# Wire Security Review – Phase 2 – Android Client for Wire Swiss GmbH

**Final Report**

2018-03-07

*FOR PUBLIC RELEASE*

# Contents

# 1 Summary

This report is a review of Wire's Android application security and privacy, describing strengths and limitations, and highlighting a number of minor shortcomings and potential improvements. We also report five low-severity software bugs and one high-severity bug. The source code audit otherwise reflected adherence to secure software development principles.

As noted throughout the report, Wire has adequately addressed the security issues reported.

The main Wire repositories covered during the review are wire-android, wire-android-sync-engine, and wire-cryptobox-jni.

This review does not cover:

- the core cryptography component Proteus, previously reviewed [1] ;
- the calling mechanism, covered in a separate review;
- code of third-party dependencies.

The work was performed between March and November 2017, by Jean-Philippe Aumasson (Kudelski Security) and Markus Vervier (X41 D-Sec GmbH), with support from Yolan Romailler (Kudelski Security). A total of 9 person-days were spent. It reflects the code base as provided during the time of review.

[1] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`

# 2 Android Client Review

We reviewed security and privacy features of the Wire Android client, mainly based on the source code review, as well as on dynamic analysis of the official Wire APK using device emulators, a rooted device, and test devices.

The following sections illustrates how the Wire Android application protects against various technical risks, highlighting potential issues and providing mitigation recommendations.

Several observations were made on older versions of the applications (as of March 2017), but all observations reported in this version of the report apply to the version of the app available mid November 2017:

- wire-android: revision 4135eada49b6f8504822cef59bad808dee2da2e7;
- wire-android-sync-engine: 565b72cb22fe315f79850ef728b3baa0ff516d77.

## 2.1 Privacy

This section summarizes our review of potential privacy leaks.

### 2.1.1 Logcat Leaks

We did not find privacy leaks in the logs generated by the released application.

When following the instructions in the provided README.md [1] file, no sensitive data is sent to the logs. However, users building the app themselves should know that a default

[1]`https://github.com/wireapp/wire-android/blob/master/README.md`

build of the app will enable verbose-level logs, which write various sensitive information to the Android logcat. Examples are content of messages, email address and password length at login.

### 2.1.2 Uninstall Leftovers

After installing the application, using it to send and receive a few messages, and uninstalling it, we did not find files created by the installation or the usage of the app that would reveal a previous installation of the app on the device, with the exception of logcat and app usage logs.

The Wire app was found to have a minimal footprint on the device, with downloaded files in the locations one may expect for them. Notably picture and audio files are not stored in a publicly accessible way.

After uninstalling, files are removed with the exception of:

- Video recordings located in the `Pictures/WIRE_MEDIA` folder. Failed recordings are also kept, which should be avoided and fixed. (To reproduce: begin recording, stop recording and kill the wire app when offered with the "Retry/Ok".)
- Downloaded files are kept in the `Download` folder, without any mention of Wire, hence does not reveal the use of Wire. However, Pictures ares stored in a `Wire/` folder.
- Traces can still be found in the logs (logcat, typically) and usage stats of the Android system.

Regarding the persistence of video recordings, we recommend using the internal Android file storage as often as possible instead of the external, world readable one. Upon uninstalling, the internal storage is automatically wiped by Android. This can however be a problem if large videos are taken since those could fill up too much space. The method handling the media capture is in the file `wire-android/app/src/main/java/com/waz/zclient/pages/main/conversation/AssetIntentsManager.java`, where it's requesting external storage instead of internal one to store the video files.

Wire has addressed these issues by removing references to Wire for downloaded files and pictures[2], deleting failed recordings[3] and temporary video files[4].

### 2.1.3 Telemetry Data

Localytics is used to perform analytics based on user usage data. Users can opt out of this tracking in the settings, but by default data such as the following is tracked:

- type of connection (WiFi, 4G, etc.);
- the time, size, and type of assets exchanged (photo, audio, video, or other files);
- call duration and type (audio or video), etc.

Such data can obviously leak privacy-sensitive information, even if the data is shared anonymously. Wire migrated to Mixpanel during the review. The reviewed code was still using Localytics.

### 2.1.4 Crash Reports

HockeyApp is used to collect and crash reports from the application and share them with Wire for QA purposes.

These reports do not seem to include obvious sensitive information, since they mainly contain Java stack traces. Yet stack traces and exception messages can reveal potentially sensitive information such as content URIs:

```
src/main/scala/com/waz/service/downloads/Downloader.scala
25:import com.waz.HockeyApp
165: HockeyApp.saveException(ex, s"video transcoding failed for uri: ${asset.uri}")
186: case cause: Throwable => HockeyApp.saveException(cause, s"audio encoding failed
for URI: $uri")
```

[2]See https://github.com/wireapp/wire-android-sync-engine/pull/301/ and https://github.com/wireapp/wire-android/pull/1400

[3]See https://github.com/wireapp/wire-android/pull/1414

[4]See https://github.com/wireapp/wire-android-sync-engine/pull/308

**Listing 2.1:** Crash Reports

This specific issue and other similar ones have been fixed in the latest version of the app (the asset URI is not recorded anymore).

Wire has addressed this issue by removing sensitive data such as identifiers and URIs from HockeyApp exception descriptions[5]. Additionally users can opt out of this tracking in the settings.

### 2.1.5 Screen Copies

The application attempts to detect and block remote screenshots of the Wire conversation window when it includes *timed messages*, but not otherwise. We tested this feature on a Nexus 5X, but can't guarantee that it's effective on all devices and for all Android versions (it probably isn't).

Locally, the Android app switcher shows a preview of the current Wire screen, and does not attempt to hide sensitive data such as conversations content. We recommend to create an opt-in configurable that allows to protect against unwanted screenshots.

### 2.1.6 Files Metadata

Images sent over the app are stripped of their metadata such as for example geotags. This applies to photos taken from the app or pictures from the image gallery. Videos are not geo-tagged when taken from the app, but if taken from the gallery metadata is not stripped.

Other files uploaded are left unchanged. This includes office documents, audio files, PDFs, and other documents.

[5]See `https://github.com/wireapp/wire-android-sync-engine/pull/269`

### 2.1.7 URL Previews

Previews of URLs are generated when an URL is entered, thereby generating a DNS request and subsequent HTTP requests to fetch the preview content. This behavior leaks the address of the sender to the URL's servers, and reveals that the URL was sent. It was not observed on the receiver's side.

Unlike the iOS application, the Android application doesn't allow users to disable previews.

## 2.2 CRYPTOGRAPHY

This section reports on the main cryptographic threats against the application.

### 2.2.1 Pseudorandom Generator

Android's `SecureRandom` is used to generated cryptographic material, as for example in:

```
object AESUtils {

  lazy val random = new SecureRandom()

  def base64(key: Array[Byte]) = Base64.encodeToString(key, Base64.NO_WRAP |
  ↪  Base64.NO_CLOSE)
  def base64(key: String) = Base64.decode(key, Base64.NO_WRAP | Base64.NO_CLOSE)

  def randomKey(): AESKey = AESKey(returning(new Array[Byte](32)) { random.nextBytes
  ↪  })
  def randomKey128(): AESKey = AESKey(returning(new Array[Byte](16)) {
  ↪  random.nextBytes }
```

**Listing 2.2:** SecureRandom Pseudorandom Generator

`SecureRandom` is an interface to a cryptographically secure PRNG, however `scala.util.Random` is also used, but is considered insecure[6]. It relies on

[6] https://github.com/scala/scala/blob/v2.12.0/src/library/scala/util/Random.scala#L1

`java.util.Random`, which is a linear congruential PRNG, with 48-bit state, which does not have the properties required for cryptographically secure random number generation. We examined all the uses of this insecure PRNG, and none seems security-critical. The use of non-cryptographic PRNG doesn't seem to create security risks because bad randomness would not create a security risk in this context. Nonetheless, using a strong PRNG everywhere is considered best practice.

Wire has addressed the issue by switching from `Random` to `SecureRandom` at several non-security critical places[7].

### 2.2.2 Cryptobox Integration and Usage

Relevant files reviewed are `CryptoBoxService.scala`, `OtrService.scala`, `OtrClient.scala`, `OtrClients.scala`, `CryptoSessionService.scala`, `AccountService.scala`, `ZMessaging.scala`, as well as the cryptobox-jni repository.

Cryptobox objects seem to be deleted insecurely without erasing the memory with zeroes or other values, however after investigation the issue, we concluded that this can't be fixed reliably due to the behavior of flash memory.

```
def deleteCryptoBox() = Future {
  _cryptoBox.foreach(_.close())
  _cryptoBox = None
  IoUtils.deleteRecursively(cryptoBoxDir)
  verbose(s"cryptobox directory deleted")
}
```

**Listing 2.3:** Cryptobox usage

A known technical risk, already discussed with Wire, is the use of SHA-256 as an authenticator such as for example in:

```
private def decodeExternal(key: AESKey, sha: Option[Sha256], extData:
↪  Option[Array[Byte]]) =
  for {
```

[7]See `https://github.com/wireapp/wire-android/pull/1380` and `https://github.com/wireapp/wire-android-sync-engine/pull/293`

```
    data  <- extData if sha.forall(_.matches(data))
    plain <- LoggedTry(AESUtils.decrypt(key, data)).toOption
    msg  <- LoggedTry(GenericMessage(plain)).toOption
  } yield ms
```

**Listing 2.4:** SHA-256 Authenticator

See also `encryptAssetDataCBC()`, which calls `mac` a mere SHA-256 hash of the ciphertext. Wire mentioned to us that this construction will be replaced by `encryptAssetDataGCM()`, which will use AES-GCM authenticated encryption.

### 2.2.3 Assets Handling

The `BadPaddingException` is ignored in `AESUtils.scala`, which is not an issue here as it occurs when input stream is not fully consumed. However, this is only true with the current CBC encryption. This ignored exception would be insecure with AEAD [8], hence the code should be adapted when AES-GCM is deployed:

```
def inputStream(key: AESKey, is: InputStream) = {
  val iv = returning(new Array[Byte](16))(IoUtils.readFully(is, _, 0, 16))

    new CipherInputStream(is, cipher(key, iv, Cipher.DECRYPT_MODE)) {
      ...
      override def close(): Unit = try {
        super.close()
      } catch {
        case io: IOException =>
          io.getCause match {
            case _: BadPaddingException => //ignore
            case e => throw e
          }
```

**Listing 2.5:** BadPaddingException Ignored

[8]`https://blog.heckel.xyz/2014/03/01/cipherinputstream-for-aead-modes-is-broken-in-jdk7-gcm/`

### 2.2.4 Password Protection

TLS-encrypted passwords are hashed using scrypt by the server before being stored, as per the security white paper using parameters $N = 2^{14}$, $r = 8$, $p = 1$, with a random 256-bit salt, which we could verify in the server code at `https://github.com/wireapp/wire-server`. This choice of password hashing provides strong protection against password cracking attacks.

### 2.2.5 Hardware Support

The application does not support hardware-protected keys.

## 2.3 Storage

This section reports on the data stored on the device.

### 2.3.1 Internal Storage

Sensitive data (databases and private keys: identity key and prekeys) are stored in internal storage under `data/data/com.wire/` with permissions preventing other applications to read them. These databases are unencrypted, however.

An excerpt of internally stored data is given below:

```
./data/data/com.wire/lib
./data/data/com.wire/files
./data/data/com.wire/files/.localytics
./data/data/com.wire/files/tmp
./data/data/com.wire/files/otr
./data/data/com.wire/files/otr/<ACCOUNT-ID>
./data/data/com.wire/files/otr/<ACCOUNT-ID>/sessions
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/65535
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/0
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/1
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/2
```

```
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/3
...

./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/96
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/97
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/98
./data/data/com.wire/files/otr/<ACCOUNT-ID>/prekeys/99
./data/data/com.wire/files/otr/<ACCOUNT-ID>/identities
./data/data/com.wire/files/otr/<ACCOUNT-ID>/identities/local
./data/data/com.wire/files/otr/<ACCOUNT-ID>/version
./data/data/com.wire/files/assets
./data/data/com.wire/files/assets/1a
./data/data/com.wire/files/assets/4d
./data/data/com.wire/databases
./data/data/com.wire/databases/ZGlobal.db
./data/data/com.wire/databases/ZGlobal.db-wal
./data/data/com.wire/databases/ZGlobal.db-shm
./data/data/com.wire/databases/com.localytics.android.<ID>.analytics.sqlite
./data/data/com.wire/databases/com.localytics.android.<ID>.in-app.sqlite
./data/data/com.wire/databases/com.localytics.android.<ID>.profile.sqlite
./data/data/com.wire/databases/com.localytics.android.<ID>.in-app.sqlite-journal
./data/data/com.wire/databases/com.localytics.android.<ID>.analytics.sqlite-journal
./data/data/com.wire/databases/com.localytics.android.<ID>.profile.sqlite-journal
./data/data/com.wire/databases/<ACCOUNT-ID>
./data/data/com.wire/databases/<ACCOUNT-ID>-wal
```

**Listing 2.6:** Stored Data

### 2.3.2 External Storage

Encrypted assets are written to external storage (typically, an SDcard), and can only be decrypted using the corresponding key stored in the protected database in `data/data/com.wire/databases/`. Other applications can thus see the number and size of assets (media, files), but not their clear content.

We could verify that the files stored on the SDcard can be decrypted using the secret key in the local database. However, these files are only encrypted (with AES-128 in CBC mode), and not authenticated. This allows another application to surreptitiously modify them.

## 2.4 Network

This section reports on network security risks.

### 2.4.1 TLS Connection

No HTTP content has been detected except when accessing HTTP links provided to the application by external applications, or when fetching link previews.

Connections to the backend and wire.com servers use TLS 1.2 with the `TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384` cipher suite, however platforms that don't support this configuration will fall back to a potentially weaker TLS version and cipher suite. This logic is implemented in `ClientWrapper.scala`:

```scala
val domains @ Seq(zinfra, wire) = Seq("zinfra.io", "wire.com")
val protocol = "TLSv1.2"
val cipherSuite = "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384"
(...)
override def configureEngine(engine: SSLEngine, data: AsyncHttpClientMiddleware.GetSocketData, host: String, port: Int): Unit = {
  debug(s"configureEngine($host, $port)")

  if (domains.exists(host.endsWith)) {
    verbose("restricting to TLSv1.2")
    engine.setSSLParameters(returning(engine.getSSLParameters) { params =>
      if (engine.getSupportedProtocols.contains(protocol)) params.setProtocols(Array(protocol))
      else warn(s"$protocol not supported by this device, falling back to defaults.")

      if (engine.getSupportedCipherSuites.contains(cipherSuite)) params.setCipherSuites(Array(cipherSuite))
      else warn(s"cipher suite $cipherSuite not supported by this device,
      falling back to defaults.")
    })
```

**Listing 2.7:** TLS Connection

As a response Wire has now switched to TLSv1.2, which prevents the fallback.

### 2.4.2 Pinning

The application performs certificate pinning to enforce the usage of specific CA certificates. The most relevant source code files are, in the sync engine, `ClientWrapper.scala` and `ServerTrust.scala`.

A DigiCert root certificate is hardcoded, pinned in order to validate certificates for the backend domains `wire.com` and `zinfra.io`. CA certificate pinning is applied to hostnames having any of the two above top-level domains. The hostname is verified to match the certificate's CN or SAN record.

However, the root certificate provided to validate the CDN server's certificate only uses a 1024-bit RSA modulus:

```
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        Public-Key: (1024 bit)
        Modulus:
            00:c9:5c:59:9e:f2:1b:8a:01:14:b4:10:df:04:40:
            db:e3:57:af:6a:45:40:8f:84:0c:0b:d1:33:d9:d9:
            11:cf:ee:02:58:1f:25:f7:2a:a8:44:05:aa:ec:03:
            1f:78:7f:9e:93:b9:9a:00:aa:23:7d:d6:ac:85:a2:
            63:45:c7:72:27:cc:f4:4c:c6:75:71:d2:39:ef:4f:
            42:f0:75:df:0a:90:c6:8e:20:6f:98:0f:f8:ac:23:
            5f:70:29:36:a4:c9:86:e7:b1:9a:20:cb:53:a5:85:
            e7:3d:be:7d:9a:fe:24:45:33:dc:76:15:ed:0f:a2:
            71:64:4c:65:2e:81:68:45:a7
                Exponent: 65537 (0x10001)
    Signature Algorithm: md2WithRSAEncryption
    (...)
```

**Listing 2.8:** 1024 bit Public Key

We recommend that at least a 2048-bit certificate should be used. Here the use of the MD2 hash function is unfortunate but is not a security risk since this certificate is a root certificate.

Wire has initially addressed the issue by directly pinning the 2048-bit root key[9]. It has now migrated away from CA pinning to pinning the leaf certificate public key.

[9] See `https://github.com/wireapp/wire-android-sync-engine/pull/315`

## 2.5 Platform

This section reports on general security risks related to the Android platform.

### 2.5.1 Permissions

The Wire application requests the following permissions, as defined in `app/src/main/AndroidManifest.xml`:

```
android.permission.ACCESS_NETWORK_STATE
android.permission.WRITE_EXTERNAL_STORAGE
android.permission.INTERNET
android.permission.READ_CONTACTS
android.permission.RECORD_AUDIO
android.permission.VIBRATE
android.permission.CAMERA
android.permission.FLASHLIGHT
android.permission.READ_PHONE_STATE
android.permission.MODIFY_AUDIO_SETTINGS
android.permission.BLUETOOTH
android.permission.WAKE_LOCK
com.google.android.c2dm.permission.RECEIVE
android.permission.ACCESS_FINE_LOCATION
```

**Listing 2.9:** Android Permissions

None of these permissions appear superfluous.

### 2.5.2 Components Security

We performed basic sanity checks, using drozer[10], and observed that the application does not export any content provider. This is a good practice in terms of security. The application exports four activities to other applications without requiring permissions, namely:

[10] `https://labs.mwrinfosecurity.com/tools/drozer/`

```
com.waz.zclient.MainActivity
com.waz.zclient.ShareActivity
com.waz.zclient.LaunchActivity
com.waz.zclient.SMSCodeReceiverActivity
```

**Listing 2.10:** Activities

The first three exported activities appear necessary, but Wire observed that `SMSCodeReceiverActivity` may not be used anymore, and therefore should not be exported.

Wire has addressed the issue by removing the `SMSCodeReceiverActivity`[11].

### 2.5.3 Root Detection

The app does not include any attempt of root detection or any anti-debug mechanisms. Anti-debug protection is sometimes used to complicate reverse engineering, but this isn't a concern here since the application being fully open-source.

## 2.6 Code Quality

This section reports on general issues regarding the code used in the application.

### 2.6.1 Native Libraries

The following native libraries are used:

```
libavs.so
libcryptobox.so
libjnidispatch.so
libsodium.so
libspotify_sdk.so
libcryptobox-jni.so
libgnustl_shared.so
```

[11] See `https://github.com/wireapp/wire-android/pull/1398`

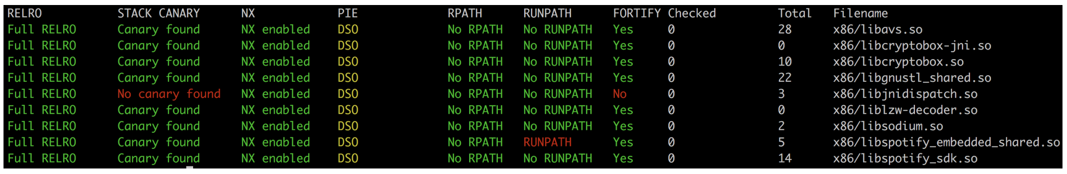
```
liblzw-decoder.so
libspotify_embedded_shared.so
```

**Listing 2.11:** Native Libraries

The AVS and Cryptobox libraries are provided by Wire. AVS includes some third-party code. Both libraries were reviewed in the previous phase (partially for AVS).

We did not review the source code of third-party libraries, but we checked that they are recent versions at the time of review. Additionally basic properties of the binary files are verified:

- `gnustl` and `spotify\_sdk` are not stripped of symbols (x86).
- Stack canaries and fortify source parameters are missing in `jnidispatch`, as showed by the results of the checksec [12] utility, as depicted in figure 2.1.



```
RELRO        STACK CANARY     NX          PIE          RPATH     RUNPATH     FORTIFY Checked     Total   Filename
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           28      x86/libavs.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           0       x86/libcryptobox-jni.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           10      x86/libcryptobox.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           22      x86/libgnustl_shared.so
Full RELRO   No canary found  NX enabled  DSO          No RPATH  No RUNPATH  No      0           3       x86/libjnidispatch.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           0       x86/liblzw-decoder.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           2       x86/libsodium.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  RUNPATH     Yes     0           5       x86/libspotify_embedded_shared.so
Full RELRO   Canary found     NX enabled  DSO          No RPATH  No RUNPATH  Yes     0           14      x86/libspotify_sdk.so
```

**Figure 2.1:** Checksec Results.

Wire has addressed this issue by adding the missing stack canaries, and removing the unused Spotify library[13].

### 2.6.2 Dependencies

Java and Scala dependencies listed in `build.gradle` include the following:

```
        scala              : "org.scala-lang:scala-library:$scalaVersion",
        scalaReflect       : "org.scala-lang:scala-reflect:$scalaVersion",

        // Lint dependencies
```

[12] https://github.com/slimm609/checksec.sh
[13] See https://github.com/wireapp/wire-android-sync-engine/pull/304

```
        lintapi            : 'com.android.tools.lint:lint-api:24.5.0',
        lintchecks         : 'com.android.tools.lint:lint-checks:24.5.0',

        // Checkstyle dependencies
        checkstyleapi      : "com.puppycrawl.tools:checkstyle:$checkstyleVersion",

        // Support libs
        multidex           : "com.android.support:multidex:1.0.1",
        supportv4          : "com.android.support:support-v4:$supportLibVersion",
        supportv13         : "com.android.support:support-v13:$supportLibVersion",
        supportdesign      : "com.android.support:design:$supportLibVersion",
        appcompatv7        : "com.android.support:appcompat-v7:$supportLibVersion",
        recyclerview       : "com.android.support:recyclerview-v7:$supportLibVersion",
        supportannotations : "com.android.support:support-annotations:$supportLibVersion",
        preferences        : "com.android.support:preference-v7:$supportLibVersion",
        cardview           : "com.android.support:cardview-v7:$supportLibVersion",

        // Play services
        psBase             : "com.google.android.gms:play-services-base:$playServicesVersion",
        psGcm              : "com.google.android.gms:play-services-gcm:$playServicesVersion",
        psMaps             : "com.google.android.gms:play-services-maps:$playServicesVersion",
        psLocation         : "com.google.android.gms:play-services-location:$playServicesVersion",

        // Other
        timber             : 'com.jakewharton.timber:timber:4.1.1',
        hockey             : 'net.hockeyapp.android:HockeySDK:3.7.2',
        threetenabp        : 'com.jakewharton.threetenabp:threetenabp:1.0.3',
        localytics         : 'com.localytics.android:library:3.8.2',
        rebound            : 'com.facebook.rebound:rebound:0.3.8',
        supportpreferences : 'net.xpece.android:support-preference:0.8.1',

        // Internal
        audioNotifications : "com.wire:audio-notifications:$audioVersion",

        stetho             : 'com.facebook.stetho:stetho:1.4.2',

        // Test dependencies
        junit              : 'junit:junit:4.12',
        testutils          : "com.wire:testutils:$zMessagingDevVersionBase",
        scalatest          : "org.scalatest:scalatest_$scalaMajorVersion:2.2.6",
        testRunner         : 'com.android.support.test:runner:0.4.1',
        testRules          : 'com.android.support.test:rules:0.4.1',
        espresso           : 'com.android.support.test.espresso:espresso-core:2.2',
        espressoIntents    : 'com.android.support.test.espresso:espresso-intents:2.2',
        hamcrestCore       : 'org.hamcrest:hamcrest-core:1.3',
        hamcrestLib        : 'org.hamcrest:hamcrest-library:1.3',
        hamcrestIntegration: 'org.hamcrest:hamcrest-integration:1.3',

        mockitoCore        : 'org.mockito:mockito-core:1.10.19',
        //The dexmaker stuff is needed for Mockito to work completely
        dexmaker           : 'com.crittercism.dexmaker:dexmaker:1.4',
        dexmakerDx         : 'com.crittercism.dexmaker:dexmaker-dx:1.4',
        dexmakerMockito    : 'com.crittercism.dexmaker:dexmaker-mockito:1.4',

        // Translations
        translations       : 'com.wire:wiretranslations:1.+',

        //Json parser
        gson               : 'com.google.code.gson:gson:2.2.4'
```

**Listing 2.12:** Dependencies

Furthermore, dependencies of the sync engine listed in `build.sbt` include:

```
      Deps.supportV4 % Provided,
      "com.koushikdutta.async" % "androidasync" % "2.1.8",
      "com.googlecode.libphonenumber" % "libphonenumber" % "7.1.1", // 7.2.x breaks
   ↪ protobuf
      "com.softwaremill.macwire" %% "macros" % "2.2.2" % Provided,
      "com.google.android.gms" % "play-services-base" % "7.8.0" % Provided
   ↪ exclude("com.android.support", "support-v4"),
      "com.google.android.gms" % "play-services-gcm" % "7.8.0" % Provided,
      Deps.avs % Provided,
      Deps.cryptobox,
      Deps.genericMessage,
      Deps.backendApi,
      "com.wire" % "icu4j-shrunk" % "57.1",
      Deps.spotifyPlayer,
      "org.threeten" % "threetenbp" % "1.3" % Provided,
      "com.googlecode.mp4parser" % "isoparser" % "1.1.7",
      Deps.hockeyApp % Provided,
      Deps.localytics,
      "net.java.dev.jna" % "jna" % "4.2.0",
      "org.robolectric" % "android-all" % RobolectricVersion % Provided
```

**Listing 2.13:** Sync Engine Dependencies

Among these dependencies, `AndroidAsync` has known vulnerabilities on older versions of Android (<4.0.3, see `https://github.com/koush/AndroidAsync/issues/478`), however the Wire application is only for Android 4.2 and higher, so this is not considered to be an issue.

### 2.6.3 Build Errors and Warnings

A build in Android Studio reports a number of errors (deprecated classes, etc.) and warnings. These are likely known by the developers though, and do not seem to pose any security risk.

### 2.6.4 Input Validation

We only report two minor observations regarding validation of input supplied through the UI:

1. The type of file transmitted is inferred from the file's extension, rather than from the file's magic signature. This simplifies attacks exploiting weaknesses in parsers of specific files formats. It would also be more convenient for users if file types are identified from their signatures.
2. Newlines in messages are not stripped in sent messages when displayed in the Android client, unlike in desktop and iOS, as observed in our emulated Nexus 5X. This is not considered to be a security issue.

## 2.7 Implementation Security Issues

Security issues that are resulting from implementation level vulnerabilities are described here.

All of these issues have been addressed by Wire, and we reviewed the relevant patches to confirm their effectiveness.

### 2.7.1 WIRE-Android-01: Bias in the Cryptographic PRNG

| | |
|---|---|
| *Severity:* | *LOW* |
| *CWE:* | *330* |

#### 2.7.1.1 Description

The JNI wrapper to libsodium's `randombytes_buf` checks that the buffer holding the generated pseudorandom bytes is not all-zero, allegedly because it could indicate a problem in the PRNG.

The affected code is in file `wire-android-sync-engine/zmessaging/src/main/jni/randombytes.c`:

```c
JNIEXPORT jboolean JNICALL Java_com_waz_utils_crypto_RandomBytes_randomBytes(JNIEnv
 ↪  *jenv, jobject obj, jbyteArray jarr, jint jcount) {
    unsigned char* buffer = (unsigned char *) (*jenv)->GetByteArrayElements(jenv,
 ↪  jarr, 0);
    size_t count = (size_t) jcount;

    randombytes_buf(buffer, count);

    // check if returned data is not empty
    int success = 0;
    for (int i = 0; i < count && !success; ++i) {
        success |= buffer[i];
    }

    (*jenv)->ReleaseByteArrayElements(jenv, jarr, (jbyte *) buffer, 0);
    return success;
}
```

**Listing 2.14:** PRNG Bias

However, an unintended consequence of this check is that if the caller requests one random byte, then this byte will never be zero. Likewise with two bytes, three bytes, etc. This introduces a bias in the PRNG, especially when it's used to request bytes one at a time.

This bias does not seem exploitable in Wire, however, because `RandomBytes` is only used to request 16 or more bytes, in which case the bias is negligible.

Wire removed the check for zero and fixed the bias [14] .

#### 2.7.1.2 Solution Advice

The check for zero can be safely removed, because libsodium's is unlikely to fail; libsodium initialization will return an error if it fails to find a proper PRNG. This return value is properly checked in the code shown in listing 2.15.

```
jint JNI_OnLoad(JavaVM * vm, void* reserved) {
    if (sodium_init() < 0) {
      return -1;
    }

    return JNI_VERSION_1_6;
  }
```

**Listing 2.15:** libsodium Check

[14] https://github.com/wireapp/wire-android-sync-engine/pull/303

### 2.7.2 WIRE-Android-02: Potential Out-of-Bound Write in the LZW Decoder

| | |
|---|---|
| *Severity:* | *LOW* |
| *CWE:* | *787* |

#### 2.7.2.1 Description

Source code for `liblzw-decoder.so` can be found in `LzwDecoder.cpp` and `LzwDecoder.h` of the sync engine. This code is used in `LzwDecoder.scala`, called in `AnimGifDecoder.scala`. The processed GIF image files could be attacker controlled. We therefore consider them untrusted.

Issues discovered in this code include:

- `LzwDecoder::clear` does not check that `x` and `y` are within the allowed bounds, and therefore allows to write arbitrary data at arbitrary (positive or negative) offsets from `*dst`.
- Invalid bound check in decode: if `data_size=image[idx++];` (`image[0]`) is equal to 31, then `clear=1<<data_size;` is negative and therefore the check `if(clear>=MAX_STACK_SIZE)return;` is successfully passed. Likewise values than 31 yield `clear=0`, passing the check again. This again would allow for out-of-bound write.

However, the C LZW decoder does not seem to pose security issues as used in Wire. This is because an attacker only controls the file's content and not directly the other parameters passed to the decoder by the Wire application. A limitation, however, is file size: both `LzwDecoder.cpp`, callers in `LzwDecoder.scala`, as well as `Gif.scala` use `int` types for file length and dimensions. Files of one gigabyte or more would thus overflow these parameters and lead to incorrect decoding.

Security checks for arithmetic overflows and bounds checks have been introduced [15] by Wire to fix this issue.

[15] `https://github.com/wireapp/wire-android-sync-engine/pull/297`

#### 2.7.2.2 Solution Advice

`LzwDecoder.scala` should be patched to add the missing bound checks.

### 2.7.3 WIRE-Android-02: Path Traversal When Saving Files

| | |
|---|---|
| *Severity:* | *HIGH* |
| *CWE:* | *23* |

#### 2.7.3.1 Description

While saving a file received by a contact, the filename is not sanitized in file `AssetService.scala`:

```scala
    def getTargetFile(dir: File): Option[File] = {
        val baseName = asset.name.getOrElse("wire_downloaded_file." + asset.mime.extension) // XXX: should get default file name form resources
        // prepend a number to the name to get unique file name,
        // will try sequential numbers from 0 - 10 first, and then fallback to random ones
        // will give up after 100 tries
        val prefix = ((0 to 10).iterator ++ Iterator.continually(Random.nextInt(10000))).take(100)
        prefix.map(i => new File(dir, nextFileName(baseName, i))).find(!_.exists())
      }
```

Listing 2.16: Directory Traversal

If the filename contains "../", a directory traversal is possible when writing the file in function `saveAssetData`:

```scala
    def saveAssetData(file: File) =
    loaderService.load(asset, force = true)(loader).future.map {
      case Some(data) =>
        //TODO Dean: remove after v2 transition period
        //Trigger updating of meta data for assets generated (and downloaded) from old AnyAssetData typ
        asset.mime match {
          case Mime.Video() if asset.metaData.isEmpty || asset.previewId.isEmpty => updateMetaData(asset, data)
          case Mime.Audio() if asset.metaData.isEmpty => updateMetaData(asset, data)
          case _ => CancellableFuture.successful(Some(asset))
        }

        IoUtils.copy(data.inputStream, new FileOutputStream(file))
        Some(file)
      case None =>
        None
    } (Threading.IO)
```

**Listing 2.17:** Function saveAssetData

### 2.7.3.2 Solution Advice

It is recommended to remove all characters that allow directory traversal such as for example leading dots and also all forward slash ("/") characters. Ideally the pathname should be normalized and fully resolved before checking if the path prefix is equal to "Environment.DIRECTORY_DOWNLOADS" as defined by the environment.

### 2.7.4 WIRE-Android-03: In-App WebViews Loading Remote Content

| | |
|---|---|
| *Severity:* | *LOW* |
| *CWE:* | *829* |

#### 2.7.4.1 Description

Several places in the application use inapp-webviews to display possibly remote content (from Wire servers) using the function `onOpenUrlInApp` defined in `AppEntryActivity.scala`:

```
  def onOpenUrlInApp(url: String, withCloseButton: Boolean): Unit = {
    val prefixedUrl =
      if (!url.startsWith(AppEntryActivity.HTTP_PREFIX) &&
↪ !url.startsWith(AppEntryActivity.HTTPS_PREFIX))
        AppEntryActivity.HTTP_PREFIX + url
      else
        url
    val transaction = getSupportFragmentManager.beginTransaction
    transaction.setCustomAnimations(R.anim.new_reg_in, R.anim.new_reg_out)
    transaction.add(R.id.fl_main_web_view,
↪ InAppWebViewFragment.newInstance(prefixedUrl, withCloseButton),
↪ InAppWebViewFragment.TAG)
    transaction.addToBackStack(InAppWebViewFragment.TAG)
    transaction.commit
    KeyboardUtils.hideKeyboard(this)
  }
```

**Listing 2.18:** Webview

An example is the terms of service URL `https://wire.com/legal/terms/embed/` that is displayed as link on the login screen:

```
  termsOfService.foreach{ text =>
    TextViewUtils.linkifyText(text, ContextCompat.getColor(getContext,
↪ R.color.white), true, new Runnable {
        override def run() =
↪  getContainer.onOpenUrlInApp(getString(R.string.url_terms_of_service),
↪  withCloseButton = true)
      })
    }
```

**Listing 2.19:** Terms of Service Link

Since the loaded content is not visible as external content, a compromised Wire server might trick users by creating fake navigational elements such as a login screen.

The issue has been fixed [16] by Wire. An external browser is opened for remote content.

### 2.7.4.2 Solution Advice

It is recommended to include all relevant content as static resources in the app. In case this is impossible, external resources should be opened in a separate Chrome browser tab.

[16] https://github.com/wireapp/wire-android/pull/1399

### 2.7.5 WIRE-Android-04: Share Activity Does Not Sanitize Shared File

| | |
|---|---|
| *Severity:* | *LOW* |
| *CWE:* | *668* |

#### 2.7.5.1 Description

When a sharing intent is received, the shared file is not sanitized to be from shared storage. Other untrusted apps on the device could potentially create malicious share intents that point to files only accessible by Wire.

The code handling share intents is defined in `ShareActivity.scala`:

```
 private void handleIncomingIntent() {
   ShareCompat.IntentReader intentReader = ShareCompat.IntentReader.from(this);
   if (!intentReader.isShareIntent()) {
       finish();
       return;
   }
...
           for (int i = 0; i < intentReader.getStreamCount(); i++) {
               Uri uri = intentReader.getStream(i);
               if (uri != null) {
                   sharedFileUris.add(new AndroidURI(uri));
               }
           }
       } else {
           Uri uri = intentReader.getStream();
           if (uri != null) {
               sharedFileUris.add(new AndroidURI(uri));
           }
       }
...
       List<URI> sanitisedUris = new ArrayList<>();
       for (URI uri : sharedFileUris) {
           String path = AssetUtils.getPath(getApplicationContext(), uri);
           if (path == null) {
               Timber.e("Something went wrong, unable to retrieve path");
               sanitisedUris.add(uri);
           } else {
               sanitisedUris.add(AndroidURIUtil.fromFile(new File(path)));
           }
       }

       switch (contentType) {
           case IMAGE:
               getSharingController().publishImageContent(sanitisedUris);
               break;
           case FILE:
               getSharingController().publishFileContent(sanitisedUris);
               break;
       }
   }
 }
```

**Listing 2.20:** Share Intent

As seen above *URIs* are read from streams of a received intent. These URIs may point to any file on the file system. Therefore a user might be tricked into sending out private files or contents of special files to a contact. Note that the user has to explicitly select a contact that the files will be shared with. However, this is not far fetched when talking about unsuspecting users.

The issue has been fixed [17] by Wire.

### 2.7.5.2 Solution Advice

It is recommended to sanitize received files. Ideally only files from shared directories and external storage should be accepted. In any case files from special directories such as `/dev`, `/proc`, and `/data/app/` should be rejected.

[17] https://github.com/wireapp/wire-android/pull/1406

### 2.7.6 WIRE-Android-05: Untrusted Websites Can Force-Logout Users

| | |
|---|---|
| *Severity:* | *LOW* |
| *CWE:* | *352* |

#### 2.7.6.1 Description

The special URL `wire://password-reset-successful` registered by the Wire app can be navigated to from untrusted web sites in Chrome on Android. This will force the current user of the Wire app to be logged out.

The scheme is registered in file `AndroidManifest.xml` as follows:

```
  <intent-filter>
  <data
      android:host="password-reset-successful"
      android:scheme="wire"
      />

  <action android:name="android.intent.action.VIEW" />

  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  <category android:name="android.intent.category.VIEW" />
</intent-filter>
```

**Listing 2.21:** Password Reset Scheme

The following names are registered for the “wire:” scheme in file `IntentUtils.java`:

```
  public class IntentUtils {

    public static final String WIRE_SCHEME = "wire";
    public static final String EMAIL_VERIFIED_HOST_TOKEN = "email-verified";
    public static final String PASSWORD_RESET_SUCCESSFUL_HOST_TOKEN =
 ↪   "password-reset-successful";
    public static final String SMS_CODE_TOKEN = "verify-phone";
    public static final String INVITE_HOST_TOKEN = "connect";
...
```

**Listing 2.22:** Registered Names

Two other registered names seem to be not used anymore and do not expose dangerous actions.

Wire fixed [18] this issue by checking an authentication cookie for server side invalidation.

#### 2.7.6.2 Solution Advice

It is recommended to provide signaling over secured channels for events such as resetting a password. The backend system could for example provide such a signal. Additionally the need for registering a dedicated scheme should be evaluated. For the benefit of security the number of intents and especially schemes should be kept low.

[18] https://github.com/wireapp/wire-android-sync-engine/pull/302

# 3 About

Kudelski Security
route de Genève, 22-24
1033 Cheseaux-sur-Lausanne
Switzerland

**Kudelski Security** is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships. Kudelski Security is a division of Kudelski Group.

For more information, please visit `https://www.kudelskisecurity.com`.

X41 D-Sec GmbH
Dennewartstr. 25-27
D-52068 Aachen
Germany

**X41 D-Sec** is an expert provider for application security services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world class security experts enables **X41 D-Sec** to perform premium security services.

Fields of expertise in the area of application security are security centered code reviews, binary reverse engineering and vulnerability discovery. Custom research and a IT security consulting and support services are core competencies of **X41 D-Sec**.

For more information, please visit `https://www.x41-dsec.de`.