Fine penetration tests for fine websites

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

Pentest-Report U4I Nahoft Android App 07.2021

Cure53, Dr.-Ing. M. Heiderich, BSc. C. Kean

Index

Introduction

This report describes the results of a penetration test and source code audit against the Nahoft mobile application, which is a tool offering file encryption services on Android phones. The work was requested by United for Iran (U4I) in June 2021 and scheduled for the following month. Importantly, the application in question is maintained by the Operator Foundation team.

Cure53 punctually carried out this assessment in mid-July 2021, namely in CW28. A total of five days were invested to reach the coverage expected for this project. A team of two Cure53 senior testers were assigned to this project's preparation, execution and finalization.

The work was structured using one work package (WP) which spanned white-box-style penetration tests and audits of the U4I Nahoft Android application. At this point, it must be underlined that this test is not the first one conducted by Cure53 against this scope. The first one, documented as *U4I-03*, was executed by Cure53 in November 2020 and focused primarily on the cryptography-related features offered by the app.

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

For the current project, registered as *U4I-06,* Cure53 was given access to a working APK as well as all relevant sources and test-supporting material. Quite clearly, white-box methodology was chosen and deployed. All preparations were done in early July 2021, namely in CW27, ensuring a smooth transition into the testing phase.

Communications during the test were done using a dedicated shared Slack channel made available by U4I. All relevant team members from U4I, Operator Foundation and Cure53 could partake in discussions there. Still, not many questions had to be asked, the scope was well-prepared and clear; no noteworthy roadblocks were encountered during the test.

Cure53 gave frequent status updates about progress and emerging findings, especially as live-reporting was requested and executed by Cure53. The Operator Foundation team was able to address several of the issues while the test was still ongoing.

On the whole, the Cure53 team managed to get very good coverage over the Nahoft app scope items and managed to spot a total of five findings, four of which were classified to be security vulnerabilities and one to be a general weakness with lower exploitation potential. It is worth noting that the majority of issues were given *Medium* severity ratings and enable problems characterized as Denial-of-Service within relevant app features. One exception in U4I-06-005 covers a flaw with *Critical* implications. The elevated severity is justified by the fact that attackers could use this issue to bypass the passcode check, therefore gaining access to clear-text that is no longer protected by Nahoft's encryption.

In the following sections, the report will first shed light on the scope and key test parameters. Next, all findings will be discussed in grouped vulnerability and miscellaneous categories, then following a chronological order in the first group. Alongside technical descriptions, PoC and mitigation advice are supplied when applicable. Finally, the report will close with broader conclusions about this summer 2021 project. Cure53 elaborates on the general impressions and reiterates the verdict based on the testing team’s observations and collected evidence. Tailored hardening recommendations for the Nahoft mobile application complex are also incorporated into the final section.

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

Scope

- **White-Box Penetration Tests & Audits against U4I Nahoft Android App**
  - In scope was the latest version of the Nahoft Android App available on GitHub
    - https://github.com/OperatorFoundation/Nahoft.git
  - Sources and debug binary of Nahoft app were shared with Cure53.
  - An updated binary with issues remediated was provided at the end of the penetration test for fix verification.
  - No server or API is in scope as the app operates completely offline.

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

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *U4I-06-001*) for the purpose of facilitating any future follow-up correspondence.

U4I-06-002 Android: DoS via intent to *ImportTextActivity* (*Medium*)

It was found that the Nahoft Android app exposes its *ImportTextActivity* to third-party apps. A malicious application could leverage this weakness to crash the *Nahoft* app at any time by sending a crafted Intent. The impact of this issue was evaluated as *Medium* as a malicious app running in the background could do this continuously to ensure that the app is completely unusable, thus causing Denial-of -Service (DoS).

The Android Manifest file indicates that an intent filter is set for the affected activity and causes it to be explicitly exported.

Affected file:
AndroidManifest.xml

Affected code:

```
<activity android:name="org.nahoft.nahoft.activities.ImportTextActivity"
android:excludeFromRecents="true" android:screenOrientation="portrait">
        <intent-filter>
                <action android:name="android.intent.action.SEND" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="text/*" />
        </intent-filter>
</activity>
```

The *IntentFuzzer* app can be used to simulate a serializable intent being sent to the Nahoft app. The following steps can be used to verify this issue.

PoC:
https://github.com/MindMac/IntentFuzzer

Steps to reproduce:

1. Open the *Nahoft* app and push it to the background while running.
2. Record the Android logs locally via:
   ```
   adb logcat > log.txt
   ```

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

3. Open the *IntentFuzzer* app and select *NonSystemApps* > *org.nahoft*.
4. Scroll down in the activities and long press *org.nahoft.nahoft.activities.ImportTextActivity* until an intent is sent.
5. Confirm in the logcat output that a serializable intent caused a fatal crash in *org.nahoft*.

Crash output (syslog):

```
07-14 13:55:44.088  4465  4465 E AndroidRuntime: FATAL EXCEPTION: main
07-14 13:55:44.088  4465  4465 E AndroidRuntime: Process: org.nahoft, PID: 4465
07-14 13:55:44.088  4465  4465 E AndroidRuntime: java.lang.RuntimeException:
Unable to start activity
ComponentInfo{org.nahoft/org.nahoft.nahoft.activities.ImportTextActivity}:
java.lang.RuntimeException: Parcelable encountered ClassNotFoundException
reading a Serializable object (name =
com.android.intentfuzzer.util.SerializableTest)
```

The crash is caused by an exception raised when attempting to read a serializable intent that is not expected by the activity, as shown in the source code below.

Affected file:
app/src/main/java/org/nahoft/Nahoft/activities/ImportTextActivity.kt

Affected code:

```
override fun onCreate(savedInstanceState: Bundle?)
       {
val maybeFriend =
intent.getSerializableExtra(RequestCodes.friendExtraTaskDescription) as? Friend
[...]
 }
```

In order to mitigate this issue, it is recommended to correctly validate the data received via intents within *ImportTextActivity*. This would ensure that the situation whereby a malicious application attempts to cause the Nahoft application to crash by sending serialized Java objects is avoided completely.

Note: *Successful remediation of this issue was verified by Cure53 in the latest APK build provided.*

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

U4I-06-003 Android: DoS via intent to *ImportImageActivity* (Medium)

Similar to the above, it was found that the Nahoft Android app exposes its *ImportImageActivity* to third-party apps. A malicious application could leverage this weakness to crash the Nahoft app at any time by sending a crafted Intent. The impact of this issue was evaluated as *Medium* as a malicious app running in the background could do this continuously to ensure that the app is completely unusable, thus causing Denial-of-Service (DoS).

The Android Manifest file indicates that an intent filter is set for the affected activity and causes it to be explicitly exported.

Affected file:
AndroidManifest.xml

Affected code:

```
<activity android:name="org.nahoft.nahoft.activities.ImportImageActivity"
android:excludeFromRecents="true" android:screenOrientation="portrait">
        <intent-filter>
            <action android:name="android.intent.action.SEND" />
            <category android:name="android.intent.category.DEFAULT" />
            <data android:mimeType="image/*" />
        </intent-filter>
</activity>
```

The *IntentFuzzer* app can be used to simulate a serializable intent being sent to the Nahoft app. The following steps can be used to verify this issue.

PoC:
https://github.com/MindMac/IntentFuzzer

Steps to reproduce:

1. Open the *Nahoft* app and push it to the background while running.
2. Record the Android logs locally via:
   `adb logcat > log.txt`
3. Open the *IntentFuzzer* app and select *NonSystemApps* > *org.nahoft*.
4. Scroll down in the activities and long press *org.nahoft.nahoft.activities.ImportImageActivity* until an intent is sent.
5. Confirm in the logcat output that a serializable intent caused a fatal crash in *org.nahoft*.

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

Crash output (syslog):

```
07-14 15:17:32.193 11978 11978 E AndroidRuntime: FATAL EXCEPTION: main
07-14 15:17:32.193 11978 11978 E AndroidRuntime: Process: org.nahoft, PID: 11978
07-14 15:17:32.193 11978 11978 E AndroidRuntime: java.lang.RuntimeException:
Unable to start activity
ComponentInfo{org.nahoft/org.nahoft.nahoft.activities.ImportImageActivity}:
java.lang.RuntimeException: Parcelable encountered ClassNotFoundException
reading a Serializable object (name =
com.android.intentfuzzer.util.SerializableTest)
```

The crash is caused by an exception raised when attempting to read a serializable intent that is not expected by the activity, as shown in the source code below. Although it was not possible to exploit other intent handlers during this engagement, they are highlighted for completeness' sake. In particular, they also lack data validation measures.

Affected file:
app/src/main/java/org/nahoft/Nahoft/activities/ImportImageActivity.kt

Affected code:
```
override fun onCreate(savedInstanceState: Bundle?)
       {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_import_image)

       makeSureAccessIsAllowed()

       registerReceiver(receiver, IntentFilter().apply {
              addAction(LOGOUT_TIMER_VAL)
       })
       // Check to see if a friend was selected in a previous activity
       val maybeFriend =
intent.getSerializableExtra(RequestCodes.friendExtraTaskDescription) as? Friend
[...]
@ExperimentalUnsignedTypes
private fun receiveSharedMessages()
{
    // Receive shared messages
    if (intent?.action == Intent.ACTION_SEND)
    {
        if (intent.type?.startsWith("image/") == true)
        {
            val extraStream =
intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM)
            if (extraStream != null)
[...]
 else // See if we got intent extras from the Login Activity
```

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

```
    {
        // See if we received an image message
        val extraStream =
intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM)
             if (extraStream != null)
[...]
private fun sendToLogin()
{ [...]
if (intent?.action == Intent.ACTION_SEND)
    {
        if (intent.type?.startsWith("image/") == true)
        {
        val extraStream =
intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM)
           if (extraStream != null){
               val extraUri = Uri.parse(extraStream.toString())
                loginIntent.putExtra(Intent.EXTRA_STREAM, extraUri)
         }
```

It is recommended to extrapolate the mitigation guidance offered under U4I-06-002 to solve this issue.

Note: *Successful remediation of this issue was verified by Cure53 in the latest APK build provided.*

U4I-06-004 Android: DoS via intent to *LogInActivity* (*Medium*)

Further to the above, it was found that the Nahoft Android app exposes its *LogInActivity* to third-party apps. A malicious application could leverage this weakness to crash the Nahoft app at any time by sending a crafted intent. The impact of this issue was evaluated as *Medium* as a malicious app running in the background could do this continuously to ensure that the app is completely unusable, thus causing Denial-of-Service (DoS).

The Android Manifest file indicates that an intent filter is set for the affected activity and causes it to be explicitly exported.

Affected file:
AndroidManifest.xml

Affected code:

```
<activity
        android:name=".activities.LogInActivity"
        android:excludeFromRecents="true"
        android:launchMode="singleTask"
        android:noHistory="true"
```

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

```
        android:screenOrientation="portrait"
        tools:ignore="LockedOrientationActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

The *IntentFuzzer* app can be used to simulate a serializable intent being sent to the Nahoft app. The following steps can be used to verify this issue.

PoC:
https://github.com/MindMac/IntentFuzzer

Steps to reproduce:

1. Open the *Nahoft* app and push it to the background while running.
2. Record the Android logs locally via:
   `adb logcat > log.txt`
3. Open the *IntentFuzzer* app and select *NonSystemApps* > *org.nahoft*.
4. Scroll down in the activities and long press *org.nahoft.nahoft.activities.LogInActivity* until an Intent is sent.
5. Confirm in the logcat output that a serializable intent caused a fatal crash in *org.nahoft*.

Crash output (syslog):

```
07-14 13:57:40.719  6992  6992 E AndroidRuntime: FATAL EXCEPTION: main
07-14 13:57:40.719  6992  6992 E AndroidRuntime: Process: org.nahoft, PID: 6992
07-14 13:57:40.719  6992  6992 E AndroidRuntime: java.lang.RuntimeException:
Unable to start activity
ComponentInfo{org.nahoft/org.nahoft.nahoft.activities.LogInActivity}:
java.lang.RuntimeException: Parcelable encountered ClassNotFoundException
reading a Serializable object (name =
com.android.intentfuzzer.util.SerializableTest)
```

The crash is caused by an exception raised when attempting to read a serializable intent that is not expected by the activity, as shown in the source code below.

Affected file:
app/src/main/java/org/nahoft/Nahoft/activities/LogInActivity.kt

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

Affected code:

```
private fun tryLogIn(status: LoginStatus)
{
    when (status)
    {
        // If the user has logged in successfully or if they didn't set a
passcode
        // Send them to the home screen
        LoginStatus.LoggedIn, LoginStatus.NotRequired ->
        {
             val extraString = intent.getStringExtra(Intent.EXTRA_TEXT)
             val extraStream =
intent.getParcelableExtra<Parcelable>(Intent.EXTRA_STREAM)
[...]
}
```

It is recommended to extrapolate the mitigation guidance offered under U4I-06-002 to solve this issue.

Note: *Successful remediation of this issue was verified by Cure53 in the latest APK build provided.*

U4I-06-005 Android: Passcode bypass allows message decryption *(**Critical**)*

It was discovered that the passcode code does not protect the activities in regard to decrypting images and text messages. It can be simply bypassed by traversing to the activity previous to the *LogIn* activity. The impact of this issue was evaluated as *Critical* as it allows an attacker who does know the passcode to decrypt any messages and images received in the third-party messengers.

The following steps can be used to verify this issue.

Steps to reproduce:

1. Activate the app-internal passcode lock and log out of the Nahoft app.
2. In a third-party messenger, select a text encrypted with Nahoft and share it with the Nahoft app.
3. This will open the passcode lock screen, however, if the user just switches back to the previous activity, it shows *ImportTextActivity*.
4. Enter the encrypted text and press *Import Text*.
5. Confirm that the text was decrypted successfully.

Affected files:

- *app/src/main/java/org/nahoft/Nahoft/activities/ImportTextActivity.kt*
- *app/src/main/java/org/nahoft/Nahoft/activities/ImportImageActivity.kt*

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

Affected code:

```
override fun onCreate(savedInstanceState: Bundle?)
{
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_import_text)
       makeSureAccessIsAllowed()
[...]
}

private fun makeSureAccessIsAllowed()
{
Persist.getStatus()

if (Persist.status == LoginStatus.NotRequired || Persist.status ==
LoginStatus.LoggedIn)
       {            return       }
       else         {            sendToLogin()       }
}

private fun sendToLogin()
{
 // If the status is not either NotRequired, or Logged in, request login
this.showAlert(getString(R.string.alert_text_passcode_required_to_proceed))

 // Send user to the Login Activity
 val loginIntent = Intent(applicationContext, LogInActivity::class.java)

 // We received a shared message but the user is not logged in
 // Save the intent
 if (intent?.action == Intent.ACTION_SEND)
    {
              if (intent.type == "text/plain")
              {
              val messageString = intent.getStringExtra(Intent.EXTRA_TEXT)
              loginIntent.putExtra(Intent.EXTRA_TEXT, messageString)
              }
        [...]
    }
    startActivity(loginIntent)
}
```

It is recommended to stop exporting activities that provide critical decryption features. The implementation of intents redirects to the lock screen proves to be challenging as they can be easily bypassed and tampered with. A less stricter measure would be to disable the decryption function while the passcode is not entered. This will make the

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

activity still directly accessible, however an attacker will not be able to decrypt messages.

Note: *The first remediation attempt did not resolve this issue. The problem remained reproducible with the following steps.*

The *IntentFuzzer* app can be used to simulate a Null Intent being sent to the Nahoft app.

PoC:
https://github.com/MindMac/IntentFuzzer

Steps to reproduce:

1. Activate the app-internal passcode lock and log out of the Nahoft app.
2. In the *IntentFuzzer* app, send a Null Intent to *ImportTextActivity*.
3. This will open the passcode lock screen, however, if the user just pushes the *IntentFuzzer* app to the background and reopens it again, they will see the *ImportTextActivity* again.
4. Enter the encrypted text and press *Import Text*.
5. Confirm that the text was decrypted successfully.

Note: *The remediated source code and binary were provided to Cure53 and the issue was ultimately confirmed as resolved.*

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

Miscellaneous Issues

This section covers those noteworthy 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.

U4I-06-001 Android: General hardening recommendations for Android app *(Info)*

During the assessment of the Nahoft app, it was discovered that not all security flags offered by Android are utilized. The absence of these flags does not introduce a security issue but could allow an attacker to exploit other problems more easily. As such, the flags described below should be considered as defense-in-depth mechanisms.

FLAG_SECURE

By setting the *FLAG_SECURE* for Android views, the app's windows can no longer be manually "*screenshotted*"[1]. Additionally, the items will be excluded from automatic screenshots or screen-recordings, which ultimately prevents screen data from being leaked to other apps. Including this flag is especially recommended for the implemented views that show sensitive data, such as medical images, which are received by the doctor and displayed in the app.

filterTouchesWhenObscured

The *filterTouchesWhenObscured* security flag for views protects against so-called Tapjacking attacks. A malicious app can overlap the currently active app with a hidden screen overlay. The latter would need to have the ability to intercept data entered into the underlying app. Once the flag is set, a view will no longer receive touches when it is obscured by another window, therefore making this attack infeasible[2].

Note: *The remediated source code and binary were provided to Cure53 and the issue was confirmed as resolved.*

[1] https://developer.android.com/reference/android/view/WindowManager.Layou...ml#FLAG_SECURE
[2] https://blog.devknox.io/tapjacking-android-prevent/

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

Conclusions

This July 2021 project fits into the broader efforts to secure the U4I Nahoft complex. This time around, two members of the Cure53 team focused on evaluating the Nahoft mobile application for Android. After dedicating five days to this examination, they spotted five items relevant from a security stance. Although most issues reside in the realm of *Medium* risks, the presence of one *Critical* flaw should not be disregarded.

To give some details, the codebase of the Nahoft Android app was subject to code-assisted penetration testing. The app provides an encryption service which is operating purely offline while piggy-backing on third-party apps to take care of delivering messages. Thus, the threat model of this engagement mainly focused on threats arising from an attacker who manages to obtain the phone and control malicious, third-party apps.

The exposed activities, broadcasts, content providers and services were audited for manipulation via intents or data leakage. In particular the exported activities were found to cause the majority of issues filed in this report, with the U4I-06-002 to U4I-06-005 range. In Cure53's view, the security implications concerning parsing of and redirecting with intents are not fully mitigated, as reflected by the absence of strict countermeasures against intent tampering in the codebase. Apart from these primary issues, the app could provide better protection to its users by limiting screenshots as well as tapjacking in the context of the app (U4I-06-001). The app was also analyzed for insecure storage which could lead to information leaks by inspecting the memory via *ADB.* It was found that the app uses *EncryptedSharedPreferences*[3] to successfully protect its secrets at rest. Thus, no issues were identified as far as secure storage is concerned.

To conclude with the verdict on its mobile security, the Nahoft mobile applications made a mixed impression in terms of minimizing the attack surface. This is evident especially in the context of using intents. The proposed measures should be interpreted as necessary steps to help harden the Nahoft mobile app and put it on the right trajectory with regard to mobile security. That being said, all discovered issues were remediated shortly after being reported to the Nahoft team. Therefore, the app now appears ready for secure use in production.

Cure53 would like to thank Milad Keshtan and Yasaman Choubeh from the United for Iran team as well as Adelita Schule and Jessica Garcia of Operator Foundation for their excellent project coordination, support and assistance, both before and during this assignment.

[3] https://developer.android.com/reference/androidx/security/crypto/EncryptedSharedPreferences