FH MÜNSTER
University of Applied Sciences

Security Audit Report

Threema 2019

Fabian Ising, M.Sc.,
Damian Poddebniak, M.Sc.,
Prof. Dr. Sebastian Schinzel.
Date: 2019-03-28

Table of Contents

Executive Summary

This report describes the results of a thorough independent code audit of the Threema apps for iOS and Android that was performed by the IT Security group at Münster University of Applied Sciences headed by Prof. Dr. Sebastian Schinzel. In the course of the code review, Threema provided full access to the source code of the Android and iOS apps and supported the researchers in regular meetings.

Both Threema apps and the Threema Safe feature successfully passed the assessment and the audit revealed no deviations from the specifications published in the Threema Cryptography Whitepaper. Overall, no high risk or critical vulnerabilities were discovered in Threema Android, iOS and Threema Safe.

Several low to medium risk issues were identified by the examiners and Threema addressed them quickly. They are no longer present in current versions of the apps.

In summary, Threema performs as specified in the published documentation and its security and privacy features are intact and effective.

1. Introduction

“Threema: The messenger that puts security and privacy first.”

Threema is an encrypted messenger produced by the Swiss company Threema GmbH. It offers end-to-end encrypted chats, group chats and calls. Threema’s product line focuses on privacy: neither a phone number nor an email address is required to register or use the apps. Furthermore, it claims that it generates as little data on servers as possible and to delete messages from servers as soon as they were picked up.

Threema GmbH has approached the IT security group at Münster University of Applied Sciences to perform an independent review of the code and architecture of its apps and particularly evaluate the security of its newly implemented Threema Safe cloud backup feature.

2. Context and Scope

The goal of this audit was a thorough third-party examination of the source code of the Threema apps on both iOS and Android for possible issues that could affect the security or the privacy of its users and to check the validity of Threema's public claims stated in the Threema Cryptography Whitepaper[1].

The audit was performed during the period from October 2018 to January 2019 and took 22 person days. Any live testing was done using the production Threema apps downloaded from the Google and Apple app stores.

The scope of the audit was set to Threema Android version 3.54 and iOS version 3.0.9 and the Android implementation of Threema Safe. The Threema server code was out of scope.

Threema provided remote access to an isolated computer within the Threema network that had copies of the Threema Android and iOS apps' source code. The Threema Android source repository also contains the code for Threema Safe on Android.

[1] **https://threema.ch/press-files/cryptography_whitepaper.pdf** (last accessed on 2019-01-23).

3. Findings

The audit revealed no high risk or critical vulnerabilities in Threema Android, iOS and Threema Safe. The appendix of this report lists an excerpt from the test cases that the ITS group applied, but that did not reveal findings. This list and the fact that most findings were merely low risk issues or informational notes is a sign for the overall good quality and high security standard of the analyzed apps. The Threema developers were very responsive and open to the suggestions of the ITS group and mostly provided fixes within hours or a few days. All medium and low risk findings were fixed in Threema Android version 3.62 and Threema iOS version 4.1.

The audit revealed two medium risk vulnerabilities: A malicious Android app could trick the Threema app to send sensitive files such as the Threema private key to another Threema user. However, the victim needs to explicitly choose the receiving Threema user, requiring a component of social engineering to be practically exploited. Furthermore, parts of the Threema Safe password were written to a log file specific to the Threema Android app. An attacker having access to this Threema-internal log could learn parts of the password. While this log is not accessible to other applications, it poses a risk to reveal the password to an attacker.

Most of the findings have low risk rating or are merely informational, leaving a good overall impression of the code. The two identified medium risk vulnerabilities do not pose a directly exploitable risk to users but should be fixed nevertheless.

Other than that, no findings with high or critical risk rating were discovered.

While no audit can prove the complete absence of any vulnerabilities in a software, this audit left the impression that Threema takes the security and privacy of their users very seriously. Furthermore, the audit revealed no deviations from the Threema Cryptography Whitepaper or the claims made by the Threema website.

The findings revealed by this audit are outlined in Appendix A. Appendix B contains an overview of the test cases applied by the IT Security group.

Appendix A: Detailed Findings

Detailed Findings Android App

Threema-2018-android-001 Private Data Leak via Share Intents (Medium)

Threema allows sharing media to Threema contacts from other applications using share intents (*android.intent.action.SEND*). Upon receiving such an intent, the contact list is opened to select a recipient. These intents carry a file URI that defines which file should be sent.

We found that Threema can be tricked into sharing sensitive app data using crafted file URIs, for example the cryptographic private key in the *key.dat* file. This can be done from any malicious app on the smartphone. However, an attacker cannot influence who receives the sensitive data, as the intent opens the sharing dialog where the victim needs to choose the recipient.

Proof-of-concept

A malicious app sends an intent to share the *key.dat* file to the Threema app:

```
Intent sharingIntent = new Intent(android.content.Intent.ACTION_SEND);
sharingIntent.putExtra(Intent.EXTRA_STREAM,
        Uri.parse("file:///data/data/ch.threema.app/key.dat"));
sharingIntent.setType("*/*");
this.startActivity(sharingIntent);
```

This prompts the victim to send this file to a Threema contact. The attacker needs to convince the victim to send the file to the attacker. This can also be triggered by sending the same intent from the adb shell:

```
$ am start -a "android.intent.action.SEND"
                --eu "android.intent.extra.STREAM"
                "file:///data/data/ch.threema.app/files/key.dat"
                -t "/"
```

Recommendation

Threema should filter intent URIs to prevent leaking sensitive data.

Threema-2018-android-002 Leak of GUI elements when PIN lock is active (Low)

Threema allows the user to set an access PIN to access the Threema app. This prevents a malicious user with access to the unlocked Android phone to access Threema. Users need to enter the PIN to access the Threema app.

We found that when using a PIN lock for Threema, the last used activity containing GUI elements is shown as a screenshot under *recent activities*. This may leak sensitive information to an attacker who has physical access to the unlocked smartphone, but who does not know the PIN.

This can be mitigated using the setting to hide thumbnails in recent activities.

Proof-of-concept

Activate the app protection PIN in the Threema app. Then open a chat with sensitive information and close the app. Once the app is locked, open the recent activities menu and scroll to the Threema app. The screenshot with the last opened sensitive chat is shown, even though Threema is locked with a PIN.

Recommendation

We recommend to disable the previews in recent activities in case the PIN lock is activated. Alternatively, the user should be warned when activating the PIN lock that the recent activities may still leak information to attackers not knowing the PIN.

Threema-2018-android-003 Leak of available contacts when PIN lock is active (Informational)

Threema allows the user to set an access PIN to access the Threema app. This prevents a malicious user with access to the unlocked Android phone to access Threema. Users need to enter the PIN to access the Threema app.

We found that when using a PIN lock for Threema, opening *threema://add?id=[id]* for a Threema *[id]* that is already in the contact list shows a toast message “ID already in contacts” which allows an attacker with physical access to an unlocked phone to enumerate the contact list.

Note that if the contact synchronization is enabled, this information would be available to a physical attacker anyway.

Proof-of-concept

Open a Threema URL such as threema://add?id=[id] with an [id] that is in the contact list while the Threema app protection PIN lock is active. When the toast message “ID already in contacts” appears, then this [id] exists in the contact list.

Recommendation

We recommend blocking handling of Threema URIs when the PIN lock is engaged.

Threema-2018-android-004 Messages can be sent via Google Assistant when PIN lock is active (Low)

Threema allows the user to set an access PIN to access the Threema app. This prevents a malicious user with access to the unlocked Android phone to access Threema. Users need to enter the PIN to access the Threema app.

We found that when using a PIN lock for Threema, sending messages using Google Assistant commands is still possible without entering the PIN.

Proof-of-concept

Activate the Threema app protection PIN lock and wait for the PIN lock to engage. Then start Google Assistant and say, “Send a Threema message to [contact]” and dictate a message. The message is sent even though the user did not enter the PIN.

Recommendation

We recommend blocking Google Assistant intents when the PIN lock is engaged.

Threema-2018-android-005 Leak of ICE candidates shortly after a call (Informational)

ICE candidates and therefore IP addresses of the client that participate in calls are buffered during calls for a short time. Even after disconnecting a call previously collected ICE candidates will be sent to the call partner when the timeout for buffering occurs.

Proof-of-concept

Monitor the traffic during calls. Just after the call ends, any existing ICE candidates are still sent.

Recommendation

We recommend clearing the cached ICE candidates on disconnecting a call.

Threema-2018-android-006 Broadcast intent during Threema calls (Low)

When accepting a Threema call, a *broadcast* intent containing the Threema ID and ICE candidates of the call partner is sent. This intent can be intercepted by any installed app on the phone, which leaks information that a Threema call was started and potentially leaking the IP addresses and the Threema IDs of all participants.

Proof-of-concept

Register a broadcast receiver for the intents that the Threema app sends during call initialization.

Recommendation

Remove the broadcast intent and replace it with a local broadcast manager.

Threema-2018-android-007 Vulnerable external library zip4j (Low)

The Threema app uses the external library “zip4j” in version 1.3.2. This version is vulnerable to CVE-2018-1002202 which may allow writing arbitrary files during extraction of an attacker-controlled archive.

However, this CVE is not directly applicable here, because compressed zip archives are never extracted to the file system.

Recommendation

Threema should regularly update external libraries.

Threema-2018-android-008 Vulnerable external library slf4j (Low)

The Threema app uses the external library “slf4j” in version 1.7.24. This version is vulnerable to CVE-2018-8088 which may allow an attacker to execute arbitrary code using a crafted XML serialized string.

However, this CVE is not directly applicable here, because no serialized XML strings are handled.

Recommendation

Threema should regularly update external libraries.

Detailed Findings iOS App

Threema-2018-ios-001 Usage of the deprecated UIWebView (Low)

In some places, the Threema app uses *UIWebView* controls to simply load external websites. One example is the Threema Support view. *UIWebView* controls were deprecated in iOS 8 and should not be used anymore.

Proof-of-concept

The Threema Support is loaded externally.

Recommendation

Replace UIWebViews with WKWebViews or copy content that is now loaded from external sources into the Threema app.

Threema-2018-ios-002 JavaScript and UniversalFileAccess allowed in UIWebView (Low)

JavaScript and *UniversalFileAccess* are activated for UIWebViews.

Recommendation

Replace UIWebViews with WKWebViews.

Threema-2018-ios-003 Unrestricted WebViews (Informational)

WebViews in the license and support are not restricted to specific URLs, e.g. https://threema.ch/, which may allow an attacker to load external pages. However, links in the WebViews cannot be clicked and the user cannot enter URLs.

Recommendation

Replace UIWebViews with WKWebViews.

Threema-2018-ios-004 Missing public key pinning in HTTPS request (Low)

After downloading a blob from the blob servers, a client marks the blob as done so it can be deleted. This specific HTTPS request does not use public key pinning, while all other HTTPS requests do. This could potentially allow a powerful MITM attacker to intercept the request.

Recommendation

Change the request to use public key pinning.

Detailed Findings Threema Safe

Threema-2018-safe-001 Password summary in log files (Medium)

In Threema Safe, when deriving a key from the user provided password using SCrypt, a *summary* of the password is written to logcat logs. This summary effectively allows to reduce the possible choices for each character of the password down to three possible candidates, which greatly reduces the brute-force work for an attacker.

The log cannot be read by other applications, as it is written with verbosity level “verbose”.

Proof-of-concept

```
void log_params(JNIEnv *env, jbyteArray passwd, jbyteArray salt, jint N, jint r, jint p,
jint dkLen) {

	ALOG("Parameters for native scrypt run:");
	ALOG("passwd (summary): %s", get_byte_array_summary(env, passwd));
	ALOG("salt (summary): %s", get_byte_array_summary(env, salt));
	ALOG("N, r, p, dkLen: %d, %d, %d, %d", (int32_t) N, (int32_t) r, (int32_t) p,
(int32_t) dkLen);
}
```

Recommendation

Remove the log output of the password summary.

Threema-2018-safe-002 Custom SCrypt parameters (Informational)

Threema Safe uses SCrypt to derive the encryption key from the user password. SCrypt is used with the parameters $N = 2^{16}, r = 8, p = 1, key_{length} = 64$. However, the recommended value of *N* for encrypting sensitive files is 2^{20}. For reference, see https://www.tarsnap.com/scrypt/scrypt.pdf.

Recommendation

Threema noted that the SCrypt operation has to perform reasonably well on older Android devices. From this viewpoint, we feel that 2^{16} is a reasonable choice. We recommend to regularly reevaluate the chosen SCrypt parameters given the supported mobile devices.

Appendix B: Applied Test Cases

Applied Test Cases Android App

In the following we provide a list of test cases that were applied to the Threema Android app. The list is not necessarily complete, but rather meant to show how we approached the audit.

We loosely based these testcases on the Android developer security guidelines[2], the Android Secure Coding Standards published by Carnegie Mellon University[3], and the OWASP Mobile Security Testing Guide[4]. Additionally, we considered JNI security issues presented by Gang Tan et Al.[5], and JNI tips from the Android developer guidelines[6].

File permissions

For the security of sensitive files, it is necessary to choose the correct file permissions. Threema app files are neither set to world readable (*MODE_WORLD_READABLE*) nor world writable (*MODE_WORLD_WRITABLE*) and are therefore secure from external access.

Encryption of Local Files

To further protect sensitive files from unauthorized access they should be encrypted in storage. All files in Threema's private app directory are encrypted using AES256-CBC-PKCS5. The private key is saved in the *key.dat* file in the private directory and can be secured using a passphrase.

Secure Content Providers

Exported content providers allow access to resources inside an Android application, potentially allowing unauthorized access to sensitive files. Threema uses a single non-exported *FileProvider* that is used to handle app internal file URIs and does therefore not allow access to other apps.

External Storage

Files saved in external storage can be accessed by any app. Threema does not save any sensitive files to external storage without the user's explicit request to do so (e.g. saving an image file to the gallery).

[2] https://developer.android.com/topic/security/ (Last accessed: 2019-02-20)

[3] https://wiki.sei.cmu.edu/confluence/display/android/Android+Secure+Coding+Standard (Last accessed: 2019-02-20)

[4] https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide (Last accessed: 2019-02-20)

[5] http://www.cse.psu.edu/~gxt29/papers/safejni.pdf (Last accessed: 2019-02-20)

[6] https://developer.android.com/training/articles/perf-jni (Last accessed: 2019-02-20)

Permissions

Permissions are a security measure taken by Android to restrict the access to security and privacy relevant features of a smartphone. Therefore, requesting unnecessary permissions or abusing requested permission is a security problem. All permission requested by the Threema app are necessary and, where possible, are requested only when needed (see Table 1).

Table 1: Permissions

Permission	**Description/Usage**
Dangerous Permissions	
CONTACTS	Necessary to the access phone book.
EXTERNAL_STORAGE	Used for sharing and saving shared files as well as saving backups.
ACCESS_LOCATION	Used to send location messages, is requested on demand.
RECORD_AUDIO	Used for Threema Calls and Voice Messages. Requested on demand.
READ_PHONE_STATE	Required to detect incoming regular calls during Threema calls. Requests on demand.
CALL_PHONE	Required to automatically hang up incoming regular calls during Threema calls. Requested on demand.
Normal Permissions	
INTERNET	Needed for connection to servers.
ACCESS_NETWORK_STATE	Needed to check if network connection is available.
READ_SYNC_SETTINGS	Used for setting up automatic phone book sync.
WRITE_SYNC_SETTINGS	Used for setting up automatic phone book sync.
VIBRATE	Used for notifying the user of incoming messages and calls.
WAKE_LOCK	Used for keeping the phone awake during Threema Web Sessions and other long running operations.
RECEIVE_BOOT_COMPLETED	Used to auto start Threema on reboot.
REQUEST_INSTALL_PACKAGE	Used to request installation of updates for the Threema Shop version.
MANAGE_ACCOUNTS	Used to access the address book for syncing.
AUTHENTICATE_ACCOUNTS	Used to access the address book for syncing.
MODIFY_AUDIO_SETTINGS	Used to control volume during calls.
BLUETOOTH	Used to communicate with Bluetooth devices during calls.
CHECK_LICENSE	Used to check Google Play Store License.
INSTALL_SHORTCUT	Used to create contact shortcuts in the launcher.
C2D_MESSAGE	Used to receive GCM push notifications.

Dependencies

Any vulnerabilities present in the dependencies of an app are potentially present and exploitable in the app itself. Therefore, we checked the dependencies of the Threema app using OWASP Dependency Check[7]. Vulnerabilities were found in *zip4j* and *slf4j*, other dependencies were not found to be vulnerable.

[7] **https://www.owasp.org/index.php/OWASP_Dependency_Check** (Last accessed: 2019-02)

Services

Exported services can be accessed by other apps and provide functionality to them. It is therefore necessary to protect exported services from unauthorized access. We found that Threema exports only a small number of services, mainly necessary to synchronize the address book (see Table 2).

Table 2: Services that are explicitly available to other apps or the Android system

Service	Exported	Description/Usage
AutostartService	False	Needed to start Threema on boot.
AccountAuthenticatorService	True	Needed to synchronize the Threema account. An intent-filter is implemented to only allow AccountAuthenticator intents.
ContactsSyncAdapterService	True	Needed to synchronize the address book. An intent-filter is implemented to only allow SyncAdapter intents.
PassphraseService	False	Service for persistent Master Key locked notification.
WidgetService	False	Service to display Threema content in a widget on the home screen. Requires *BIND_REMOTEVIEWS* permission to access.
ConnectivityChangeService	False	Needed to recognize disconnects and reconnects to the internet.
RestrictBackgroundChanged-Service	False	Used to recognize changes background data restrictions.
GcmMessageListenerService	False	Used for push notifications. Intents are filtered to *c2dm.intent.RECEIVE*.
GcmInstanceIDListenerService	False	Used for push notifications. Intents are filtered to *gms.iid.InstanceID*.
GcmInstanceRegistrationIntentService	False	Used for push notifications.
RecipientChooserTargetService	False	Allows choosing the Threema app as a target for intents, e.g. sharing intents. Restricted to *android.service.chooser.ChooserTargetService* intents.

Malicious Intents

Any app that receives intents from other apps must check for malicious behavior to prevent security problems. The Threema app uses intent-filters to filter incoming intents. While an attacker cannot directly trigger malicious behavior via crafted intents, they can use share intents to trick users into leaking sensitive data from the Threema app directory. Registered and exported BroadcastReceivers and their intent-filters are shown in Table 3.

Table 3: Broadcast receivers that are not explicitly marked as non-exported

Name	Intent-Filters	Description
AutostartNotify	BOOT_COMPLETED	Receiver to start the app on reboot.
GcmReceiver	c2dm.intent.RECEIVE,c2dm.intent.REGISTRATION	Receiver for push notifications. Requires the c2dm send permission.
Connectivity-ChangeReceiver	-	Receiver to receive Android connectivity change events.
RestrictBackground-Change-Receiver	-	Receiver to receive events about changes to the background data restrictions.
AlarmManagerBroad-castReceiver	-	Receiver to receive scheduled events.
WidgetProvider	APPWIDGET_UPDATE	Receiver to receive events about app widgets.
UpdateReceiver	MY_PACKAGE_REPLACED	Receiver for app updates.
FetchMessagesBroad-castReceiver	-	Receiver to trigger fetching messages from the Threema server.
VoipMediaButtonReceiver	MEDIA_BUTTON	Receiver to recognize media key actions during Threema Calls.
PowerSaveModeReceiver	POWER_SAVE_MODE_CHANGED	Receiver to recognize power save mode.

Broadcast Intents with Sensitive Information

Intents sent via Broadcast are available to all applications that register a BroadcastReceiver for this intent. Therefore, any sensitive information sent in broadcast intents is available to malicious apps on the smartphone. We found that Threema sends some broadcast intents disclosing potentially sensitive information that can be received by any other apps (see Table 4).

Table 4: List of Broadcast Intents

Name	Action	Description
Backup Complete	MEDIA_SCAN-NER_SCAN_FILES	Tells the Android media scanner about the completed backup file.
Media File Saved	MEDIA_SCAN-NER_SCAN_FILES	Tells the Android media scanner about a file saved to the filesys-tem.
License Check Failed	ch.threema.license_not_allowed	Sent when the user provided li-cense is invalid.
Update Failed	ch.threema.update_available	Sent when an update for the app is available.
Contacts changed	ch.threema.contacts_changed	Sent when a contacts sync is fin-ished.
Audio Device Changes	ch.threema.*	Sent when changes to audio set-tings are made.
ICE Candidates Available	ch.threema.*	Sent when new ICE candidates are available.

Use HTTPS Connections Only

Any data sent over an unencrypted HTTP connection is potentially available to a man-in-the-middle attacker. Therefore, the Android security guidelines recommend using only HTTPS connections. We found that all HTTP connections to the Threema servers are made over HTTPS.

Certificate Handling

To prevent an attacker from tricking an app into accepting a malicious certificate for a HTTPS connection it is recommended to perform Certificate Authority or Public Key pinning. We found that Threema uses CA pinning and that all certificates are validated against the Threema CA.

WebView Settings

When using Android WebViews, developers have to be careful about specific security relevant settings. These include the availability of plugins, JavaScript, and file access. We found that Threema disables plugins for all WebViews, but enables JavaScript for the support view as it is necessary there. Furthermore, local file access is allowed for all WebViews.

Prevent Attacker Controlled WebViews

Any intents that allow sending URIs to WebViews allow an attacker to show malicious content inside an app and potentially allows for UI redressing, coaxing the user into supplying sensitive data to them. However, we found that Threema does not allow intents with WebView URIs.

Logging of Sensitive Data

Logging facilitates debugging and is a useful technique for developers. However, sensitive data must not be logged, as logs are often requested to troubleshoot crashes, etc.

We reviewed logcat calls for sensitive information and found that most logs are connection information. Cryptographic operations are only logged to the validation log if explicitly activated by the user.

However, we found a flaw in the native implementation of SCrypt. A so called "summary" of the user provided password is logged in the verbose log. Other native code does not log any sensitive information.

Restrict Access to Activities

Any activities exported to other apps pose the risk of triggering actions not permitted by the user. Therefore, developers should take care to protect activities using sufficient intent-filters.

Threema protects accessible activities using sufficient intent-filters (see Table 5).

Table 5: Description of Activities that have an associated intent-filter and can be called from other applications. Activity names are simplified.

Name	Intent-Filter	Description
MainActivity	MAIN, LAUNCHER, MULTI_WINDOW_LAUNCHER	Main Threema activity can be called from the launcher.
ComposeMessageActivity	VIEW	Allows opening the chat activity with a specific contact. Takes a contact mime type as input.
RecipientListActivity	SEND, SEND_MULTIPLE	Allows sharing to Threema. Opens the recipients overview. SEND accepts any MIME type to be shared. SENDTO does not allow sharing files directly and is only callable from the contacts app, launcher shortcuts and via URI threema://compose.
ComposeMessageActivity	VIEW	Allows opening the chat activity with a specific contact. Takes a contact mime type as input.
NotificationSettings	MAIN	Allows to open the notification settings from the Android settings.
MediaSettings	MANAGE_NETWORK_USAGE	Allows to open the data usage settings from the Android settings.
AddContactActivity	VIEW	Activity to add new Threema ID to the address book. Callable via URI threema://add.
EnterSerialActivity	VIEW	Activity to activate a Threema license. Callable via URI threema://license.
CallActivity	VIEW	Activity for Threema calls. Takes a Threema contact to call.
CallActionIntentActivity	VIEW	Takes a special mime type to show the call overview.
SMSVerificationLinkActivity	VIEW	Activity to link a mobile number via SMS verification. Callable via URI threema://link mobileno.
VoiceActionActivity	SEND_MESSAGE_TO_CONTACTS	Allows Google Assistant message in text/plain and audio/wav form to be send. This is not restricted by using a PIN lock.

Secure Random Source

Cryptographic algorithms need cryptographically secure randomness in order to generate keys. Low entropy sources may have catastrophic effects on cryptographic algorithms.

Upon app initialization the standard secure random provider is replaced with a provider that reads random bytes from /dev/urandom. This random source is used to generate all random numbers, including cryptographic keys and nonces. For the generation of the long-term private key additional entropy is generated by moving a finger on screen. Insecure Java/Android random providers are not used.

Secure Random

Any time bytes are read from the secure random provider, the call blocks until enough bytes are available.

Secure Parameters (SCrypt)

The SCrypt parameters used (N=2^16 , r=8, p=1, key_length=64) are reasonable even though they do not match the values proposed by the SCrypt paper.

Secure Parameters (PBKDF2)

PBKDF2 is called with the hash function HMAC-SHA1, 8 bytes random salt, and 100000 iterations. These parameters are generally regarded as secure.[8]

Secure Parameters (AES)

AES is always used with random or derivate 256 bit keys (PBKDF2 or SCrypt).

Secure Primitives

Standard Android implementations are used for AES-CBC with PKCS5, PBKDF2-HMAC-SHA1 (to generate an encryption key for the master key in storage), SHA256. PBKDF2-HMAC-SHA256 is implemented in Java code.

The SCrypt implementation is taken from com.lambdaworks by Will Glozer[9]. The database is encrypted using SQLCipher (AES256-CBC, HMAC-SHA1).

Zeroing Memory

Threema does not zero sensitive data in memory. However, zeroing is generally very difficult to do reliably, especially in Java. Thus, it might not be worth the effort.

[8] See for example NISTs recommendation of at least 10000 iterations **https://pages.nist.gov/800-63-3/sp800-63b.html#sec5** (Last accessed: 2019-02).
[9] **https://github.com/wg/scrypt** (Last accessed: 2019-02)

Caching Sensitive Information (Storage)

No data is cached explicitly in storage.

Caching Sensitive Information (Memory)

Some information is cached in memory, including avatars, image thumbnails, contacts, and ICE candidates. However, since these in memory caches should be inaccessible from other apps caching these seems reasonable.

Caching Sensitive Information (Keyboard)

The keyboard cache is deactivated for input fields containing sensitive passwords, e.g. the PIN lock, the Threema Safe password and the master key password. Additionally, users can choose to use an incognito keyboard in the settings to prevent caching sensitive information in conversations.

Caching Sensitive Information (GUI Objects)

Last used GUI elements are cached in memory unless explicitly deactivated in the Settings. This means that in default settings sensitive data might remain visible in Androids recent activities. This leads to confusing behavior when using a PIN lock.

NaCl

Threema uses NaCl's reference implementation[10] when possible, but provides a fallback to JNaCl[11] (a pure Java implementation of NaCl). Furthermore, it uses unmodified NaCl (nacl.cr.yp.to; nacl-20110221). Additionally, they implemented a helper function to work around an implementation detail of NaCl, where the first 32 bytes of the message-to-be-encrypted must be 0x00. We concluded that the helper function works as expected.

In order to call NaCl from Java, Threema wrote an JNI wrapper. We verified the NaCl Java Native Interface for common mistakes[12], and found no obvious issues. Due to the simplicity of the interface, most common mistakes, e.g. passing unchecked `Strings`, forgotten `Release`'s, and Java privilege violations do not apply.

Resource Allocation (JNI)

Resources `Get` via JNI must be `Released` in order to prevent memory leaks. `Get` may return `NULL`, which must be checked to avoid a free/release of NULL.

[10] http://nacl.cr.yp.to (Last accessed: 2019-02)
[11] https://github.com/neilalexander/jnacl (Last accessed: 2019-02)
[12] https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/JNI_OBE/JNI_OBE.html (Last accessed: 2019-02), http://www.cse.psu.edu/~gxt29/papers/safejni.pdf, https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87150750#Rule20.JavaNativeInterface(JNI)-Java (Last accessed: 2019-02), https://www.youtube.com/watch?v=8HgrlqQNnpk (Last accessed: 2019-02)

This issue most often does not apply due to the usage of `GetByteArrayRegion` (as recommended). We reviewed the remaining instances for NaCl manually and found no issues.

We found a potential NULL-dereference and a potential out-of-bounds write in the Scrypt Java Native Interface. We did no further evaluation because Threema decided to delete the erroneous functions completely.

Violating Access Control Rules (JNI)

C code may bypass Java's access controls, e.g. it is possible to call private functions.

Does not apply for Threema.

String Handling (JNI)

Java Strings are different from C Strings. Java Strings are guaranteed to be valid UTF-16 and are not NULL-terminated. Erroneous C code may produce unexpected results when working with Java Strings directly and not using MUTF-8 representation with appropriate filters when working with un-trusted data.

Does not apply to Threema.

Exception Handling (JNI)

Most JNI functions must not be called when an exception is pending. We reviewed the NaCl and Scrypt JNI for problematic code and found only minor issues. Especially, most return values are checked and handled correctly.

Comparison of Object References (JNI)

Object references must only be compared via `IsSameObject()` and not via `==` or `!=`.

Does not apply to Threema.

Applied Test Cases iOS App

In the following we provide a list of test cases that were applied to the Threema iOS app. The list is not necessarily complete, but rather meant to show how we approached the audit.

We loosely based these testcases on the Introduction to Secure Coding Guide by Apple[13], the Secure iOS Application Development guidelines collected by Felix Gröbert[14], and the OWASP Mobile Security Testing Guide[15].

Secure Random Source

Cryptographic algorithms need cryptographically secure randomness in order to generate keys. Low entropy sources may have catastrophic effects on cryptographic algorithms.

Threema reads random data from /dev/urandom. This random source is used to generate all random numbers, including cryptographic keys and nonces. For the generation of the long-term private key additional entropy is generated by moving a finger on screen. Native random providers are not used.

Secure Parameters (PBKDF2)

PBKDF2 is called with the hash function HMAC-SHA256, 8 bytes random salt, and 100000 iterations. These parameters are generally regarded as secure.[16] An exception is made for zip encryption as described below.

Secure Parameters (AES)

AES is used for encrypting conversation backups using MiniZip. It is called with a key derived via PBKDF2-HMAC-SHA1 (iteration count 1000) as defined in the zip encryption standard AE-2[17]. Other AES operations are performed using random 256 Bit keys.

Secure Primitives

For encrypting zipped conversation backups, the AES and PBKDF2 implementation of Brian Gladman is used. For identity backups the PBKDF2 Apple's CommonCrypto implementation is used. For SHA1, SHA256 and other AES operations Apple's CommonCrypto implementation is used. For NaCl the reference implementation by Daniel J. Bernstein is used.

[13] **https://developer.apple.com/library/archive/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html** (Last accessed: 2019-02)

[14] **https://github.com/felixgr/secure-ios-app-dev** (Last accessed: 2019-02)

[15] **https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide** (Last accessed: 2019-02)

[16] See for example NISTs recommendation of at least 10000 iterations **https://pages.nist.gov/800-63-3/ sp800-63b.html#sec5** (Last accessed: 2019-02).

[17] **http://www.winzip.com/aes info.htm** (Last accessed: 2019-02)

Zeroing Memory

Threema does not zero sensitive data in memory. However, zeroing is generally very difficult to do reliably especially, when keys are accessed repeatedly. Thus, we feel it might not be worth the effort.

Local Authentication

Threema implements Touch ID to prevent unauthorized access to the app. Additional protections, for example against Jailbreaks are not implemented.

Secure Deserialization

Whenever objects are deserialized in iOS apps, developers should be careful about potentially malicious data that could be supplied by an attacker. Threema deserializes all objects using *NSJSONSerialization*, which is generally regarded secure.

Secure Handling of SQL Statements

Handwritten SQL statements manipulating an app internal database that handle user supplied input are potentially vulnerable to SQL injections. We found that Threema does not use SQL directly but all Database access is done via *NSPersistentStoreCoordinator*, which is not vulnerable to SQL injections.

Compile-Time Options

As iOS apps are often written in Objective-C, they are potentially vulnerable to issues like Memory Corruptions and Buffer Overflows. Various compile-time options are available as countermeasures to make attacking these vulnerabilities harder. We found that Threema enables stack smashing protection and position independent executables. Automatic reference counting is enabled for the Threema code, but disabled for some dependencies where necessary.

Code Signing

To prevent malicious clones of an iOS app reaching the Apple App Store, an app should be signed with a secure developer certificate. We found that Threema signs their release build using an iPhone developer certificate.

Anti-Reversing Techniques

To prevent attackers tampering with an iOS app it is recommended to take some anti-reversing measurements, for example stripping debug symbols and using a Jailbreak and debugging detection. We found that Threema strips all debug symbols from release builds, but does not implement a Jailbreak or debugging detection. We, however, feel this is reasonable as the checks are usually not a real help against dedicated attackers.

App Transport Security

Any data sent over an unencrypted HTTP connection is potentially available to a man-in-the-middle attacker. Therefore, iOS apps by default cannot make unencrypted HTTP connections to webservers without changing the settings of App Transport Security (ATS). We found that Threema uses ATS in default configuration, therefore not allowing unprotected HTTP connections.

Secure HTTP Settings

iOS apps can set up HTTPS connections with debug options resulting in potentially sensitive information like cryptographic keys to be logged. Additionally, parameters passed in HTTPS URIs might be available at the proxy level. We found that Threema neither has debug options for HTTPS enabled nor do they send sensitive data in HTTPS URIs.

Certificate Handling

To prevent an attacker from tricking an app into accepting a malicious certificate for a HTTPS connection it is recommended to perform Certificate Authority or Public Key pinning. We found that Threema checks all certificates using TrustKit, which is configured for public key pinning. These checks are applied for all HTTPS connections, except for the request to mark an encrypted media blob.

Verification of app URI calls

iOS apps can export URIs that can be called from other apps to trigger actions in the app. Developers should take care that they do not automatically perform malicious actions upon URI calls. We found that in Threema all actions that can be triggered via URLs are verified or the user is asked for permission. See Table 6 for all available URIs.

Table 6: Threema URIs

URL	Description/Usage
threema://restore	Allows restoring an identity backup, only allowed during app setup.
threema://add	Allows adding a new contact. Asks the user for permission. Can be deactivated in the settings.
threema://compose	Allows opening the compose dialog with a contact. Also allows sharing images via pasteboard.
threema://link_mobileno	Links a Threema identity with a mobile number using a code from the verification SMS.
Threema://license	Allows activating a Threema work license using a username and a password.
file://	Allows sharing a file via Threema.

Verification of outgoing URLs in WebViews

WebViews in iOS apps should be restricted to a subset of URLs to prevent displaying malicious content inside the app. We found that WebViews in Threema are not restricted to specific URLs, but don't allow clicking links or browsing the web. However, onclick handlers and redirects are still possible.

Prevent XSS in WebViews

Whenever a WebView displays attacker controlled data cross-site scripting vulnerabilities are possible. To prevent this, developers should restrict JavaScript usage and file access on WebViews. We found that Threema WebViews do neither.

Avoid HTML Previews

To prevent the executing of malicious local or remote content in iOS apps it is recommended to avoid HTML previews for content. We found that Threema does not support HTML previews.

Disable Autocorrect for Sensitive Data Fields

To prevent sensitive data (e.g. passwords) from leaking while being supplied by the user, auto correct should be disabled on sensitive input fields. We found that in Threema autocorrect is disabled for all sensitive input fields, including password, ID, E-mail and the license username fields.

Secure Pasteboard Handling

Any data shared using pasteboards is potentially available to other apps. Developers should, therefore, be careful about supplying sensitive information in pasteboards. We found that Threema uses the global pasteboard only to share information with other apps and does not implement its own pasteboard.

Secure Keychain Permissions

Secrets saved in the iOS keychain should use the minimum possible permission to prevent abuse. We found that the Threema identity key is only accessible on the device it was created on after it was first unlocked. An identity backup is available whenever the keychain is unlocked.

Secure File Storage

Files stored on the storage might potentially be available to an attacker if they are not sufficiently secured. We found that all files are sufficiently protected as Threema uses the *NSFileProtectionCompleteUntilFirstUserAuthentication* permission.

Secure Temporary Files

Temporary files should only be created in the secure temporary *NSTemporaryDirectory* to secure them from attackers. We found that Threema uses this directory for temporary files.

Secure Backups

App data saved in backups might potentially fall in the hands of an attacker and should therefore be sufficiently secured. The Threema database can be excluded from backups. Keychain items are not synchronizable and therefore are not available in iCloud backups. They can however be restored using an iTunes backup although they are only decryptable on the same device. User settings are included in a backup.

Memory Leaks

As iOS apps are often written in Objective-C, they are potentially vulnerable to issues like Memory Leaks. Therefore, developers should take care to free allocated memory upon deallocation. We found that allocated buffers (mainly for random bytes) in Threema are handled by *NSData* and are freed upon deallocation. Also allocated buffers are freed upon exceptions.

Memory Corruption Format Strings

As iOS apps are often written in Objective-C, they are potentially vulnerable to issues like Memory Corruption via format strings. However, we found no vulnerable format strings in Threema.

Dependencies

Any vulnerabilities present in the dependencies of an app are potentially present and exploitable in the app itself. Therefore, we checked the dependencies of the Threema app manually for vulnerabilities. None of the used dependencies were found vulnerable.