about anyone with a relationship in China also use WeChat out of necessity. WeChat also complies with Chinese government and local police requests for data and information, essentially becoming a mass surveillance tool for local authorities. It also operates a massive content censorship ecosystem for the features on its platform.

Understanding what data the WeChat application and ecosystem transmits, and to whom, may be especially important due to the heavily automated surveillance and content censorship ecosystem operated by the platform. For vulnerable populations that must use WeChat (for instance, domestic journalists and foreign correspondents, grassroots and diaspora activists), knowing the limitations of the app can protect them. This kind of risk assessment requires a more granular understanding of information flows within the WeChat ecosystem.

In the case of WeChat, a large portion of network communications, including when messaging, viewing WeChat’s “Moments” posts, or sometimes even when using WeChat Mini Programs, utilize a proprietary encryption protocol called MMTLS. The closed-source and undocumented nature of this network protocol has also made it difficult for researchers to conduct a thorough review of information flows from the application. In our study we had to develop our own tools to enable us to thoroughly study the information that flows back to WeChat servers.

Our primary research question investigates what information flows back to WeChat servers across the WeChat ecosystem. To set the stage for this work, we first reverse engineer the networking stack in order to develop instrumentation and tooling to study WeChat network requests. In a followup report, we will further discuss our full understanding of the network stack, including the custom network security measures employed by WeChat to encrypt message data.

We then use this tooling to systematically categorize and study the composition of network requests flowing from the WeChat client to the server in various contexts in the WeChat ecosystem.

Background

WeChat is often described as a "super-app" not only due to the large number of features which it has accrued since its inception, but also because it is a popular platform for third-party applications. Originally developed as a chat app, the WeChat platform has grown to support voice calls, "Moments" social media posts, and file sharing. Specific to China, the app also serves as a major payment and financial transaction platform. Globally, the app also supports Mini Programs, which are a key feature relied on by a majority of WeChat users.
Despite WeChat's global popularity, the platform has come under repeated scrutiny for security and privacy issues. The platform lacks end-to-end encryption of chat messages, giving Tencent visibility into all messages sent over the platform. Users with mainland China accounts are subjected to automated, keyword-based censorship of messages, and images they post or send via chat message are censored according to the text contained in those images and according to their similarity to images on an internal blacklist. While such censorship is commonly believed to only affect users with mainland China accounts, previous work has found how even the communications of non-China users are subjected to political surveillance and used to build up WeChat's Chinese political censorship system.

What are Mini Programs and why are they important?

Mini Programs are lightweight applications that can be downloaded and launched within the

WeChat program, and can also sync and link with users' WeChat account and certain associated data, like their contact information. The breadth and variety of Mini Programs is essentially the same as any other application ecosystem, covering e-commerce, health, public services, gaming, and any other service you could feasibly imagine an app would be used for. Some Mini Programs deal with particularly sensitive data, such as health apps (e.g. Tencent Health), government service apps (e.g., the local contact tracing apps that were compulsory during the COVID-19 pandemic), or apps that perform financial transactions on behalf of the user (e.g., shopping apps like Pinduoduo or budgeting apps like "Small Ledger"/收款小账本).

WeChat implements Mini Programs by loading web pages into an embedded browser instance. Mini Program developers, then, can provide WeChat a web-page-like package(e.g. a set of packaged HTML-like, Javascript, and CSS files) to load into the embedded browser. WeChat also injects JavaScript into the web page. Among other purposes, some of these scripts provide a Javascript interface (under namespace "wx"). This interface allows the web-based Mini Program access to various mobile features, like geolocation, with WeChat acting as a bridge between the two. The interface also allows the Mini Program to access data associated with the user's WeChat account, such as payment information or the user's phone number.

The Mini Program ecosystem has also undergone criticism around various privacy and security issues. A 2020 report from CNCERT/CC found that 60 percent of the 50 banking applications that they investigated did not encrypt any user data transmitted over the network, among a litany of other common security issues. Another report investigated the third-party tracking ecosystem across 52 commonly used Mini Programs, found that many shared user data with third parties, and generally provided unsatisfactory notice and opt-outs to users.

Since the CNCERT report, WeChat has attempted to tighten user data controls within its Mini Program ecosystem. One such example is that web requests can no longer be directly made from the Mini Program, and instead must go through WeChat's internal API. This way, WeChat can enforce which third-parties are contacted during Mini Program usage, and also enforce minimum security for network requests, in particular by ensuring that they are all encrypted using HTTPS.

Despite the global popularity of WeChat and amount of attention to the Mini Programs

third-party ecosystem, what types of data WeChat transmits as well as what information flows from these third-party applications back to WeChat is understudied compared to other popular social media ecosystems like Facebook, and other tracking ecosystems like Google. Previous work has attempted to use automated methods to analyze the Windows desktop version of WeChat, finding that various identifiers related to a user's machine are transmitted to WeChat's servers.

Methods

In this section we explain our methods for analyzing WeChat's data flows and tracking. Our primary analysis method is the use of reverse engineering. In our report, we analyzed the Android version of the app, specifically version 8.0.23 released on May 26, 2022, downloaded from the WeChat website, and we used an account registered to a U.S. number for the analysis, which changes the behavior of the application compared to a mainland Chinese number. Our setup may not be comprehensive, and the full limitations are discussed in a section below.

Reverse engineering methods

In this work, we utilized both static and dynamic analysis methods, static referring to those methods which do not involve running an analyzed application and dynamic referring to those which do. For static analysis, we used Jadx, a popular Android decompiler, to decompile WeChat's Android Dex files into Java source code. We also used Ghidra and IDA Pro to analyze the native libraries bundled with WeChat.

For dynamic analysis, we analyzed the application installed on a rooted Google Pixel 4 phone and an emulated Android OS, using Frida to inject scripts and manipulate and export application memory. We also performed network analysis of WeChat's network traffic using Wireshark. However, due to WeChat's use of nonstandard cryptographic libraries like MMTLS, standard network traffic analysis tools that might work with HTTPS/TLS do not work for all of WeChat's network activity. Our use of Frida was paramount for capturing the data and information flows we detail in this report. These scripts are designed to intercept WeChat's request data immediately before WeChat sends it to its MMTLS encryption module. The Frida scripts we used are published in the Appendix.

Using these dynamic analysis methods, we then manually simulate multiple common “user activities” that a user may perform in practice. The discrete user activities we tested are as follows:

- Account creation
- Account login
- Loading messages
- Sending messages
    - Sending multimedia messages, including images and video
- Loading Moments feed
- Posting Moments
    - Posting multimedia moments, including images and video
- Commenting and interacting with moments
- Mini Program usage

For each of these activities we identify the destination of all network requests made during them and record the types of data transmitted during them. We collect data under two primary modes: a “permissive” and “restrictive” mode. In a “permissive” mode, we grant the application all sensitive permissions, to demonstrate to what extent permissioned APIs are relied on during regular operation of the app.

To detect sensitive content flows, we identify various pieces of sensitive data related to the current user. For certain fields that are user-controllable, we use identifiable strings of text. We then attempt to find these pieces of data in WeChat’s plaintext, uncompressed transmissions. For instance, we identified the following information to track their transmissions:

- Network carrier
- Phone number
- User’s internal WeChat ID
- OS details – Android API version
- Device details – model, brand

For spoofable numeric data, such as geolocation coordinates or screen size on an Android emulator, we search for the combination of certain numbers (such as both the width and height, or all three geocoordinates) to avoid matching false positives. For fully user-controllable fields, we inject recognizable and descriptive names, such as “USERNAME_XXXXX” for the name of the account. A full list of canaries and identified information used to track content flows can be found in the Appendix.

Mini Programs

In order to study WeChat behavior during Mini Program usage, we load “WeChat Example Mini Program,” which is a test application that can trigger various API calls. We also collect data from five popular Mini Programs across different categories from 2021 from a sample of posts, based on usage and search metrics, since there is no Mini Program ordering provided by WeChat [1] [2]. The† five Mini Programs we studied were Meituan (food), Pinduoduo (online shopping), Didi Chuxing (taxi service), Tongcheng Travel (tourism), and Kuaishou (online video).

Limitations

This investigation only looks at client behavior, and is subject to other common limitations in privacy research that can only perform client analysis. Much of the data that is sent to WeChat servers may be required for functionality of the application. For instance, WeChat servers can certainly see chat messages since WeChat can censor them according to their content. We cannot say what WeChat is doing with the data that they collect, but we can make inferences about what is possible. Certain limited inferences about data sharing can be made based on prior work like this report, which indicates that messages sent by non-mainland-Chinese users are used to train censorship algorithms for mainland Chinese users. In this report, we focus on the version of WeChat for non-mainland-Chinese users.

Our investigation was also limited due to legal and ethical constraints. It has become increasingly difficult to obtain Chinese phone numbers for investigation due to the strict phone number and associated government ID requirements. Therefore, we did not test on Chinese phone numbers, which causes WeChat to behave differently. In addition, without a mainland

Chinese account, the types of interaction with certain features and Mini Programs were limited. For instance, we did not perform financial transactions on the application.

We imposed additional constraints to limit the scope of the work. We only investigated the Android application, we only performed the study on one Android API version (Android 32) and a rooted Google Android Pixel device (Pixel 4), including an emulator for that same device. Though we did not observe any operational differences in behavior between using the emulator and a regular device, there may be downstream effects to our use of an emulator to collect much of the data in this report.

We also only evaluated a recent version of WeChat (8.0.23 released on May 26, 2022). Testing different versions of WeChat, the backwards-compatibility of the servers with older versions of the application, and testing on a variety of Android operating systems with variations in API version, are great avenues for future work.

We also acknowledge that the scope of the Mini Programs work is limited compared to the size of the entire ecosystem, as we only take a few Mini Programs to investigate thoroughly. We attempt to offset this by focusing on the relationship between Mini Program API calls and data sent back to WeChat servers. A comprehensive and representative overview of the entire ecosystem would be a great avenue for future work.

Finally, the WeChat codebase is vast. Over the years, the application has grown to cover various features, including ones that may transmit especially sensitive data (such as linking financial data and performing financial transactions). In this work, we do not purport to cover all of them. In fact, our research packages our preliminary understanding of WeChat. We provide our most accurate understanding of the codebase, but acknowledge that there exist limitations in our methodology and the vast nature of the WeChat application.

Findings

In this section we detail our findings concerning WeChat's tracking during regular application usage as well as during mini-program usage. The structure of captured data can be found in the

Appendix.

First-party tracking on Mini Programs

The data collection observed on Mini Programs is likely in-place to enable the application monitoring and analytics features provided by WeChat, namely, "We分析" or "WeAnalyze". However, from our analysis, we find that all Mini Programs are automatically enrolled into the WeAnalyze program and data collection, and there is no reasonable way to opt-out. To put this data collection into perspective, it would be an equivalent privacy violation if Google Play Store automatically injected Google Analytics tracking scripts into all applications that were available on the platform.

During regular Mini Program usage, user interactions are sent back to WeChat servers. In addition, extremely verbose logging data is sent to WeChat servers during Mini Program usage; we do not observe this type of logging data sent during any of our other experiments. Many of the requests observed, especially extraneous logging and activity tracking, do not serve a baseline functionality for the Mini Programs, instead records and tracks user behavior across the program.

Details of tracking during Mini Program usage

The class JsApiOperateRealtimeReport (pinging endpoint /wxartrappsvr/route), send the following data back to WeChat servers:

- appid,appversion
- page_path (i.e. the current user view)
- click data and time spent on this particular view
- networktype
- language/locale
- device information: model, OS, OS version, device brand, SDK version
- screen size

AppBrandIDKeyBatchReport (/wxausrevent/wxaappidkeybatchreport) sends a large stream of

logging data, including API calls made by the Mini Program and other internal API calls made by the host platform, to WeChat.

Finally, the class JsApiOperateWXData is also responsible for sending data back to WeChat servers, typically containing a Javascript API name and a custom serialized JSON blob of data. The JSON data sent in this object is primarily scoped to the functionality of the Javascript API at hand. Some of these network requests to WeChat servers may be necessary for the type of operation with the application; for instance, if the third-party application requests data about the logged-in WeChat user.

Infrastructure and server locations

On testing from our vantage point in Canada, when logged out, the application primarily makes MMTLS requests to the domain **hkextshort.weixin.qq.com**. Once logged in, the application switches to using **sgshort.wechat.com** for MMTLS requests. Presumably, the prefixes “hk” and “sg” intend to refer to servers located in Hong Kong and Singapore, as outlined in WeChat’s Privacy Policy. As expected, these hostnames resolved to IP addresses owned by Tencent, WeChat’s parent company.

Upon startup, the application sends a regular HTTP GET request to the endpoint, *http://dns.weixin.qq.com/cgi-bin/micromsg-bin/newgetdns*, which is a formatted list of WeChat server domain names and associated IP addresses. This is combined with other MMTLS requests (for instance, we also observe MMTLS requests to */cgi-bin/micromsg-bin/getcdndns*) to resolve certain IP addresses that the client might choose from. The full contents of this endpoint as we observed it can be found on our Github.

In this file, we can identify a number of other prefixes, including “ml”, “sh”, “sz”, in addition to the “hk” and “sg” prefixes we observed. “sz” may stand for Shenzhen, the location of Tencent’s headquarters. Though it was not observed in our testing, requests to “szshort.weixin.qq.com” are commonly observed in online posts, possibly from developers located in China. This domain is also mentioned in a test case in some of Tencent’s open-source networking code.

In general, we did not observe changes in the region prefix that was preferred by MMTLS

requests. However, cross-referencing from existing data and observations by other researchers, we can conclude that the servers that are preferred by the WeChat application depends on a combination of IP address location, especially if the user is logged out, and the registered phone number locale (if the user is logged in).

Opportunistic permissions

WeChat makes use of some dangerous permissions (as defined by Google) , in particular “Location” and “Files and media”, during regular operation of the application, if those permissions are granted to the application.

In addition, a network request on opening the app contains an encrypted header containing device serial numbers, specifically the IMEI[1] and IMSI[2]. On Android 10 devices and above, where the API no longer permits obtaining the IMEI, a dummy string “1234567890ABCDEF” is sent in place of the IMEI. Operating systems that do not have this protection in place would likely send the full IMEI.

These data points demonstrate that WeChat makes opportunistic use of device access to many types of sensitive data, which means hardening the operating system and device access to certain permissions does restrict the amount and type of data that is available to WeChat. However, in practice, this landscape is complicated by the fact that WeChat also manages the permissions for its Mini Programs.

Any particular Mini Program can request an OS-level sensitive permission, such as reading storage or accessing the user location. In order to grant any Mini Program a particular permission, then, by design, WeChat must also request that permission from the operating system. There is no way to enable a particular permission for a Mini Program but not for the host WeChat application.

---

[1]International Mobile Equipment Identity. This is an unique identifier for every phone

.[2]International Mobile Subscriber Identity. This is an unique identifier for every SIM card.

WeChat's Mini Program ecosystem is often compared to an application ecosystem for a particular operating system; in this analogy, WeChat, or other super-apps, are akin to an operating system. OS-level permissioning systems have been studied at length in the past, and there are an increasing number of studies on the misuse of permission boundaries within super-apps.

Tracking during regular application usage

Here, we describe notable personal data tracking during general/regular application usage, outside the scope of what may be necessary for the operation of the WeChat application.

First, many WeChat MMTLS request includes, as a header:

- Platform data, specifically OS API version
- User's internal WeChat UID

Data description Example
Operating system details sdk_gphone64_arm64arm64-v8a
WeChat internal UIN a 10-digit number
WeChat internal UID ("MMGUID") a 15 character hex string"A9543d226cb39f0f_1683146442889"
API version android-32

Table TKTK:

When the user opens the WeChat application, or logs into the WeChat application, a comprehensive device report is sent, including:

Data description Example
Phone number Our U.S. testing phone number
IMEI 1234567890ABCDEF (for Android 10+)
Operating system details sdk_gphone64_arm64arm64-v8a
Manufacturer Google
Major OS release version 12 (Android 12)
Minor OS release version 8015633
Build display name sdk_gphone64_arm64 12 S2B2.211203.006 8015633

Network carrier T-Mobile

Table TKTK:

WeChat sends various batch reports that send application usage and monitoring data, as well as location data, if particular permissions are enabled by the user. A class called “CliReportKV” that collects and sends a large series of operational messages that are collected over a large period of time, and “NetTypeReporter” sends networking and location data at a regular interval back to WeChat servers.

Finally, we find that when running with all permissions enabled, WeChat accesses the location API on any regular startup of the application, while logged in. Enabling location permissions automatically enables the “People Nearby” feature, which broadcasts your location so you can identify and interact with WeChat users nearby.

Discussion

Privacy policies and Weixin as “third party”

WeChat has a different privacy policy for its users, depending on whether the account is attached to a mainland Chinese mobile number. The WeChat privacy policy refers to accounts with mainland Chinese mobile numbers as “Weixin accounts,” which are covered under a separate privacy policy. “Weixin” is a direct transliteration of WeChat’s Chinese name. We refer to these, respectively, as the WeChat privacy policy and the Weixin privacy policy.

Within the WeChat privacy policy, Weixin is referred to as a third party. The WeChat privacy policy also defers to the Weixin privacy policy when it comes to first-party data collection on various features it designates as “operated by Weixin”. In general, we note that various key features and services, such as Advanced Search and Channels, that tend to collect more user
data, are all considered “third party services operated by Weixin” according to WeChat’s privacy policy. This separation between WeChat and Weixin features is not communicated within the application itself. Unlike other popular apps like Tiktok, and its Chinese counterpart, Douyin, network data for both “WeChat services” and “Weixin services” all go to the same server, which seems to be determined by the user’s IP address or phone number.

We identify additional privacy disclosure issues under this WeChat and Weixin dichotomy, and inconsistencies between the wording of these privacy policies to the observed behavior of the app.

First, the WeChat privacy policy states that it will only share data with Weixin as necessary. However, app usage tracking for analytics is not necessary for the operation of the platform. In addition, we note that prior research found that non-mainland-Chinese user data was being used to train censorship algorithms for mainland-Chinese users.

Second, the WeChat privacy policy implies that only third-party privacy practices and policies govern Mini Programs, when in fact, WeChat/Weixin also collects lots of data. In fact, Mini

Programs are *not* listed as subject to the Weixin privacy policy, and instead listed under “Weixin Open Platform,” which are only governed by third-party privacy policies.

The Weixin privacy policy does mention that when using “Weixin’s Mini Programs,” they will collect data related to “logging in to, browsing, and using Mini Programs.” The wording of this translation is vague as to whether it refers to Mini Programs directly operated by Weixin, or all Mini Programs available on the Weixin platform. There is no opt-out for this feature on either the developer (third party) or the user’s behalf.

Recommendations

From our discussions above, we make the following recommendations to the platform and platform users to improve user privacy.

Recommendations for WeChat:

● Remove forced enrollment of Mini Program analysis and tracking features, and change to an opt-in model. Currently, both developers and users are automatically enrolled into the We分析 tracking program with little notification. There is currently no way to opt out of the program.

● Enact a more fine-grained permissioning model. Certain Mini Programs might need permissions, but users should have some guarantee that the host platform cannot abuse those permissions if granted.

● Remove the delineation between “Weixin” vs “WeChat” services in the Privacy Policy. Despite that the Privacy Policy claims certain core features of “WeChat” are run by a third-party named “Weixin,” there is no such delineation between “WeChat” and “Weixin” services in the regular operation of the application.

● Disclose WeChat’s first-party collection of Mini Program user data in the WeChat Privacy Policy.

● Allow users to opt-out of analytics tracking during usage of “Weixin” services. Users should be able to opt-out of tracking that is not necessary to the function of the app.

Recommendations for users:

● Avoid features delineated as “Weixin” services if possible. We note that many core “Weixin” services (such as Search, Channels) as delineated by the Privacy Policy perform more tracking than core “WeChat” services.

● Use stricter permissions. In modern versions of Android, it is possible to restrict certain permissions (like location access) to when the application asks for it. Given the opportunistic use of enabled permissions by the application and the lack of a more fine-grained permissioning model between Mini Programs and the host WeChat platform, we recommend locking down application permissions.

● Update your device’s OS regularly for security features. Many new security features on modern versions of Android are working correctly to enforce permission boundaries and limit certain types of identifiers that are available to the application (such as IMEI). We recommend using an OS that has all of these features in place, and regularly updating for additional security features down the line.

We note that these improvements are incremental at best, and recommendations for users should be taken in context with a particular threat model. For users with certain high risk profiles, no amount of these adjustments will make WeChat completely safe to use. While we would generally suggest using alternative messaging applications if possible, we understand that most users use WeChat out of necessity, and that using more secure messaging applications can itself be a risk factor depending on the particular situation.

Acknowledgments

We would like to thank TKTK

Appendix

Program components related to network requests

The contents of each network message we captured included a serialized Protobuf object and a request URI that is internal to WeChat operation. From the deserialized, unencrypted data contents, associated metadata, and static analysis of code paths around these requests, we can infer the purpose and usage of this particular data.
WeChat internally provides a unified Java API for components to make network requests. Each API endpoint is represented in a Java class (which we refer to as “API class”) that is extended from the NetSceneBase class.

Each API class defines the following important properties:

- Internal URI, usually starts with /cgi-bin/.
- Request type, usually a 3 digit integer.

The most important API interfaces during request making are:

- NetSceneQueue.checkAndRun(): Starting a request of a specific API class. This function is a fixed implementation in NetSceneBase. This function is invoked from external components that need to make network requests.
- NetSceneBase.doScene() : Filling data fields. Implemented in API class. ● GYNetEndI.onGYNetEnd() : Callback when response is received. Implemented in the API class.

In the rest of this section, we illustrate the structure outlined above with a real API class: NetSceneEncryptCheckResUpdate. Note that most of the symbol names are given by us based on our understanding of its purpose.

```java
/* renamed from: com.tencent.mm.pluginsdk.res.downloader.checkresupdate.o */
/* loaded from: classes10.dex */
public final class NetSceneEncryptCheckResUpdate extends AbstractNetsceneCheckresUpdate {
    @Override // com.tencent.p486mm.pluginsdk.res.downloader.checkresupdate.AbstractNetsceneCheckresUpdate
    protected final String getTag() {
        return "MicroMsg.NetSceneEncryptCheckResUpdate";
    }

    /* renamed from: dig */
    public static void checkandrun() {
        MMKernel.getmCoreNetwork().netscenequeue.checkAndRun(new NetSceneEncryptCheckResUpdate(), 0);
    }

    @Override // com.tencent.p486mm.pluginsdk.res.downloader.checkresupdate.AbstractNetsceneCheckresUpdate
    /* renamed from: i */
    protected final BaseResponsePBMessage mo19192i(MMTLSConnection mMTLSConnection) {
        return ((EncryptCheckResUpdateRR) mMTLSConnection).abbP.abDR;
    }

    @Override // com.tencent.p486mm.p501al.NetSceneBase
    public final int getType() {
        return EcdhMgr.auth_info_prefs_use_new_ecdh ? 784 : 722;
    }
```

Figure TKTK:

In NetSceneEncryptCheckResUpdate, the *request type* is either 784 or 722 depending on the value of a configuration EcdhMgr.auth_info_prefs_use_new_ecdh.

NetSceneEncryptCheckResUpdate's request URI is defined in its subclass EncryptCheckResUpdateRR. The URI also depends on the configuration value EcdhMgr.auth_info_prefs_use_new_ecdh.

```java
/* renamed from: com.tencent.mm.pluginsdk.res.downloader.checkresupdate.o$a */
/* loaded from: classes10.dex */
static class EncryptCheckResUpdateRR extends ReqRespBase {
    private final MMEncryptCheckResUpdate.EncryptCheckResUpdateReq abbO = new MMEncryptCheckResUpdate.EncryptCheckResUpdateReq();
    private final MMEncryptCheckResUpdate.EncryptCheckResUpdateResp abbP = new MMEncryptCheckResUpdate.EncryptCheckResUpdateResp();

    @Override // com.tencent.p486mm.p501al.ReqRespBase, com.tencent.p486mm.network.MMTLSConnection
    public final int getOptions() {
        return 1;
    }

    @Override // com.tencent.p486mm.p501al.ReqRespBase
    public final MMBase.Req getReqObjImp() {
        return this.abbO;
    }

    @Override // com.tencent.p486mm.network.MMTLSConnection
    public final int getType() {
        return EcdhMgr.auth_info_prefs_use_new_ecdh ? 784 : 722;
    }

    @Override // com.tencent.p486mm.network.MMTLSConnection
    public final String getUri() {
        return EcdhMgr.auth_info_prefs_use_new_ecdh ? "/cgi-bin/micromsg-bin/secencryptcheckresupdate" : "/cgi-bin/micromsg-bin/encryptcheckresupdate";
    }
```

Figure TKTK:

NetSceneEncryptCheckResUpdate inherits most of its properties and methods from AbstractNetsceneCheckresUpdate, which implements doScene.

```java
/* renamed from: com.tencent.mm.pluginsdk.res.downloader.checkresupdate.n */
/* loaded from: classes10.dex */
public abstract class AbstractNetsceneCheckresUpdate extends NetSceneBase implements GYNetEndI {

    /* renamed from: abbL */
    protected static final SparseArray<InterfaceC69184a> sparsearray1 = new SparseArray<>();

    /* renamed from: abbK */
    protected final List<Res4IntListPBM> list1 = new LinkedList();
    private volatile OnSceneEndI callback;

    protected abstract String getTag();

    /* renamed from: i */
    protected abstract BaseResponsePBMessage mo19210i(MMTLSConnection mMTLSConnection);

    protected abstract MMTLSConnection iUq();

    /* renamed from: a */
    public static void m19212a(InterfaceC69184a interfaceC69184a) {
        sparsearray1.put(39, interfaceC69184a);
    }

    /* JADX INFO: Access modifiers changed from: package-private */
    public AbstractNetsceneCheckresUpdate() {
        int[] iArr;
        for (int i : CheckResUpdateConstants.abbn) {
            InterfaceC69184a interfaceC69184a = sparsearray1.get(i);
            if (interfaceC69184a == null || !interfaceC69184a.mo19252Iw(i)) {
                Res4IntListPBM res4IntListPBM = new Res4IntListPBM();
                res4IntListPBM.woO = i;
                this.list1.add(res4IntListPBM);
            }
        }
    }

    @Override // com.tencent.p486mm.p580al.NetSceneBase
    public final int doScene(NetworkConnectionInterface networkConnectionInterface, OnSceneEndI onSceneEndI) {
        C69255p c69255p;
        List<C69259r> emptyList;
        List emptyList2;
        this.callback = onSceneEndI;
        Log.m17332d(getTag(), "before dispatch");
        try {
            for (Res4IntListPBM res4IntListPBM : this.list1) {
                int i = res4IntListPBM.woO;
                c69255p = C69255p.C69256a.abcw;
                SqliteDB sqliteDB = !c69255p.suw ? null : c69255p.abct.pdV;
                if (sqliteDB == null) {
                    emptyList = Collections.emptyList();
                } else {
                    Cursor query = sqliteDB.query("ResDownloaderRecordTable", null, "urlKey" + String.format(Lo
                    if (query != null && !query.isClosed()) {
                        if (query.moveToFirst()) {
```

Figure TKTK:

AbstractNetsceneCheckresUpdate also implements onGYNetEnd.

Figure TKTK:

Finally, when is this request sent? We see NetSceneQueue.checkAndRun() called with NetSceneEncryptCheckResUpdate only in a simple wrapper within the class itself:

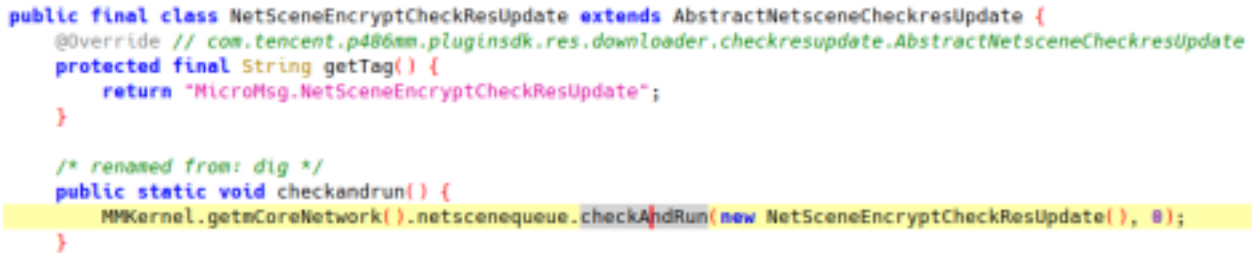

Figure TKTK:

The wrapper checkandrun() is called from com.tencent.mm.ui.LauncherUI.onCreate(), which is called during the application startup. This fits our traffic observation that API request /cgi-bin/micromsg-bin/secencryptcheckresupdate gets sent during application startup.

Frida script

// introduce the components that our frida script hooks into

// explain why our frida script can intercept plaintext requests

Table of identified network requests and associated data

Application startup (logged out)

/cgi-bin/micromsg-bin/secencryptcheckresupdate

Account login

/cgi-bin/micromsg-bin/bindopmobileforreg

/cgi-bin/micromsg-bin/secmanualauth

/cgi-bin/micromsg-bin/getcdndns

/cgi-bin/micromsg-bin/checkresupdate

/cgi-bin/mmbiz-bin/usrmsg/getserviceapplist

/cgi-bin/mmexptappsvr-bin/getexptconfig

/cgi-bin/mmfddataappsvr/getexptappconfig

/cgi-bin/micromsg-bin/getopenimresource

/cgi-bin/micromsg-bin/textstatusgetuserpermission

/cgi-bin/micromsg-bin/textstatusgeticonconfig

/cgi-bin/micromsg-bin/prconfig

/cgi-bin/micromsg-bin/finderinit

/cgi-bin/micromsg-bin/findersync

/cgi-bin/mmbiz-bin/wxaapp/getwxausagerecord

/cgi-bin/mmbiz-bin/wxaapp/getpubliclibinfo

/cgi-bin/micromsg-bin/getprofile

/cgi-bin/micromsg-bin/getforcepush

/cgi-bin/micromsg-bin/oplog

/cgi-bin/micromsg-bin/getboundharddevices

/cgi-bin/micromsg-bin/get_user_bind_iot_device_info

/cgi-bin/mmbiz-bin/wxausrevent/getservicenotifyoptions

/cgi-bin/micromsg-bin/newinit

/cgi-bin/micromsg-bin/reportclientcheck

Application startup (logged in)

/cgi-bin/micromsg-bin/textstatusgeticonconfig

/cgi-bin/micromsg-bin/androidfcmreg

/cgi-bin/micromsg-bin/checkresupdate

/cgi-bin/micromsg-bin/secautoauth

/cgi-bin/micromsg-bin/finderinit

/cgi-bin/micromsg-bin/findersync

/cgi-bin/micromsg-bin/getcdndns

/cgi-bin/micromsg-bin/statusnotify

/cgi-bin/micromsg-bin/newsync

Mini Program Open

/cgi-bin/mmbiz-bin/js-operatewxdata

/cgi-bin/micromsg-bin/newreportkvcomm

/cgi-bin/mmbiz-bin/wxausrevent/wxaappidkeybatchreport

/cgi-bin/mmbiz-bin/wxartrappsvr/route

Mini Program Download