# KakaoTalk Disclosures

## Summary

We identify 4 issues in KakaoTalk 5.5.5 (Android), 4 issues in KakaoTalk's "Secure Chat" end-to-end encryption protocol, and 3 design issues that could impact user privacy.

Below is a list of all the identified issues:

| **KakaoTalk 5.5.5 Security Issues** | | |
|---|---|---|
| **Issue Number** | **Description** | **Severity** |
| KT-01-001 | Improper WebView implementation | Critical |
| KT-01-002 | Insufficient Transport Layer Protection | Info |
| KT-01-003 | WebViews Ignore Certificate Validation Warning Messages | Info |
| KT-01-004 | Misconfigured TLS Endpoints | Info |
| **"Secret Chat" End-to-end Encryption Security Issues** | | |
| KT-01-005 | Lack of Message Integrity | Medium |
| KT-01-006 | Vulnerable to Known Plaintext attacks | Medium |
| KT-01-007 | Missing Freshness | Info |
| KT-01-008 | Static Values | Info |
| **KakaoTalk 5.5.5 Privacy Protection Issues** | | |
| KT-01-009 | Information Leakage | Info |
| KT-01-010 | User Tracking | Info |
| KT-01-011 | Spam | Info |

Out of these 11 issues we highlight the following 4 as the most important to address:

**Security Vulnerabilities**

- Improper WebView implementation (Critical)
    - Proposed fix: Do not allow third-parties (e.g., “Plus Friend” advertisers) to access the *addJavascriptInterface()* method in a WebView. One solution could be to increase the *minSdkVersion* from 14 to 17 (JELLY_BEAN_MR1) in the AndroidManifest.xml file.

- Lack of Message Integrity (Medium)
    - Proposed fix: Consider using an Authenticated Encryption (AE) block cipher mode such as Galois/Counter Mode (GCM).

**Design Issues**

- Insufficient Transport Layer Protection (Info)
    - Proposed fix: Enforce TLS and certificate pinning across all channels. Make end-to-end encrypted messaging a default feature (similar to WhatsApp). Provide informative certificate warning messages to the user in case there is a TLS error (similar to warning messages provided by modern web browsers). Users should know about the consequences of a (possibly) compromised TLS connection.

- Misconfigured TLS Endpoints (Info)
    - Proposed fix: Disable deprecated TLS cipher suites on all production servers.

Full details for every issue is provided in the subsequent sections.

# KakaoTalk 5.5.5 Analysis

KakaoTalk's 5.5.5 main software vulnerabilities are as follows (Android version):

## KT-01-001 Improper WebView Implementation (Critical)

As a hybrid mobile application, KakaoTalk makes heavy use of WebKit which is an open source web browser engine. KakaoTalk uses the framework's core view class WebView [6] to display web content that the application downloads from backend servers. We discovered a number of possible vulnerabilities due to the way how KakaoTalk has implemented its WebViews: On the one hand, the application uses various methods of the android.webkit.WebSettings class to configure its WebViews. We found that for almost all WebViews KakaoTalk is calling the following methods:

- **setJavaScriptEnabled()**: This method allows to execute JavaScript within a WebView.
- **setAllowFileAccess():** The setAllowFileAccess() method allows KakaoTalk to read cached web content from the file system. This feature is enabled by default if the developer has not explicitly disabled it.
- **setPluginState():** KakaoTalk calls this method in its `InAppBrowserActivity` to enable plugin support (e.g., to display Adobe Flash content).

On the other hand, KakaoTalk is making use of the `addJavascriptInterface()` public method of the WebView class. The method implements a Java to JavaScript bridge - which KakaoTalk declares either as “kakaotalk”, “kakaoTalk”, or “Kakao” - that can be used to allow JavaScript code to control the native application. This feature makes KakaoTalk vulnerable to possible remote code execution attacks if an attacker uses the bridge to access the `getClass()` method which is inherited from the `Object` class [7] (see CVE-2012-6636 and CVE-2013-4710). The attack works only on devices running Android versions older than 4.2 [8]. Worryingly, not only a MITM attacker but also any third-party brand or pop artist that advertises its products through KakaoTalk's “Plus Friend” marketing channel is able to access the JavaScript bridge. This means that any third-party may be able to execute arbitrary code if Kakao does not verify the integrity of web scripts prior publishing them on “Plus Friend”.

As a result of these WebView misconfigurations, an attacker may be able to inject arbitrary JavaScript (JS) code since KakaoTalk makes use of unauthenticated channels and also accepts invalid TLS certificates (see next section). In a quick test, we injected simple JS statements such as `alert(document.cookie);` which were then executed in the context of the WebView. Also, we injected a false login page that may be used to phish user credentials and were able to retrieve a list of available HTML5 file objects (`window.File`, `window.FileReader`, and `window.FileList`). These objects may allow to read files from the file system even though we were not successful most probably due to same-origin policy (SOP) restrictions.

## KT-01-002 Insufficient Transport Layer Protection (Info)

KakaoTalk utilizes the HTTPS protocol for most network connections in order to prevent passive eavesdropping of the communication link. The application takes a number of important steps to validate the server's X509 certificate: First, it checks whether the certificate's Common Name (CN) matches the domain's name. If they do not, KakaoTalk throws a *CertificateException* and rejects establishing a TLS connection to the server. Second, KakaoTalk verifies whether the server's certificate has been signed by a trusted Certificate Authority (CA). For this, the application checks if the signing CA certificate has been signed by a trusted Root CA certificate which is located in Android's trusted CA store. At this point, an attacker may be still able to compromise a TLS session by luring the user to copy his own malicious CA certificate into the victim's trusted CA store. For this reason, KakaoTalk also verifies the certificate chain by maintaining a whitelist of public keys that are trusted to sign certificates. This list is consulted when KakaoTalk validates the certificate chain, and if it does not include at least one of the *pinned* keys, certificate validation fails. This security concept is called “public key pinning” or “certificate pinning” and was proposed in RFC 7469 [3] by Google in 2015.

Despite of implementing these certificate validation techniques, KakaoTalk accepts self-signed MITM certificates for the domains *auth.kakao.com*, *auth.kakaocdn.net* and *katalk.kakao.com* after a user accepts a certificate validation error message. The pop-up message says that “There are problems with the security certificate for this site” and allows the user to either accept the error or to go “Back”. After accepting the dialog, KakaoTalk detects that the server's certificate chain is invalid and writes a *java.security.cert.CertPathValidatorException* warning message to the system log. However, the app ignores this error and continues working by silently accepting invalid TLS certificates in the background. This would allow an attacker from this time on to read user credentials (KakaoTalk ID and password) and OAuth authentication tokens. If the user taps on the “Back” button, KakaoTalk does not perform any action or shows a blank WebView with a message saying "Unstable network connection".

## KT-01-003 WebViews Ignore Certificate Validation Warning Messages (Info)

We found that for some WebViews KakaoTalk does not display a certificate validation error message. One non-critical example is KakaoTalk's privacy policy and terms of service which are displayed in a WebView that loads the HTML from https://1.201.0.61/android/account/privacy.html?locale=en and https://1.201.0.61/android/account/terms.html?locale=en. Although the server cannot prove that it is “1.201.0.61” as its certificate is from “*.kakao.com”, the WebView does not notify the user that the server’s hostname does not match the certificate’s Common Name.

## KT-01-004 Misconfigured TLS Endpoints (Info)

In addition to the certificate validation issues described above, we found that some TLS endpoints of KakaoTalk's Web API backend are misconfigured. For this analysis, we used Qualys TLS Server Tes [4] in order to scan the backend for common TLS configuration flaws. We tested the domains *ac-talk.kakao.com*, *auth.kakao.com*, *katalk.kakao.com*, and *auth.kakaocdn.net* and retrieved the following scanning results:

**auth.kakao.com**

- The server is vulnerable to the POODLE attack
- The server is vulnerable to the DROWN attack
- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107)
- Weak certificate signature algorithm (SHA1withRSA)
- The server supports weak cipher suites (e.g., TLS_RSA_WITH_RC4_128_MD5)
- The server does not support Forward Secrecy

**ac-talk.kakao.com and auth.kakao.com**

- The servers are vulnerable to the DROWN attack
- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107 [5])
- Weak certificate signature algorithm (SHA1withRSA)
- The servers support weak cipher suites (e.g., TLS_RSA_WITH_RC4_128_MD5)
- The servers do not support Forward Secrecy

**auth.kakaocdn.net**

- OpenSSL padding oracle vulnerability in AES-NI CBC MAC check (CVE-2016-2107)
- The server supports weak Diffie-Hellman (DH) key exchange parameters
- Weak certificate signature algorithm (SHA1withRSA)
- The server does not support Forward Secrecy

## Miscellaneous Issues

### TLS Not used for all Network Connections

Finally, we detected that KakaoTalk does not use TLS for all network connections. For instance, the application does not use HTTPS to secure the communication channel on older versions of Android. On a mobile handset that ran Android 4.1.2 we found that all communications to KakaoTalk's Web API backend are handled by the HTTP protocol only. Moreover, while running KakaoTalk on a more recent Android version, we recognized that some web content is still being served via HTTP.

For instance, KakaoTalk downloads unauthenticated HTML:
http://1.201.0.61/android/help?lang=en&country_iso=DE&a=android%2F5.5.5%2Fen
and JavaScript: http://ad1-kant.kakao.com/track_jquery3.js when the user opens the com.kakao.talk.activity.setting.HelpActivity and com.kakao.talk/.activity.setting.NoticeActivity. We also noticed that several resources including images and zip archives are being downloaded unencrypted. This would allow a MITM attacker to inject arbitrary code that may be then executed on the user's mobile phone (see “Improper WebKit WebView implementation” below).

### KakaoTalk's OAuth Access Token may Lack Randomness

Since the key generation algorithm is implemented on the server-side we do not know how the access tokens are generated. We did not try to feed the key generator with different input (e.g., a different device UUID or phone number) or otherwise investigate any further. However, by performing a simple String comparison of different access tokens we found potential evidence that the access tokens may lack randomness. From our basic analysis it seems that an access token may be a concatenation of random and static Strings. As illustrated in Figure 3, we assume that an access token may be a concatenation of a random 32 character String (red color), a static six digit sequence of zeros (blue color), a thirteen digit timestamp in milliseconds (green color), a static four digit sequence of zeros (black color), and a random sequence of ten Base64 encoded characters (orange color).

```
b6a6a5efa91d45b8bb91df86673da24c00000014589403796850000Ab6rEoqxUn
ddee7c13b3fd41bea37bd1ef1856452400000014580886920360000kzOh09Jmwl
c4461ab6c06a4bcaa3dd2455c8942c1d000000145928603372700064Pt63C9BJ
b2f0742ca6e2489182e74a73bd302a7f00000014617841824550000D4tubTHa1k
```

Figure 3: Similar patterns in KakaoTalk's OAuth access tokens.

## Forensic Analysis

While examining the mobile handset's file system and physical memory, we found numerous artifacts that may allow an attacker who is able to get a hold of the device, to obtain sensitive information or impersonate a KakaoTalk user. We found that KakaoTalk uses a shared preference file which stores various configuration settings. For instance, the file stores a session identifier which is used to authenticate the KakaoTalk application. In addition, KakaoTalk persistently stores private and public RSA and Ed25519 keys in the `TalkKeyStore.preferences.xml` file in plaintext. With these key-pairs an attacker may be able to decrypt the shared secret value that is being used to compute the E2E encryption key. Moreover, an attacker may be able to forge valid Ed25519 signatures to impersonate a KakaoTalk user. If the mobile phone's file system is not encrypted and has adb debugging enabled, an attacker may be able to obtain a copy of KakaoTalk's files even from a non-rooted device. We also recognized that KakaoTalk uses the `file.delete()` method to delete files from the file system. However, it is well known that this may be an insecure way of deleting files [9].

## Other Issues

- KakaoTalk exports 31 Activities and four Broadcast Receivers with no permissions. These application components may be invoked by other applications to obtain sensitive data or to inject malicious code.
- Even though KakaoTalk encrypts some database entries, it does encrypt its SQLite database as a whole (e.g., by using SQLCipher).

- A four-digit pin that can be used to protect a KakaoTalk installation can be easily brute forced or guessed [10].
- Prior Android 4.1 any application that declares the `READ_LOGS` permission is able to read the log files of any other application. Since KakaoTalk logs a number of events, this may be abused by an attacker to obtain sensitive information.
- KakaoTalk creates a world-readable and executable traceroute binary called “lowell” within its private data directory if the user starts a network test.

# KakaoTalk 4.7.0 “Secret Chat” End-to-end Encryption Protocol Analysis

In the following we list a number of flaws that we have discovered in KakaoTalk's end-to-end encryption messaging system. This analysis was done on KakaoTalk 4.7.0 (Android version)

## KT-01-005 Lack of Message Integrity (Medium)

All IM communications are relayed through the LOCO messaging backend which decrypts packets, creates new ones, and forwards them re-encrypted to the destination. Due to the interception and transformation of messaging packets there is no integrity verification implemented on the client-side. Otherwise, clients would constantly throw an exception that messages have been tampered during network transmission. The only piece of information that is protected by a MAC is the actual chat message. However, KakaoTalk uses the concept of MAC-then-encrypt which means that the ciphertext integrity remains unprotected. Worse, all other data fields in KakaoTalk's E2E encryption protocol do not have any integrity protection mechanisms. The lack of integrity protection would allow a network MITM attacker to manipulate the ciphertext without being detected. And since KakaoTalk's LOCO “secure” or “encrypted” packet uses malleable block cipher modes (e.g., AES-CBC or AES-CFB), “bit-flipping” attacks [1] such as flipping a status code bit to change message flow behaviour, may be possible.

## KT-01-006 Vulnerable to Known Plaintext Attacks (Medium)

The LOCO messaging protocol leaks various known plaintexts which makes it vulnerable to known plaintext attacks (KPA). Most of the data fields of the LOCO messaging protocol contain known and static plaintext entries (e.g., packet length, LOCO command, ID, and other entries). As a result, the LOCO messaging protocol may be vulnerable to a number of plaintext recovery attacks which do not require access to the encryption key.

Moreover, by observing encrypted messaging traffic we found that the first 15 bytes of the first ciphertext block of an encrypted LOCO packet are always static if KakaoTalk is using AES in Cipher Feedback mode. Only the 16th byte of the block is randomized each time a new packet is sent. The fact that the first plaintext block is static and the way how Cipher Feedback mode encryption works [2], means that KakaoTalk most probably increments the Initialization Vector (IV) for each message (see Figure 2).

```
00 00 00 CE E8 F5 6F EF AE 55 A4 38 9 65 A8 10 77 48 F2
00 00 00 CE E8 F5 6F EF AE 55 A4 38 9 65 A8 10 77 48 F2
```

Figure 2: The first 15 most significant bytes of an encrypted LOCO packet remain static (highlighted in red color). The packet structure starting from the most significant byte is as follows: Length (4 bytes) + ID (4 bytes) + Status (1 byte) + Command (11 bytes).

## KT-01-007 Missing Freshness (Info)

While we had not the time to test this in depth, we found evidence that the protocol may not provide freshness to prevent replay attacks (e.g., by adding a random nonce to each message). In a MITM scenario we were able to perform a simple replay attack against KakaoTalk's E2E encryption protocol. We were successful in replaying and reordering packets, as well as replacing ciphertext blocks. The latter chosen ciphertext attack (CCA) worked even though the LOCO messaging server or client auto-corrects this misbehaviour by resending the original message. However, we did not try to mirror or withhold particular messages. A more serious issue may be the possibility that an attacker who stole a KakaoTalk's user mobile handset, may be able to replay previously captured encrypted packets to the device in order to obtain the plaintext.

## KT-01-008 Static Values (Info)

During our source code analysis we recognized a number of static values that are used for message encryption and user authentication. For instance, we found that KakaoTalk uses the hard-coded Initialization Vectors “locoforever” or “KaKAOtalkForever” if the application chooses to use AES-CBC for message encryption. This means that the ciphertext is not randomized and that an attacker may spot plaintexts that result in identical ciphertexts. Another issue that we found was that KakaoTalk uses the static String value “e2dc694aee2540c2de6b4a8be2d7718846a0dfb9” in case it cannot determine the mobile handset's device UUID. This might be an issue since the device UUID is used in the HTTPS authorization header (alongside with an access token) as well as for mapping the user's public keys to an user identity. Finally, we found that KakaoTalk uses the static hard-coded salt value “53656372657443686174526f6f6d4b6579” that is used together with a random shared secret value to compute the E2E encryption key. Since this value can be easily obtained by decompiling the application, an attacker may be able brute-force an E2E encryption key more easily.

### Miscellaneous Weaknesses (Info)

Finally, a number of other weaknesses in KakaoTalk's messaging architecture exist. One weakness may be that Kakao's Web API backend allows unauthenticated calls over HTTP. Even though we were not able to force KakaoTalk to use HTTP instead of HTTPS connections, it does not mean that KakaoTalk may not fallback to use HTTP under certain circumstances that we have not tested (e.g., older Android or KakaoTalk versions). In addition, we recognized that KakaoTalk sometimes transmits the LOCO `CHECKIN` packet in plaintext. A MITM attacker may be able to modify the server reply of this packet in order to point the KakaoTalk client to a malicious IM server. However, we must admit that we were not able to reproduce this behaviour.

## KakaoTalk 5.5.5 Information Privacy Analysis

We found a number of system and design issues that could violate the privacy of users.

### KT-01-009 Information Leakage (Info)

We discovered that KakaoTalk leaks data through unauthenticated channels. First, KakaoTalk sends crash reports to a remote server in case there are issues such as a bad JNI memory access, a Java null pointer exception, or other errors. These reports are sent to the http://group1.magpie.daum.net/magpie/put/ URL unencrypted over HTTP and include sensitive information such as the detailed software and hardware configuration of the user's device as well the error cause, user ID, environment variables and other data. Another source of information leakage is KakaoTalk's network test. This test also sends a report over an unauthenticated channel to a PHP script located at http://nettest.kakao.com/up.php. The network test report includes the user ID, mobile OS and KakaoTalk version as well as the mobile country code. Finally, KakaoTalk may unintentionally leak sensitive information through its usage of various permissions. For instance, the `ACCESS_WIFI_STATE` permission may leak data that may be abused to track individuals [11]. Regarding KakaoTalk's permission usage, we found that KakaoTalk constantly tries to access the user's private phone address book even if the user has disabled the application's feature to sync the address book with the Web API backend. We discovered that KakaoTalk accesses the address book in cases where it does not seem necessary, e.g., while browsing through the application's main UI tabs. Summing up, our found information disclosures may not seem critical, however we argue that even mild privacy leaks may uniquely identify a KakaoTalk user when they are combined together.

### KT-01-010 User Tracking (Info)

First of all, KakaoTalk does not provide anonymous messaging. The application requires the user's phone number in order to sign-up for the service and a private e-mail address in case a user wishes to use additional services. Also, Kakao links the randomly generated device UUID against the user's phone number. In addition, we found evidence that KakaoTalk is tracking the way how a user is using the application. For instance, it tracks the time a user has spent on certain Activities in a database. Moreover, it runs a web cookie database which stores up to six different cookies that are linked to different domains including `tiara.daum.net` and `tiara.kakao.com`. To the detriment of the user's privacy, there is no mechanism to control these cookies as KakaoTalk does not provide a configuration setting to change cookie tracking policies (i.e., configuration options that would allow to accept all cookies, block third-party cookies, or disable all cookies). KakaoTalk also downloads external JavaScript that is most probably used for tracking. For example, scripts including http://m2.daumcdn.net/tiara/js/td.min.js and http://ad1-kant.kakao.com/track_jquery3.js are being downloaded onto the device to log actions such as which news entries the user has tapped on within the *NoticeActivity*.

### KT-01-011 Spam (Info)

In most messengers a new contact needs to be explicitly authorized by the user before the contact is allowed to add the user to her friend list. This feature prevents random persons from adding arbitrary contacts in order to send spam messages. KakaoTalk does not enable this feature by default which means that strangers may be able to add random contacts without requiring their explicit consent. This design may allow a chat bot to automatically add contacts to spread spam messages.

## Limitations of our Study

We did not have the resources to develop Proof-of-Concepts that would show the exploitation of our discovered vulnerabilities in practice. In order to further verify the impact of our detected weaknesses, one would need to perform more testing. Profound testing would include testing our vulnerabilities on the latest version of KakaoTalk (`5.8.3` as of 29.08.2016), on different Android version, as well as on other mobile operating systems such as iOS.

In addition, we used the Android version of KakaoTalk `5.5.5` for most of our analysis work since we started our project in February 2016. We used KakaoTalk `4.7.0` for reverse-engineering the application’s end-to-end encryption protocol since this older version was less obfuscated and therefore faster to analyse than version `5.5.5`. That being said, exploitation might not always be possible as the level of compromise depends on which mobile platform KakaoTalk is running.

## References

[1] https://en.wikipedia.org/wiki/Bit-flipping_attack
[2] https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Feedback_.28CFB.29
[3] https://tools.ietf.org/html/rfc7469

[4] https://www.ssllabs.com/ssltest/
[5] https://www.openssl.org/news/secadv/20160503.txt
[6] https://developer.android.com/reference/android/webkit/WebView.html
[7] http://d3adend.org/blog/?p=314
[8] http://stanford.edu/~pcm2d/blog/jsbridge.html
[9]
https://www.nowsecure.com/resources/secure-mobile-development/coding-practices/understand-secure-deletion-of-data/
[10] http://www.datagenetics.com/blog/september32012
[11] https://hal.inria.fr/hal-00994926