Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Pentest- & Audit-Report Threema Mobile Apps 10.2020

Cure53, Dr.-Ing. M. Heiderich, M. Wege, BSc. C. Kean & other team members

Index

- Scope
- Test Methodology
  - WP1: White-box tests against Threema mobile app for Android
  - WP2: White-box tests against Threema mobile app for iOS
  - WP3: White-box tests and reviews against NaCl & general crypto integration
- Miscellaneous Issues
  - 3MA-01-001 WP2: Enabled NSURLRequest leads to client-side request caching (Info)
  - 3MA-01-002 WP2: Lack of restricted segments in dylib code injection (Info)
  - 3MA-01-003 WP2: Incomplete iOS filesystem protections (Info)
  - 3MA-01-004 WP1: Outdated android-gif-drawable software dependency (Info)
  - 3MA-01-005 WP1: Lack of FORTIFY_SOURCE for third-party shared objects (Low)
  - 3MA-01-006 WP1: Info-disclosure in auto-generated screenshots (Medium)
  - 3MA-01-007 WP1: PIN code comparison not timing-safe (Low)
  - 3MA-01-008 WP1: Improvements for security settings (Info)
- Conclusions

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Introduction

"The messenger that puts security and privacy first. Pay once, chat forever. No ads. No collection of user data."

From https://threema.ch/en

This report describes the project focused on the security of the Threema mobile applications for iOS and Android. Carried out by Cure53 in October 2020, the project entailed a source code audit and a broader assessment of Threema, which is a mobile messenger platform standing out due to offering stringent privacy and security guarantees. The objective of Threema is to furnish its apps as open source in the near future[1].

The work was requested by the Threema team and then executed by Cure53 in October 2020, precisely in CW42. Due to the nature of the project described above, this audit was requested to make sure that an independent third-party, in this case Cure53, has a look at the code before the project becomes widely available.

As for the resources, four members of the Cure53 team were tasked with this project on the basis of skills best-matching the objectives communicated by Threema. The auditors planned, executed and finalized the work over the course of sixteen person-days, budgeted in line with dedication to getting good coverage and necessary depth of research.

For better structuring of the work against the main features and areas, the project was split into three different work packages (WPs). These were:

- **WP1**: White-box tests against Threema mobile app for Android
- **WP2**: White-box tests against Threema mobile app for iOS
- **WP3**: White-box tests and reviews targeting Sodium and general cryptographic integration.

It can be derived that white-box methods have been used in this project. Cure53 could access all relevant source codes and was also supplied with binaries to run the app on their test devices. All material that was needed to get the expected coverage level was shared with Cure53, which helped the overall efficiency. The tested project itself was used for communications. The Threema team created a dedicated group conversation and invited relevant personnel from Cure53 to join in and participate. Exchanges and feedback were done with a high degree of efficacy and professionalism, again assisting in the project being able to progress at a good pace.

[1] https://threema.ch/en/blog/posts/open-source-and-new-partner

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

In terms of findings, eight issues were spotted and documented. Importantly, none of the noticed flaws could be seen as actual security vulnerabilities. Instead, they belong to a broad category of general weaknesses, characterized by limited exploitation potential. Corresponding to this, the findings received severity scores largely situated in the *Informational* and *Low* range. Foreshadowing conclusions, it can be stated that this is a great result for the tested mobile application compound.

In the following sections, the report will first shed light on the scope, key test parameters and shared material. That section will be succeeded by a test methodology chapter, in which Cure53 will elaborate on what was tested and how - even if no findings were spotted in the given area. The clear goal here is to guarantee better levels of transparency to the readers, especially given the purpose of the audit and the likely public exposure and scrutiny of this work. Next, all findings will be discussed in a chronological order. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this autumn 2020 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team's observations and the collected evidence. Tailored hardening recommendations for the Threema mobile applications are also incorporated into the final section.

***Note**: This report was updated in late November 2020 after Cure53 was able to successfully perform a fix verification process in collaboration with the Threema team.*

Each issue ticket has been updated with a note to clarify on the status of the respective fix or mitigation. Fixes have been verified based on diffs and detailed descriptions.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Scope

- **Penetration-Tests & Code Audits against Threema Mobile Apps for Android & iOS**
  - **WP1**: White-box tests against Threema mobile app for Android
    - Sources were shared
  - **WP2**: White-box tests against Threema mobile app for iOS
    - Sources were shared
  - **WP3**: White-box tests and reviews of Sodium and general crypto integration
    - *See above*

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Test Methodology

The following section documents the testing methodology applied during this engagement and sheds light on various areas of the mobile application subject to inspection and audit. It further clarifies which areas were examined by Cure53 but did not yield any findings.

WP1: White-box tests against Threema mobile app for Android

The information below describes the tests and coverage achieved for the Android security-related testing of the given Threema scope. The section comments on which areas were investigated through the enumerated approaches.

- The local storage of the Threema Android application was examined via *adb shell* on a Magisk rooted device on Android 9.0 and two x86 emulated Android devices.
- As Android employs sandboxing using SELinux to prevent apps from accessing local storage and data of other users, it can be assumed that the Threema local storage is secure when it comes to third-party app access. However, this countermeasure might vanish on a rooted device. Nevertheless, it has to be noted that the app does not ask for any personal information or credentials, which further reduces the potential attack surface.
- The Android device's logcat output was examined for sensitive information leaks from the app. However, no such leaks were spotted during or following the usage of this application.
- The Android app's network communications were also reviewed by intercepting the connection. It was found that plain-text HTTP communications are not in use. The team also attempted to intercept TLS traffic with invalid certificates, which the application correctly rejected.
- Threema's remotely reachable attack surface within the Android branch has been reviewed; explicit focus was placed on identifying directory traversal bugs potentially triggerable from remote. This would be done by sending specially crafted files, as well as logic bugs within the VoIP stack of Threema.
- The local attack surface has been reviewed from the perspective of a malicious application running on a victims' device. In particular, exposed services and activities of the Threema application have been investigated and Cure53 attempted to interact with them in order to cause violations of security properties.
- It was positively noted that the Threema for Android client is sufficiently hardened and no severe vulnerabilities could be identified within the given time frame. The identified weaknesses should be seen as recommendations, which would enhance the security of the Threema application through improving external third-party applications with compiler flags (3MA-01-005) and keeping

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

dependencies up-to-date (3MA-01-004). Disabling the automatic screenshot capturing feature when the app is backgrounded (3MA-01-006) also belongs to this category of recommendations as it could potentially leak sensitive information to the filesystem if the Threema application is backgrounded, e.g., when the Master Key Passphrase or the PIN code to unlock Threema are being set.

WP2: White-box tests against Threema mobile app for iOS

A list of items below seeks to detail the tasks completed during the iOS-centered portion of the security testing phase of this project. This is to underline what the Cure53 testers covered during their analysis, particularly in regard to iOS security within Threema.

- The local storage of the Threema iOS application was examined via SSH connection[2] on a jailbroken device; version iOS 13.3.1 with the *checkra1n* exploit[3] was used.
- The iOS branch of Threema was found to exhibit some potential to be further improved by restricting filesystem permissions (3MA-01-003) and completely disabling client-side caching (3MA-01-001).
- It was positively noted that the iOS app takes advantage of the most common compiler and linker flags such as PIE, ARC or the Stack Canary. This could be further enhanced by enabling restricted segments (3MA-01-002) which would harden the app against DyLib code injection.
- As iOS employs sandboxing to prevent apps from accessing other users' local storage, it can be assumed that the Threema local storage is secure when it comes to third-party app access. However, this countermeasure might vanish in a jailbroken or similarly altered iDevice. In terms of strengths, the app does not ask for any personal information or credentials, which further reduces the potential attack surface.
- The iOS app's network communications were reviewed by intercepting the connection. It was found that plain-text HTTP communications are not in use. The team also attempted to intercept TLS traffic with invalid certificates, which the application correctly rejected. Furthermore, the Threema iOS app was found to have App Transport Security *(*ATS*)*[4] enabled and does not define ATS exceptions as gateways to insecure connections.
- The iOS device logs were examined for sensitive information leaks from the app. However, no such problems arose.

[2] https://cydia.saurik.com/package/openssh/
[3] https://checkra.in/
[4] https://developer.apple.com/documentation/bundleresources/information_proper...pptransportsecurity

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

WP3: White-box tests and reviews against NaCl & general crypto integration

A list of items below seeks to detail the tasks around the cryptographic integration testing phase of this project. This is to underline what the Cure53 testers covered during their analysis, especially in terms of NaCl and the general cryptographic integration within Threema.

- Threema for Android provides a JNI wrapper for interacting with the native crypto-related code from Java and employs NaCl for end-to-end encryption, as well as to secure the chat protocol at the transport level. The integration of NaCl in Threema for Android was reviewed through source code as well as by attempting to intercept network communication (MitM attacks) and carry out attacks pursuant to information linked to the local storage. As part of this review, no code flaws or possibilities of obtaining plain-text messages from encrypted blobs were identified.
- It has to be noted that Threema for Android persists sensitive information inside an encrypted database and the AES256 master key, used for encrypting and decrypting locally stored application data, is persisted on the device inside a file called *key.dat*. The private key is also stored on the device in an encrypted form, encrypted with the AES256 master key. It was pointed out that not setting an additional passphrase allows a local attacker, who has highly privileged access on the victim's device, to read the *key.dat* file, remove the applied obfuscation, and finally obtain the deobfuscated master key. From there, one could decipher the *threema.db* file and private keys of users. Assuming that a user has set the aforementioned passphrase, no circumvention or leakage could be identified.
- The iOS branch of Threema employs NaCl for end-to-end encryption and to secure the chat protocol at the transport level. The integration of NaCl in Threema was reviewed both in the code and in relation to MitM attacks and approaches based on information from the local storage. However, no code flaws or possibilities of reversing encrypted blobs with assumed-broken TLS were identified.
- The local data is protected by a Core Data database with encryption enabled. This is achieved by setting the option *completeUntilFirstUserAuthentication*. The private key is stored in the iOS Keychain with the option *kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly*. The functionality of the local data encryption and key storage were verified and no circumvention or leakage could be documented.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

3MA-01-001 WP2: Enabled *NSURLRequest* leads to client-side request caching *(Info)*

***Note**: This issue has not yet been addressed but is flagged as a work-in-progress. The fix is planned to be rolled out in Threema release 4.6.4 for iOS.*

It was discovered that the *NSURLCache* is enabled for some API communications of the iOS app. This could accidentally expose API communications containing sensitive data such as user-credentials, *authentication* tokens or PII. The impact of this issue was evaluated as *Info* since no sensitive information is cached in the current iOS Threema build.

Affected Files:

- *Library/Caches/ch.threema.iapp/Cache.db*
- *Library/Caches/ch.threema.iapp/Cache.db-wal*
- *Library/Caches/ch.threema.iapp/Cache.db-shm*

Client-side caching should be disabled to prevent the automatic recording of API communications in the cache. The *Secure Mobile Development* guide[5] can be reviewed for further instructions regarding the aforementioned cache getting disabled. It should be noted that the default *NSURLCache* does not support altering the protection level of its store, as advised in 3MA-01-003.

Consequently, this means that all requests and responses will still be cached and left unprotected at rest via the *NSURLCache,* even when an application implements Data Protection at the application level. If the *URLCache* is required, this can be avoided with a custom *NSURLCache* subclass, thus storing responses on an SQLite *DB* file with the *NSFileProtectionComplete* attribute set.

[5] https://github.com/nowsecure/secure-mobile-development/blob/master/en/io...-requests-responses.md

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3MA-01-002 WP2: Lack of restricted segments in *dylib* code injection *(Info)*

***Note**: This issue has been addressed successfully by the Threema team. The fix is planned to be rolled out in Threema release 4.6.4 for iOS.*

While reviewing the Threema binary on iOS, it was noted that it lacks a *__restrict* segment to ignore *Dynamic Loader* (*dyld)* environment variables which could facilitate code injection. The impact of this issue was evaluated as *Info* since no code injection could be achieved in the limited time frame available for this engagement. It is likely that this kind of code injection is only feasible in a jailbroken environment or on iOS below version 10. However, the latter is not supported by the Threema build in scope. The absence of the *__restrict* segment can be verified on MacOS with the *size* command. The following command has to be run on the binary contained in the extracted IPA archive of the application.

Command:
```
size -x -l -m Threema.app/Threema | grep -w __RESTRICT -A 5
```

In order to flag a binary as restricted, one has to configure the *linker* in XCode by adding the following flags into the *Other Linker Flags* section[6] located in *Select Project in file navigator sidebar→ Build Settings → Linking → Other Linker Flags*.

Compiler Flags:
```
-Wl,-sectcreate,__RESTRICT,__restrict,/dev/null
```

The described measure is based on the documentation in the *Dynamic Loader* (*dyld)* source code[7] contained in the Apple Open Source Library:

> *“Look for a special segment in the mach header. Its presence means that the binary wants to have DYLD ignore DYLD_ environment variables.”*

It is recommended to consider whether the described countermeasure is required in the security model of the Threema iOS app. The official documentation for this type of exploitation, as well as its countermeasure, is scarce for iOS and largely based on analogous code injections exploits on MacOS or app modifications on jailbroken iOS devices. Therefore, any steps to counter this on iOS can only be seen as an optional hardening measure.

[6] https://theevilbit.github.io/posts/dyld_insert_libraries_dylib_injection_in_macos_osx_deep_dive/
[7] https://www.opensource.apple.com/source/dyld/dyld-210.2.3/src/dyld.cpp

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3MA-01-003 WP2: Incomplete iOS filesystem protections *(Info)*

***Note**: No fix is planned to be created here, the issue was mutually accepted by Threema and Cure53 as a false alert, as the described behavior is a technical requirement for handling notifications.*

It was found that the iOS app does not take full advantage of the native iOS filesystem protections and fails to fully protect some of its data files at rest. The affected files are only protected until the user authenticates for the first time after booting the phone. The problem is that the key to decrypt these files will remain readable in memory even while the device is locked. The impact of this issue was evaluated as *Info* because no sensitive information is exposed.

This issue requires physical access to an iDevice set to a locked screen and a method of accessing the local storage, for instance, through an SSH connection established via a jailbreak. While being locked, the files represented below remain protected whenever they are not flagged with "*Operation not permitted*".

Command:

```
tar cvfz files_locked.tar.gz *
```

Output:

```
Library/Caches/ch.threema.iapp/Cache.db
Library/Caches/ch.threema.iapp/Cache.db-wal
Library/Caches/ch.threema.iapp/Cache.db-shm
Library/Caches/spki-hash.cache
Library/Caches/SentryCrash/Threema/Data/CrashState.json
Library/Caches/SentryCrash/Threema/Data/ConsoleLog.txt
Library/Saved Application
State/ch.threema.iapp.savedState/KnownSceneSessions/data.data
tar: Library/SplashBoard/Snapshots/sceneID\:ch.threema.iapp-default/FBDBAFC7-
3849-497C-B3D3-953E77FFA733@2x.ktx: Cannot open: Operation not permitted
tar: Library/SplashBoard/Snapshots/sceneID\:ch.threema.iapp-default/597F1F68-
9C19-48D2-ADD7-90F664505513@2x.ktx: Cannot open: Operation not permitted
tar: Library/SplashBoard/Snapshots/sceneID\:ch.threema.iapp-default/downscaled/
AF8382AE-941C-4194-B31E-C5272E450B31@2x.ktx: Cannot open: Operation not
permitted
tar: Library/SplashBoard/Snapshots/sceneID\:ch.threema.iapp-default/downscaled/
75F92B09-3B68-46DF-9D45-AA1A506EDCDB@2x.ktx: Cannot open: Operation not
permitted
Library/SplashBoard/Snapshots/ch.threema.iapp - {DEFAULT GROUP}/0B81E617-3C8F-
4748-9C35-EB1953E107A0@2x.ktx
Library/SplashBoard/Snapshots/ch.threema.iapp - {DEFAULT GROUP}/E422F160-465A-
4B87-9129-95ECE8AC9862@2x.ktx
Library/SplashBoard/Snapshots/ch.threema.iapp - {DEFAULT GROUP}/84994B6B-26CB-
4E72-85A5-A041116508B6@2x.ktx
```

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
Library/SplashBoard/Snapshots/ch.threema.iapp - {DEFAULT GROUP}/AD82C750-A81B-
4363-9910-2ABCE86CFDD4@2x.ktx
```

In order to solve the issues related to file access, it is recommended to implement the *NSFileProtection-Complete* entitlement at the application level[8] for all files, as well as considering changes to the *NSUrlCache* described in 3MA-01-001.

3MA-01-004 WP1: Outdated *android-gif-drawable* software dependency *(Info)*

***Note**: This issue has been addressed successfully by the Threema team. The fix was rolled out in Threema release 4.43 for Android (in beta).*

While reviewing third-party software components used by the Threema for Android application, it was identified that the *android-gif-drawable*[9] library integrated into Threema is outdated.

android-gif-drawable provides functionality to render animated *GIF*s on Android devices. The software dependency offers a thin Java layer to interact with the native code responsible for rendering *GIF* files. *android-gif-drawable* has gotten some public attention in the past as *Critical* vulnerabilities were identified[10]. The used *android-gif-drawable* version v1.2.19 dates back to October 2019 and the latest available software version is v1.2.21, which was released on 2020-10-14. Even though the referred third-party software component is not vulnerable against any publicly known vulnerabilities, the version remains outdated and more than a year old. Besides this, one potentially interesting looking commit, supposed to fix a Heap-based buffer overflow in v1.2.19, has been reverted in v1.2.20, as described online[11].

It is recommended to keep software versions up-to-date, as older releases potentially contain known (and unknown to the public) vulnerabilities that malicious users might try to exploit.

[8] https://developer.apple.com/library/ios/documentation/iP...App/StrategiesforImplementingYourApp.html
[9] https://github.com/koral--/android-gif-drawable
[10] https://awakened1712.github.io/hacking/hacking-whatsapp-gif-rce/
[11] https://github.com/koral--/android-gif-drawable/pull/714

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3MA-01-005 WP1: Lack of *FORTIFY_SOURCE* for third-party shared objects *(Low)*

***Note**: This issue has not yet been addressed but is flagged as a work-in-progress. The fix is planned to be rolled out in future Threema versions.*

While analyzing the official Threema for Android application acquired from the Play Store, it was noticed that some of the included shared objects are not compiled using the *FORTIFY_SOURCE*[12] compiler option. The *FORTIFY_SOURCE* macro provides basic support for detecting buffer overflows within various functions that perform memory and string operations, including the following list of functions: *memcpy, mempcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat, sprintf, vsprintf, snprintf, vsnprintf* and *gets.*

Listed below are the libraries compiled without using *FORTIFY_SOURCE*:

- *libnacl-jni.so*
- *libmapbox-gl.so*
- *libsqlcipher.so*
- *libpl_droidsonroids_gif.so*
- *libscrypt.so*

PoC:

The lack of fortified functions can be determined by running the following command against the extracted shared object files:

```
$ readelf -s libpl_droidsonroids_gif.so  | grep "_chk"
   28: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
$ readelf -s libsqlcipher.so  | grep "_chk"
  132: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
$ readelf -s libscrypt.so  | grep "_chk"
   12: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
$ readelf -s libnacl-jni.so  | grep "_chk"
    15: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
$ readelf -s libmapbox-gl.so  | grep "_chk"
    66: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
```

Below it is shown how a shared object with fortified functions would look like:

```
$ readelf -s libjingle_peerconnection_so.so  | grep "_chk"
    3: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __stack_chk_fail@LIBC (2)
   32: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __memcpy_chk@LIBC (2)
  104: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __vsnprintf_chk@LIBC (2)
```

[12] https://man7.org/linux/man-pages/man7/feature_test_macros.7.html

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
111: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __vsprintf_chk@LIBC (2)
119: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __FD_CLR_chk@LIBC (2)
120: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __FD_ISSET_chk@LIBC (2)
121: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __FD_SET_chk@LIBC (2)
122: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __read_chk@LIBC (2)
172: 0000000000000000  0 FUNC  GLOBAL DEFAULT  UND __strchr_chk@LIBC (2)
```

Cure53 recommends to consider compiling the referred shared objects with FORTIFY_SOURCE enabled. This should be accomplished by using the compiler option *-D_FORTIFY_SOURCE=2*[13].

3MA-01-006 WP1: Info-disclosure in auto-generated screenshots *(Medium)*

***Note**: The core issue behind this finding has been addressed successfully by the Threema team. The fix has been rolled out in Threema release 4.43 for Android (in beta). Cure53's suggestion of converting the modal "password change" dialogues to separate activities are under consideration. Furthermore an option for disabling screenshots for the entire application is present.*

To provide users of modern mobile applications with an aesthetically pleasing experience at the time when the application starts up or exits, many manufacturers introduce the screenshot-saving feature, which operates whenever the application is sent to the background. This feature potentially poses a security risk, since sensitive data may be exposed if the user deliberately "screenshots" the application by pressing the device's home button.

In this context, the app may be sent to the background while sensitive data is displayed, e.g., when setting the Master Key Passphrase, PIN code for unlocking the Threema UI or setting password when creating a backup using "*Threema safe*".

The auto-generated screenshots are written to the device's local storage inside the directory */data/system_ce/0/snapshots* and remain persistent as long as the app has not been closed completely. As a result, there is a potential risk that a malicious app is running on the victim's device, while doing so collecting and recovering screenshots stored on the device while the Threema app is sent to the background.

It is important to emphasize that Threema allows users to *opt-in* and have the *"App protection"* enabled, which would prevent the generation of screenshots when the app is sent to the background. Nevertheless, it is Cure53's opinion that important activities, such as those related to entering the Master Key passphrase, PIN code for unlocking the

[13] https://github.com/hashbang/hardening#source-fortification

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

Threema UI or using a password upon backup creation, should have the auto-generation of screenshots disabled by default.

PoC:

1. Open the Threema for Android mobile application and navigate to *My Profile* > *Settings* > *Security* > *Passphrase* and set a passphrase. Next, click on the *"eye"* icon next to the entered password in order to double check that the entered password is correct.
2. Press the device's home button to send the app to the background.
3. Connect to the Android mobile test device using the *adb* utility.
4. Navigate to the following directory on the device and verify that a screenshot is stored therein, as verified below.

```
sargo:/data/system_ce/0/snapshots # ls -la
total 34
drwx------ 2 system system  3488 2020-10-19 14:06 .
drwxrwx--- 6 system system  3488 2020-10-19 12:50 ..
-rw------- 1 system system 15016 2020-10-19 14:06 22.jpg
-rw------- 1 system system    12 2020-10-19 14:06 22.proto
-rw------- 1 system system  4408 2020-10-19 14:06 22_reduced.jpg
```

5. Copy the generated screenshot from the device to the local machine.
6. An example screenshot, automatically captured and stored unencrypted on the device, shows the entered Master Key Passphrase. It is important to emphasize that similar steps are possible when setting a password during backup creation and when one is enabling a PIN or pattern for locking the UI.

It is recommended to prevent the Android app from generating screenshots containing sensitive information on the device's local storage when the app is "backgrounded". This must absolutely take effect by default for security-critical activities such as entering the Master Key passphrase, using PIN code for unlocking the Threema UI or offering a password when creating a backup. The following online resources provide noes on remedying this on Android[14] mobile applications.

[14] https://mobile-security.gitbook.io/mobile-security-testing-g...o-generated-screenshots-mstg-storage-9

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

3MA-01-007 WP1: PIN code comparison not timing-safe *(Low)*

***Note**: This issue has been addressed successfully by the Threema team. The fix is planned to be rolled out in Threema release 4.44 for Android.*

During the code audit phase, it was noticed that the comparison of the user PIN code fails to utilize a timing-safe comparison construct[15]. In effect, an attacker could brute-force PIN code in use, effectively bypassing the protections they are meant to offer. The primary authorization function, however, utilizes a brute-force protection through throttling, thus lowering the severity of this issue considerably.

Affected Files:
app/src/main/java/ch/threema/app/activities/PinLockActivity.java

Affected Code:

```
public class PinLockActivity extends ThreemaActivity {
	[...]
	private String pinPreset;
	[...]

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);

		logger.debug("onCreate");
		[...]
		pinPreset = getIntent().
			getStringExtra(ThreemaApplication.INTENT_DATA_PIN);
		[...]
	}

	private void handleNext() {
		final String pin = passwordEntry.getText().toString();
		if (lockAppService.unlock(pin) || pin.equals(pinPreset)) {
			EditTextUtil.hideSoftKeyboard(passwordEntry);

			setResult(RESULT_OK);
			finish();
		} else {
			if (isCheckOnly) {
				passwordEntry.setEnabled(false);

				handler.postDelayed(() ->
			RuntimeUtil.runOnUiThread(this::finish), 1000);
			}
```

[15] https://codahale.com/a-lesson-in-timing-attacks/

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

```
                        if (++numWrongConfirmAttempts >=
                                FAILED_ATTEMPTS_BEFORE_TIMEOUT) {
                                long deadline =
                        setLockoutAttemptDeadline(DEFAULT_LOCKOUT_TIMEOUT);
                        // TODO default value
                                handleAttemptLockout(deadline);
                        } else {
                                showError(R.string.pinentry_wrong_pin);
                        }
                }
        }
        [...]
}
```

It is recommended for all security-critical comparisons to utilize a timing-safe comparison function which provides a constant runtime when comparing two strings.

3MA-01-008 WP1: Improvements for security settings *(Info)*

***Note**: This issue has not yet been addressed but is flagged as a work-in-progress. The fix is planned to be rolled out in Threema release 4.44 for Android.*

While reviewing the Threema application on Android, it was observed that the mobile application lets users protect the app from unauthorized access through several configuration settings:

- Access Protection: Enabling this feature allows users to set PIN/Pattern/Biometric credentials. This layer of protection acts as a pure UI protection, e.g. for private chats or when the user is opening the application.
- Encryption of locally stored data: Enabling this feature means that users can set a Master Key passphrase, which is used to encrypt locally stored data, such as the Threema database or a private key file.

Both security configurations are recommended and are meant to cover distinct use cases and potential attack vectors:

- The Access Protection setting is useful for fending off simple and unsophisticated closed access attacks, e.g. preventing unauthorized parties from reading a user's chat history when getting physical access to a user's device inside a restaurant, bar or similar place.
- The Encryption of locally stored data is an important security feature which elevates the security posture of Threema and protects users from situations where an attacker already has highly privileged access to a victim's phone, thus

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

s/he attempts to decrypt the private key file (resulting in an account takeover) or the Threema database.

The purpose of this ticket is to encourage Threema to adjust the naming / configuration user-interface to be more precise in terms of what is the goal of each, doing so for a typical, non-security affine user. At the current state, it was Cure53's impression that regular users may potentially assume that setting the PIN code credentials, configurable under *'Access Protection',* already provides enough security / safeguards, which may mean they refrain from setting a Master Key passphrase.

Conclusions

As explained in the *Introduction,* Threema has a sensitive purpose and ambitious premise, which absolutely justifies the external scrutiny that was here offered by the Cure53 team. However, despite dedicating sixteen days to the security-centered investigations and reaching the expected coverage, three members of the Cure53 team could only spot seven minor weaknesses on the scope. The absence of vulnerabilities and the generally low severity scores contribute to the positive verdict reached about the security standing of the Threema mobile applications during this October 2020 assessment.

It should be added that the support offered to Cure53 by Threema - especially in the form of tested-communications - was very good and increased the quality of the project's outputs. All design decisions were unambiguously explained and access to the respective documentation and protocol descriptions was provided on time. Daily updates regarding tasks and progress were given to keep the client apprised of the audit moving forward.

Cure53 needs to underline that the overall impression of the code quality and general structure of the project can only be described as unusually solid. The design and implementation were clearly accomplished by a rare team of experienced and security-affine engineers. In Cure53's opinion, there should be no doubt about the focus of these processes being on providing a highly secure messaging application without encumbering the overall user-experience.

Once again, the goal of this assessment was to make sure - in the frame of the externally conducted project - that no major insecurities were left in the application codebase before it is announced through an open source release. There is a caveat to the positive verdict in that only always a certain level of depth can be reached, especially when an extensive, approximately 400k-lines codebase, is at stake. In other words, while the analyses were deemed sufficient, it cannot be guaranteed that no hidden vulnerabilities remain within the Threema scope.

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

To first focus on the Threema for Android, this mobile application has been reviewed both statically (auditing the provided source code) and dynamically (hooking and tampering with various messages that can be triggered by a remote user). Special attention was given to the exposed attack surface reachable remotely, as it could potentially allow malicious users to attack arbitrary Threema users. It must be emphasized that the massive complexity of the Threema Android client and the limited time frame made an exhaustive security assessment unrealistic.

The identified weaknesses are recommendations that could enhance the security of the Threema application. They focus on hardening external third-party applications by using additional compiler flags and keeping dependencies current as well as disabling the automatic screenshot capturing feature when the app is backgrounded. It is important to note that the miscellaneous findings should be interpreted as additional hardening steps, since none of the issues poses a direct or severe risk to the Threema for Android application.

Overall, the Threema Android mobile application is in good shape from a security perspective and was found to have a robust security posture. It is evident that the development team has followed best practices in terms of secure programming and cryptography primitives. Conclusively, even though no actual vulnerabilities with exploitation routes have been identified, it is of utmost importance to stress that this security review should not be considered exhaustive.

Second, moving to the iOS Threema branch, the impressions here mirrored the positive aspects highlighted for the Android counterpart. The overall cryptographic integration is similarly sound. An explicit focus was notably set on the exposed attack surface in the local storage, the keychain, as well as the transport and chat protocol layer setup regarding NaCl. In addition, further checks for problematic functions in the code base or certain compiler flags were performed. The discovered issues mainly concerned local storage configuration and the mentioned compiler flags. However, the proposed countermeasures are again just recommendations rather than necessary steps. The bottom line is that no sensitive files or information were exposed. Conclusively, the Threema iOS client was found to have a robust security posture.

To reiterate, it should be quite clear that a positive verdict was reached by Cure53 during this October 2020 project. This is underlined by the low total score of the problems and the limited severities ascribed to the spotted flaws. Cure53 recommends instating a reasonable bug bounty program in parallel to the open source release. This should sufficiently entice the information security community to continuously keep looking for vulnerabilities in the already existing codebase and any future modifications thereof. The

Dr.-Ing. Mario Heiderich, Cure53
Bielefelder Str. 14
D 10709 Berlin
cure53.de · mario@cure53.de

auditors highly commend the clients on the decision to release the application's codebase as open source in their effort to pave the way for reproducible builds. The broader aim is also clear in that Threema believes in creating an even safer messaging application for their customers. Exposing the project to the scrutiny of the public eye will surely safeguard the users and customers of Threema in the years to come.

Cure53 would like to thank Silvan Engeler, Manuel Kasper, Patrik Oprandi, Danilo Bargen and Martin Blatter from the Threema team for their excellent project coordination, support and assistance, both before and during this assignment.