Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report Zom Mobile Apps and API 04.2016

Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Aranguren, N. Hippert, Dr. J. Magazinius, D. Weißer

Index

- Introduction
- Scope
- Identified Vulnerabilities
  - ZOM-01-001 iOS and Android: Lack of Pinning for https traffic (Medium)
  - ZOM-01-002 Android: Lack of screen capture protections (High)
  - ZOM-01-003 iOS: Possible hijacking via unprotected files at rest (Medium)
  - ZOM-01-004 Android: Complete lack of Tapjacking mitigations (Medium)
  - ZOM-01-005 Android: MitM without user warnings via lack of TLS (Critical)
  - ZOM-01-006 Android app supports plain-text authentication (High)
  - ZOM-01-008 API Server: Django logins available over clear-text HTTP (High)
  - ZOM-01-009 iOS app supports plain-text authentication (Medium)
  - ZOM-01-010 iOS: Permanent DoS via unknown contact message (High)
  - ZOM-01-011 Android: DoS via crafted message with high size value (Medium)
  - ZOM-01-012 iOS and Android: Automated linkification of phone URLs (Medium)
  - ZOM-01-015 iOS and Android: Unclear messages during TLS MitM (Medium)
  - ZOM-01-016 iOS: ChatSecure UserVoice Pod uses clear-text HTTP (Medium)
- Miscellaneous Issues
  - ZOM-01-007 XMPP Server supports plain-text authentication (Low)
  - ZOM-01-014 API: Assignment weaknesses on the device endpoint (Info)
- Conclusion

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Introduction

"Zom is the new way to gather together with your friends on the go. Easy Setup! Zom helps you create an account quickly and find ways to connect with your friends. Totally Free! No costs for messages, and no limits on what you say, or who you can talk to. Great Sharing! Easily send voice messages, share photos, send stickers and more! Multiple Accounts! You can create and use different identities for home, work, family and more. Any Server, Any Where! Unlike other apps that keep you stuck in their walled garden, Zom is fully interoperable with any app that supports OTR and XMPP, such as ChatSecure, Conversations, Adium, Jitsi, and more."

From https://zom.im

This report documents the findings of a security assessment of the Zom application, carried out by five members of the Cure53 team in late April and early May of 2016.

The test has been scheduled to encompass fifteen days and eleven of them have been thus far used to complete the goals of the project. The remaining four days are to be dedicated to a through protocol review. The scope of the assignment discussed in this report encompassed tests against mobile applications. The sources were made available to the Cure53 testers by the Zom team, and a test-version for each application was shared. The assessment focused on Android and iOS versions of the Zom suite.

The test has yielded a total of sixteen issues, comprising fourteen vulnerabilities and two general weaknesses. One discovery, namely the problem described under ZOM-01-005, was ranked "Critical" with regard to security implication. This was due to the fact that the identified lack of TLS allowed for a Man-in-the-Middle (MitM) attack without issuing warnings to the Android users. Other findings varied in severity but were not deemed as detrimental to the overall good state of security identified within the Zom application.

Scope

- **Zom Android Code:**
  - https://github.com/zom/Zom-Android/releases/tag/15.0.1-BETA-3
- **Zom Android APK:**
  - https://zom.im/download/zom.apk
- **Zom iOS Code:**
  - https://github.com/zom/Zom-iOS/releases/tag/1.0.3-28-cure53

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 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 a 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. *ZOM-01-001*) for the purpose of facilitating any future follow-up correspondence.

ZOM-01-001 iOS and Android: Lack of Pinning for https traffic *(Medium)*

It was found that the iOS app does not currently implement any form of certificate or public key pinning when it communicates with the API server or the Uservoice Forum. A malicious attacker with a certificate trusted by the relevant iOS certificate store (i.e. most governments and attackers with considerable resources) could therefore intercept network communications and hijack user sessions. As a consequence, malicious attackers could impersonate logged-in Zom users on the API and forum servers. In addition, this process would have taken place without any warnings issued to the users. A less severe issue also occurs on the Android app, but only in situations where a user voluntarily sends a crash report via the *hockeyapp* module.

The following examples illustrate some of the information available to the attackers once communications are intercepted through this vulnerability.

Example 1: iOS: All API calls can be intercepted

This weakness is enough for a Man-in-the-Middle (MitM) to interact on behalf of the user with the API server, hence capturing information. In effect, the MitM attacker is able to perform some destructive actions like modifying account data on the API server.

The code snippets below reveal some of the data available to the MitM.

Request:

```
POST https://push.zom.im/api/v1/accounts/ HTTP/1.1
[...]

{"username":"04D49550-FA03-4559-8A24-
F16238","password":"6Ci7QlgASRzdBvr0ujyiXpIoGMROchzDJwNih0UDd09Weix3QuoWqPTVP+YB
Dt3cWOFDMLDDEnfhCyBpavehRbi1tXlnghbocbPA"}
```

Response:

```
HTTP/1.1 200 OK
Set-Cookie: __cfduid=d37318fe96b7a14568ab02f658cc296001461196023; expires=Thu,
20-Apr-17 23:47:03 GMT; path=/; domain=.zom.im; HttpOnly
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
[...]

{"username":"04D49550-FA03-4559-8A24-F16238","email":"","id":"22b5e7fb-1ac9-
4f60-9cff-da60b8695167","token":"68d89c02e2618259c3e6ec9745b5559e9f8dac58"}
```

Request:

```
POST https://push.zom.im/api/v1/tokens/ HTTP/1.1
[...]
{"apns_device":"8c304fc4-95b1-4d20-adca-fdfcab34f1f1"}
```

Response:

```
HTTP/1.1 201 Created
Set-Cookie: __cfduid=dee9f0132aece604bd7d98d3866308e281461196024; expires=Thu,
20-Apr-17 23:47:04 GMT; path=/; domain=.zom.im; HttpOnly
[...]

{"token":"a8b51fadf59d2c893fb782dc7126078ff5bd5552","apns_device":"8c304fc4-
95b1-4d20-adca-fdfcab34f1f1"}
```

Example 2: iOS: Creating a Forum user

iOS users can access Forum when they choose to send feedback.

Request:

```
POST https://chatsecure.uservoice.com/api/v1/users.json HTTP/1.1
[...]
request_token=MY87hMT1OzHCgPKeNEg&user[display_name]=&user[email]=abraham
%40cure53.de
```

Response:

```
HTTP/1.1 200 OK
[...]
{"user":
{"url":"http://chatsecure.uservoice.com/users/163007019","id":163007019,"name":"
Anonymous","title":null,"guid":null,"anonymous":false,"email":"abraham@cure53.de
","email_confirmed":false,"authentication":{"provider":"external"},"roles":
{"owner":false,"admin":false},"avatar_url":"https://secure.gravatar.com/avatar/8
405a2b384d504a17a8f93e7cd71893a?
size=70\u0026default=https://assets0.uvcdn.com/pkg/admin/icons/user_70-
62136f6de7efc58cc79dabcfed799c01.png","karma_score":0,"created_at":"2016/04/27
18:23:16 +0000","updated_at":"2016/04/27 18:23:16
+0000","supported_suggestions_count":0,"created_suggestions_count":0,"visible_fo
rums":[{"id":229504,"name":"ChatSecure -
Android","is_private":false,"idea_count":46,"url":"/forums/229504-chatsecure-
android","max_votes":10,"forum_activity":
{"votes_available":10,"supported_suggestions":[]}},
{"id":193939,"name":"ChatSecure -
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

iOS","is_private":false,"idea_count":131,"url":"/forums/193939-chatsecure-ios","max_votes":10,"forum_activity":{"votes_available":10,"supported_suggestions":[]}}]},"token":{"oauth_token":"iCpK42UEKUAXQMJrxCjw","oauth_token_secret":"dr1a1nL5Ua9vJd1WCGppips38RP3QbUQlVvLq4KjCok"}}

Example 3: Android: Crashes sent via *hockeyapp* use SSL but no pinning

This is a less severe issue, which simply leaks data to the MitM attacker.

Request:

```
POST
https://sdk.hockeyapp.net/api/2/apps/3cd4c5ff8b666e25466d3b8b66f31766/crashes/
HTTP/1.1
[...]

raw=Package: im.zom.messenger
Version Code: 1501800
Version Name: 15.0.1-BETA-3
Android: 6.0.1
Manufacturer: LGE
Model: Nexus 5X
CrashReporter Key: D006DBA9-B12D-3ED0-61D7-22AD2A7189DC96FFF8EB
Date: Fri Apr 22 19:30:43 GMT+02:00 2016

java.lang.RuntimeException: Unable to start activity
ComponentInfo{im.zom.messenger/org.awesomeapp.messenger.ui.LockScreenActivity}:
java.lang.NullPointerException: Attempt to invoke virtual method 'boolean
java.lang.String.equalsIgnoreCase(java.lang.String)' on a null object reference
       at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2416)
       at
android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2476)
       at android.app.ActivityThread.-wrap11(ActivityThread.java)
       at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1344)
       at android.os.Handler.dispatchMessage(Handler.java:102)
       at android.os.Looper.loop(Looper.java:148)
       at android.app.ActivityThread.main(ActivityThread.java:5417)
       at java.lang.reflect.Method.invoke(Native Method)
       at
com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
       at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
       at de.robv.android.xposed.XposedBridge.main(XposedBridge.java:133)
Caused by: java.lang.NullPointerException: Attempt to invoke virtual method
'boolean java.lang.String.equalsIgnoreCase(java.lang.String)' on a null object
reference
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
        at
org.awesomeapp.messenger.ui.LockScreenActivity.checkCustomFont(LockScreenActivit
y.java:403)
        at
org.awesomeapp.messenger.ui.LockScreenActivity.onCreate(LockScreenActivity.java:
72)
        at android.app.Activity.performCreate(Activity.java:6251)
        at
android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1107)
        at
android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2369)
        ... 10 more
&userID=&contact=&description=#ChatSecure debug trail file
#Fri Apr 22 18:52:08 GMT+02:00 2016
prev_service_create=2016-04-22T16\:08\:14UTC
service_create=2016-04-22T16\:52\:07UTC
database_open=2016-04-22T16\:52\:07UTC
connections=2
prev_database_open=2016-04-22T16\:08\:14UTC
last_boot=2016-04-22T16\:52\:07UTC
empty_key=
prev_connections=2
&sdk=HockeySDK&sdk_version=3.5.0
```

It is recommended to implement pinning on both the Android and iOS apps, at least for the default servers. While this will not prevent researchers from inspecting API calls, it will protect average users from governments and well-funded attackers able to forge the widely trusted SSL certificates. For more information about pinning, including Android and iOS examples, please see the *OWASP Pinning Cheat Sheet*[1] and the *OWASP Certificate and Public Key Pinning*[2] technical guide.

ZOM-01-002 Android: Lack of screen capture protections *(High)*

It was found that the Android app does not currently implement screen capture protections. Even in the latest Android versions, this allows malicious apps with root privileges to take screenshots or even full videos in the background, without any user-interaction whatsoever. The ways in which malicious apps usually gain root privileges entail either simply prompting the user on a rooted phone, or exploiting a known vulnerability on an outdated device. Malicious apps without root privileges can also accomplish this but require a user-prompt, which might only be shown once, assuming that the user taps on the "*Do not show again*" message. As a consequence, all information displayed on the screen, including secret chats and photos displayed while

[1] https://www.owasp.org/index.php/Pinning_Cheat_Sheet
[2] https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

the Zom app is in the foreground, could in fact be stolen by malicious apps using screenshots as a side-channel.

It is relatively easy to verify this issue with the use of the following commands from a non-root *adb* shell. This needs to take place while the app is open and displaying some sensitive information. It will produce a readable screenshot because screen capture protections are currently not implemented:

Commands:

```
adb shell screencap -p /mnt/sdcard/screenshot1.png
adb pull /mnt/sdcard/screenshot1.png
```

This screenshot leakage weakness can be used to steal chat conversations, photos, contacts and anything else displayed by the app on screen:

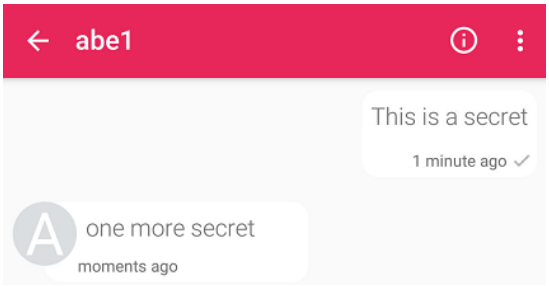

Fig.: Chat capture via screenshot

The verification of the permission prompt required by the malicious apps without root privileges can be observed in the popular Android screenshot and video recording apps. Some of the apps that do not require root privileges are *AZ Screen Recorder - No Root*[3] or *Screenshot Easy*[4]. Please note that if the user taps on the "*Don't show again*" checkbox, no further prompts will ever be shown again to the user, despite the fact that a video recording or a screenshot come to fruition:

[3] https://play.google.com/store/apps/details?id=com.hecorat.screenrecorder.free&hl=en
[4] https://play.google.com/store/apps/details?id=com.icecoldapps.screenshoteasy

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

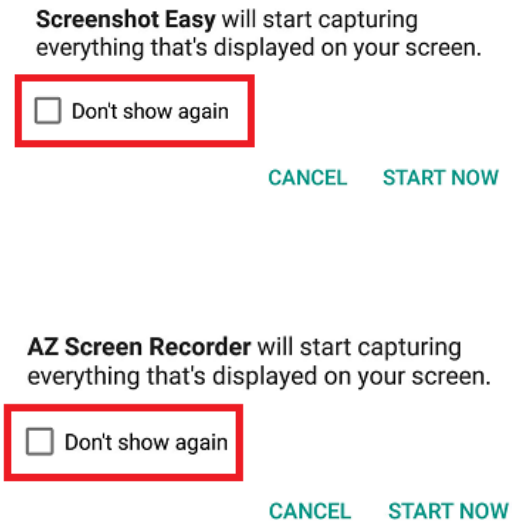

Fig.: Permission prompt required by non-root apps

As mentioned above, a more pressing concern is that, given the reality of the outdated and compromised Android ecosystem[5], absolutely no permission prompts will be shown to the user when an app with root privileges captures everything that is happening on the phone (i.e. record a full video or continuously take screenshots). In the paper *Security Metrics for the Android Ecosystem*[6], published on October 2015, researchers from the *University of Cambridge* showed that root privileges can in fact be gained on 87.7% of Android phones through a security vulnerability. The authors specifically caution:

> *"We find that on average 87.7% of Android devices are exposed to at least one of 11 known critical vulnerabilities"*

What further impacts the severity of this issue is that, on a rooted phone, a malicious app can simply prompt the user for root privileges, which is a common practice in Android. Many apps like *Easy Screenshot*[7] and others behave this way. By default, upon the root privileges prompt, the root privileges will simply be indefinitely granted if the user just taps on "*Grant*":

[5] https://public.gdatasoftware.com/Presse/Publikatio...s/G_DATA_MobileMWR_Q1_2015_US.pdf
[6] https://www.cl.cam.ac.uk/~drt24/papers/spsm-scoring.pdf
[7] https://play.google.com/store/apps/details?id=com.enlightment.screenshot&hl=en

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

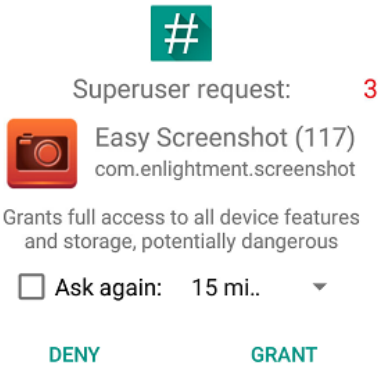

Fig.: Root privileges are granted indefinitely by default

What needs to be underscored is that a malicious app with root privileges can also fake interaction using commands such as "*input tap <x> <y>*", etc. Therefore, with the ability to view what is displayed on the screen, malicious apps with root privileges can also interact with the UI. The latter means that the malicious apps can fully impersonate users when they are not actively using their phones and leave the app open in the foreground.

In the context of the tested application, which handles sensitive information, it is recommended to ensure that all web views have the Android's *FLAG_SECURE* flag[8] set. This will guarantee that even apps running with root privileges cannot capture the information displayed by the app. It is advised that a fix similar to the following is implemented and ideally treated as a base activity that all other activities inherit:

Proposed Fix:

```
public class BaseActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /**
         * approach 1: create a base activity and set the FLAG_SECURE in it,
         * Extend all other activities, Fragments from this activity
         */
        getWindow().setFlags(LayoutParams.FLAG_SECURE,
                LayoutParams.FLAG_SECURE);
    }
```

[8] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

If the proposed solution is considered unfeasible, another possible approach could be to amend the *onCreate* method of the relevant Views on Android, making the corrective adjustment depicted below:

```
protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
       getWindow().setFlags(LayoutParams.FLAG_SECURE,
       LayoutParams.FLAG_SECURE);
}
```

ZOM-01-003 iOS: Possible hijacking via unprotected files at rest *(Medium)*

It was found that the Zom iOS app fails to take full advantage of the built-in iOS platform protections which provide encryption at rest. Therefore the ZOM iOS app is vulnerable to filesystem attacks even when the device is locked. Although some files are protected, the OTR private keys, instance tags and OTR fingerprints are all stored in clear-text. A malicious attacker could leverage this weakness to attempt to pose as a victim user or gain information about the user's contacts.

This issue can perhaps be best described with the output of the *tar* command from a jail-broken device at a stage where an iPhone was on the lock screen. In this illustrative case some files fail to copy, however, the private key file is one of many entities stored in clear-text:

Command:

```
tar cvfz ios_files_locked.tar.gz *
```

Output:

```
Documents/
Documents/otr.fingerprints
Documents/otr.instance_tags
Documents/otr.private_key
Library/
Library/Application Support/
Library/Application Support/Zom/
Library/Application Support/Zom/ChatSecure-media.sqlite
Library/Application Support/Zom/ChatSecure-media.sqlite-shm
Library/Application Support/Zom/ChatSecure-media.sqlite-wal
Library/Application Support/Zom/ChatSecureYap.sqlite
Library/Application Support/Zom/ChatSecureYap.sqlite-shm
Library/Application Support/Zom/ChatSecureYap.sqlite-wal
Library/Caches/
Library/Caches/LaunchImages/
Library/Caches/LaunchImages/im.zom.messenger/
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
tar: Library/Caches/LaunchImages/im.zom.messenger/LaunchImage-
LandscapeLeft{667,375}@2x.png: Cannot open: Operation not permitted
tar: Library/Caches/LaunchImages/im.zom.messenger/LaunchImage-
Portrait{375,667}@2x.png: Cannot open: Operation not permitted
Library/Caches/Snapshots/
Library/Caches/Snapshots/im.zom.messenger/
Library/Caches/Snapshots/im.zom.messenger/im.zom.messenger/
tar:
Library/Caches/Snapshots/im.zom.messenger/im.zom.messenger/UIApplicationAutomati
cSnapshotDefault-Portrait@2x.png: Cannot open: Operation not permitted
Library/Caches/Snapshots/im.zom.messenger/im.zom.messenger/downscaled/
tar:
Library/Caches/Snapshots/im.zom.messenger/im.zom.messenger/downscaled/UIApplicat
ionAutomaticSnapshotDefault-Portrait@2x.png: Cannot open: Operation not
permitted
Library/Caches/im.zom.messenger/
Library/Caches/im.zom.messenger/Cache.db
Library/Caches/im.zom.messenger/Cache.db-shm
Library/Caches/im.zom.messenger/Cache.db-wal
Library/Caches/im.zom.messenger/com.apple.opengl/
Library/Caches/im.zom.messenger/com.apple.opengl/compileCache.data
Library/Caches/im.zom.messenger/com.apple.opengl/compileCache.maps
Library/Caches/im.zom.messenger/com.apple.opengl/linkCache.data
Library/Caches/im.zom.messenger/com.apple.opengl/linkCache.maps
Library/Caches/im.zom.messenger/com.apple.opengl/shaders.data
Library/Caches/im.zom.messenger/com.apple.opengl/shaders.maps
Library/Caches/im.zom.messenger/fsCachedData/
Library/Caches/im.zom.messenger/fsCachedData/2C1B5763-A882-4746-B8EB-
66D0691186FA
Library/Caches/im.zom.messenger/fsCachedData/34B47BCB-71E4-435E-ADE7-
91D73F4DD2A6
Library/Caches/im.zom.messenger/fsCachedData/D2165B74-63EC-41D2-920D-
82605097352C
Library/Caches/im.zom.messenger/fsCachedData/FE670082-231A-43E9-B7D0-
8585800B005A
Library/Cookies/
Library/Cookies/Cookies.binarycookies
Library/Preferences/
Library/Preferences/im.zom.messenger.plist
```

From here onwards, being in possession of the OTR files alone, the attacker has access to all OTR private keys, OTR fingerprints and the fingerprint contacts:

File:
otr.private_key

Contents:
```
(privkeys
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
 (account
(name "abe1@dukgo.com")
(protocol xmpp)
(private-key
 (dsa
  (p #0090D683788[...]#)
  (q #00B190E65F3[...]#)
  (g #42BA1B8AF5A[...]#)
  (y #1BD78CE1FBC[...]#)
  (x #3125E5EBBCA[...]#)
  )
 )
 )
)
```

File:
otr.instance_tags

Contents:

```
# WARNING! You shouldn't copy this file to another computer. It is unnecessary
and can cause problems.
abe1@dukgo.com      xmpp   22c37a3a
```

File:
`otr.fingerprints`

Contents:

```
abedroid.6ed89f80@dukgo.com     abe1@dukgo.com  xmpp
f985ce400ab2f588e4871c660881e86f6ed89f80        verified
```

Other leaked information includes all user-activity on the user-forums via the following SQLite database, as well as the associated cached files in plain-text, which cache all user-visited links and activity on the forum, including searches:

Library/Caches/im.zom.messenger/Cache.db

Please note that although files like `ChatSecure-media.sqlite` and `ChatSecureYap.sqlite` are encrypted using SQLCipher, they could leverage iOS protections as an additional layer of security. In order to solve this problem it is recommended to implement the *NSFileProtectionComplete* entitlement at the app level[9].

[9] https://developer.apple.com/library/ios/documentation/iP...App/StrategiesforImplementingYourApp.html

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

ZOM-01-004 Android: Complete lack of Tapjacking mitigations *(Medium)*

It was found that the Zom app currently fails to mitigate tapjacking attacks. This allows malicious apps who have neither root privileges nor special permissions to nevertheless be able to fool users into tapping and typing. As a result they see screens controlled by the malicious app while the taps and keystrokes reach the Zom app. Consequently, the user is led to believe they are not interacting with the malicious app because it is rendered on top. In addition to this, by rendering a transparent overlay on top, malicious apps should be able to capture screenshots due to the issue described in ZOM-01-002.

This issue was verified on a rooted phone running Android 6.0.1. The crafted tapjacking PoC APK opens the app in the background while the user sees something else in the foreground. A short video was recorded to further elaborate on how taps to the PoC screen (rendered on top) reach the target app underneath.

Aside from the above issue, it was also discovered that global searches for tapjacking protections return no matches in the source code provided, hence confirming a complete lack of these protections on all views and buttons used by the app. It is recommended to implement the *filterTouchesWhenObscured*[10][11] attribute at the Android WebView level[12]. The latter would help ensure that taps from potentially malicious apps rendered on top are ignored.

ZOM-01-005 Android: MitM without user warnings via lack of TLS *(**Critical**)*

The Android app alerts users when the intended XMPP server TLS certificate is not trusted by any CA. This takes place even if it has pinning capabilities, so that a certificate cannot be easily forged by a trusted CA. However, when one uses a XMPP server without TLS capabilities, it turns out that the Android app will attempt to authenticate. This means that it will try to send XMPP messages in clear-text, despite the fact that a default configuration is in place. The process will take place without any security warnings whatsoever. By default, the XMPP configuration has the "*Require TLS Connection*" setting enabled and *xmpp.dukgo.com* setup as the XMPP server to use.

An attacker can spoof the DNS response for *xmpp.dukgo.com* to point the app to a malicious XMPP server without TLS capabilities. He or she can then proxy the XMPP traffic to the real server from there. This provides the attacker with all user-traffic, while the user sees no errors issued by the app. Although the app will attempt to establish a secure OTR channel before sending a message, the contact is already leaked in the XML and a malicious server could just spoof a reply stating that OTR is not supported.

[10] http://developer.android.com/reference/android/view/View.html#setFilterTouche...Obscured(boolean)
[11] http://developer.android.com/reference/android/view/View.html#attr_android:filter...henObscured
[12] https://cordova.apache.org/docs/en/latest/guide/platforms/android/webview.html

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

This would mean getting access to all information between the two parties involved. Please note that this attack only works against the Android app. Conversely, on iOS the app states that the *startTLS* policy is required when the same attack is attempted. Please note that, as explained in the introduction, all the examples provided in this finding were verified with the "*Require TLS Connection*" XMPP setting enabled:

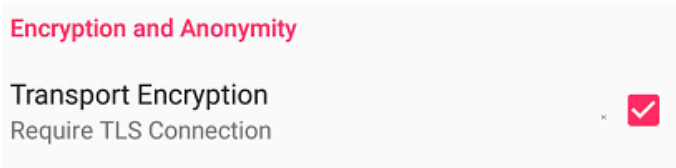

Fig.: Default XMPP Encryption setting on Android

To confirm and observe how this attack is successful, one needs to follow the steps listed below.

Step 1: (Optional) Spoof DNS replies to point the apps to the attacker's XMPP server

Note: This step is technically optional because an attacker with the ability to manipulate network communications can simply implement the next steps and redirect traffic. However, this step might still be useful for making the attack more reliable.

When a DNS request is made to *xmpp.dukgo.com,* the DNS proxy will return an attacker-controlled IP address. This will make the apps contact the attacker's XMPP server instead. The sequence was accomplished with the use of *dnschef*[13] as follows:

Command:

```
dnschef -i 192.168.7.128 --fakedomains=xmpp.dukgo.com,dukgo.com
--fakeip=192.168.7.128
```

Output:

```
[...]
[*] DNSChef started on interface: 192.168.7.128
[*] Using the following nameservers: 8.8.8.8
[*] Cooking A replies to point to 192.168.7.128 matching: xmpp.dukgo.com
[*] Cooking A replies to point to 192.168.7.128 matching: dukgo.com,
xmpp.dukgo.com
[18:06:58] 192.168.7.191: cooking the response of type 'A' for xmpp.dukgo.com to
192.168.7.128
```

[13] http://thesprawl.org/projects/dnschef/

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Step 2: Set up an XMPP server without TLS, have it pretend to be *xmpp.dukgo.com*

With no TLS in place, no user-warnings are issued. Thus, by implementing an XMPP server without the TLS capabilities (or spoofing that message from the server), the Android app is forced to use clear-text. P*rosody*[14] was used to achieve this goal and the following configuration disabling the TLS module was used. Importantly it then pretends to be *xmpp.dukgo.com*:

File:
/etc/prosody/prosody.cfg.lua

Contents:
```
modules_enabled = {

        -- Generally required
                "roster"; -- Allow users to have a roster. Recommended ;)
                "saslauth"; -- Authentication for clients and servers.
Recommended if you want to log in.
                -- "tls"; -- Add support for secure TLS on c2s/s2s connections
[...]
VirtualHost "xmpp.dukgo.com"
        enabled = true -- Remove this line to enable this host

VirtualHost "dukgo.com"
        enabled = true -- Remove this line to enable this host
```

This server can then be initiated as follows:

Command:
```
/etc/init.d/prosody start
```

Step 3: Capture and modify traffic

From this point onwards, the attacker has an XMPP server that Android apps will connect to in clear-text without any user warnings. In this context, OTR messages can simply be replaced and fallback into clear-text conversations that the attacker can observe and modify as required.

[14] https://prosody.im/

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

The following clear-text XMPP examples were observed using *Wireshark*:

Example 1: Authentication request

Android app request:

```
<stream:stream xmlns='jabber:client' to='dukgo.com'
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' xml:lang='en'>
```

Fake XMPP server response:

```
<?xml version='1.0'?><stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='48429767-2bf1-4fd9-98c8-
ff2d3db0af4c' from='dukgo.com' version='1.0'
xml:lang='en'><stream:features><mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'><mechanism>SCRAM-SHA-1</mechanism><mechanism>DIGEST-
MD5</mechanism></mechanisms></stream:features>
```

Android app request (attempts to authenticate):

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='SCRAM-SHA-
1'>biwsbj1hYmVkcm9pZC42ZWQ4OWY4MCxyPSsneilgXDRwWUklKENSe3pZb2BGTT4meWRNQUFsKyVX<
/auth>
```

Note: Being base64-decoded, this leaks the originating user's ID and the challenge response, which the fake XMPP server could replay to the real server:

```
n,,n=abedroid.6ed89f80,r=+'z)`\4pYI%(CR{zYo`FM>&ydMAAl+%W
```

Example 2: Sending messages in clear-text

Android app request (sends a message in clear-text):

```
<message to='abe1@dukgo.com' id='GXqxF-39' type='chat'><body>?OTRv2? Your
contact is requesting to start an encrypted chat.</body><request
xmlns='urn:xmpp:receipts'/></message>
```

Example 3: XMPP user registration

Since all XMPP messages can be captured this way, this also includes new user registrations, as well as any other XMPP messages:

Client to Server XMPP registration request:

```
<iq to='dukgo.com' id='1xvvk-3186' type='set'><query
xmlns='jabber:iq:register'><username>abedroid3.934ed5db</username><password>+xCY
K5hPty38</password></query></iq>
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Please note that when the same attack is attempted against the iOS app, a connection to the XMPP server is refused. The following error message appears and, additionally, it is not possible for the user to change any settings on the iOS app to connect insecurely:

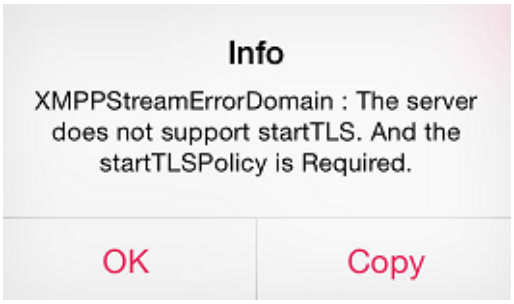

Fig.: An error message when the same attack is attempted on iOS

In order to solve this problem, it is recommended to tunnel XMPP protocol messages over a TLS tunnel. This approach will remove the possibility of tampering with the initial clear-text messages of the XMPP protocol. In addition to this, using pinning[15] on the mobile app will provide protections even in situations where a MitM has a valid root CA and is able to craft valid certificates for the XMPP server's endpoint. Although there is some pinning support on the app at present, the existing checks are done only after the *starttls* XMPP command, which is clearly too late and provides no protection against clear-text XMPP communications.

ZOM-01-006 Android app supports plain-text authentication *(High)*

It was found that the Android app will send user-credentials in clear-text when this is the only method offered by the server. This requires the app to be in its default configuration and the "*Allow Plain Text Auth*" setting cannot be enabled. Since the connection is not wrapped over SSL, an attacker can therefore spoof the replies from the server and pretend that the *starttls* capability is unsupported, as explained in ZOM-01-005. Then *PLAIN* as the only authentication mechanism can be offered. An attacker could leverage this weakness to capture user-credentials and impersonate Zom's users. The Android app will additionally attempt to authenticate all configured accounts at once, hence leaking all such credentials as soon as it attempts to connect.

Please note that, as explained in the introductory description, all examples documented in this finding were verified with the following XMPP setting disabled:

[15] https://www.owasp.org/index.php/Pinning_Cheat_Sheet

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

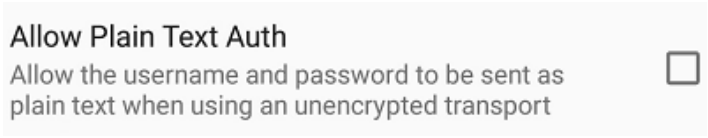

Fig.: Default XMPP setting for "Allow Plain Text Auth" in Android

A fake XMPP server was set up using *prosody*, which offered the following authentication options to clients:

Original XMPP message:

```
<?xml version='1.0'?><stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='75f47ba5-ba1b-463b-92d4-
c25c7a2ee8cc' from='dukgo.com' version='1.0'
xml:lang='en'><stream:features><mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'><mechanism>SCRAM-SHA-1</mechanism><mechanism>DIGEST-
MD5</mechanism></mechanisms><auth xmlns='http://jabber.org/features/iq-
auth'/></stream:features>
```

An attacker with network manipulation capabilities can modify the response of this type, or, alternatively, configure their own server to reply as follows instead:

Crafted XMPP message:

```
<?xml version='1.0'?><stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams' id='75f47ba5-ba1b-463b-92d4-
c25c7a2ee8cc' from='dukgo.com' version='1.0'
xml:lang='en'><stream:features><mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'><mechanism>PLAIN</mechanism><auth xmlns='http://jabber.org/features/iq-
auth'/></stream:features>
```

When this is done, the Android app will send the credentials for all configured accounts in clear-text. During testing, the XMPP requests enumerated below were captured.

Example plain-text XMPP requests:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
mechanism='PLAIN'>AGFiZWRyb2lkMi40OWJiODA4MwBEQkp4M2s5cFo1cFU=</auth>

<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
mechanism='PLAIN'>AGFiZWRyb2lkLjZlZDg5ZjgwAEBhQXdxSGpLaCMrRA==</auth>
```

The above base64-encoded values correspond to the two users that the Android app had configured:

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Command:

```
for i in $(echo 'AGFiZWRyb2lkLjZlZDg5ZjgwAEBhQXdxSGpLaCMrRA==
AGFiZWRyb2lkMi40OWJiODA4MwBEQkp4M2s5cFo1cFU='); do echo $i | base64 -d; echo;
done
```

Output:

```
abedroid.6ed89f80@aAwqHjKh#+D
abedroid2.49bb8083DBJx3k9pZ5pU
```

In addition to extrapolating the mitigations offered under ZOM-01-005 in this scenario, it is recommended to use a digest-based authentication mechanism instead of sending plaintext credentials. Furthermore, the value of the *"Allow Plain Text Auth"* XMPP setting should be respected on Android, with the policy enforced on both the server- and the client-side.

ZOM-01-008 API Server: Django logins available over clear-text HTTP *(High)*

It was found that the default API server is currently configured in a way that allows the Django administration and Django REST framework login pages to be available over clear-text HTTP. An attacker could leverage this weakness to fool an administrator to login over clear-text HTTP without browser warnings. In effect, the admin or RESTful API credentials could be captivated.

This issue can be demonstrated as one navigates with the browser to the following URLs:

Example 1: Django administration area login
http://push.zom.im/admin/login/

Example 2: Django REST framework login
http://push.zom.im/api/api-auth/login/

It is recommended to set up a permanent redirect from port 80 to port 443. This redirect should work for all URLs requested insecurely and the HSTS header[16] should be returned on all pages, consistently being served over TLS. This will significantly reduce the likelihood of the channel downgrade attacks since the HSTS header will instruct the browser to only connect to the server using TLS.

[16] https://www.owasp.org/index.php/HTTP_Strict_Transport_Security

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

ZOM-01-009 iOS app supports plain-text authentication *(Medium)*

It was found that, like the Android app, the iOS app will also send credentials in clear-text when the XMPP server does not provide any other option for authentication. This was tested using a Python script to forward XMPP messages between the mobile app and the server. The severity in the iOS app's case is lower than for its Android counterpart because the iOS app will refuse to authenticate to the XMPP server without the *starttls* command being presented as an option first. This attack therefore requires the iOS user to accept an invalid certificate prompt, which nevertheless seems plausible per explanations provided in ZOM-01-015.

This issue can be verified by intercepting XMPP traffic and modifying XMPP server responses so that *PLAIN* authentication is the only option available:

Intended XMPP message from client to server

```
<?xml version='1.0'?><stream:stream
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' from='dukgo.com'
id='1bace92f-592b-4802-9d4f-6f7b3d9f95bf' xml:lang='en'
xmlns='jabber:client'><stream:features><mechanisms
xmlns='urn:ietf:params:xml:ns:xmpp-sasl'><mechanism>SCRAM-SHA-
1</mechanism><mechanism>PLAIN</mechanism></mechanisms><auth
xmlns='http://jabber.org/features/iq-auth'/><register
xmlns='http://jabber.org/features/iq-register'/></stream:features>
```

Modified Server-to-Client XMPP message (*PLAIN* authentication is the only option)
Note: This step replaces *<mechanism>SCRAM-SHA-1</mechanism>* with *''*

```
<?xml version='1.0'?><stream:stream
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' from='dukgo.com'
id='1bace92f-592b-4802-9d4f-6f7b3d9f95bf' xml:lang='en'
xmlns='jabber:client'><stream:features><mechanisms
xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'><mechanism>PLAIN</mechanism></mechanisms><auth
xmlns='http://jabber.org/features/iq-auth'/><register
xmlns='http://jabber.org/features/iq-register'/></stream:features>
```

The iOS client authenticates with clear-text credentials:

```
<auth xmlns="urn:ietf:params:xml:ns:xmpp-sasl"
mechanism="PLAIN">AGFiZTEAVXVkaDJjUCtBSzB5SktZPQ==</auth>
```

In the base64-decoded form, the issue is as follows:

Command:

```
echo 'AGFiZTEAVXVkaDJjUCtBSzB5SktZPQ==' | base64 -d
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Output:
```
abe1Uudh2cP+AK0yJKY=
```

The server accepts the *PLAIN* authentication attempt:
```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'></success>
```

It is recommended to disable plain-text authentication by default. This mechanism should exclusively be available if, for some reason, it is deemed a valid mode of authentication at the present state of the project. If it is decided that plain-text authentication is a feature that should remain in operation, then, at a very minimum, the user should be prompted with warning prior to sending the credentials to the server insecurely.

ZOM-01-010 iOS: Permanent DoS via unknown contact message *(High)*

It was found that the iOS app will become unusable when an unknown contact sends a normal message to an iOS user. This makes the XMPP server send contact “presence” messages containing this non-contact, which crashes the app immediately upon opening,, hence making it impossible to use. Please note that other messages from the XMPP server also lead to an app crashing

Example 1: XMPP message from a non-contact

In this scenario an attacker using a non-contact account attempts to start an OTR conversation with the iOS user. This pattern will make the XMPP server send “presence” messages with this account from then on:

Initially the server sends a subscription “presence” message for the non-contact:

XMPP Message:
```
<presence type='subscribe' to='abe1@dukgo.com' from='friendlykitten@null.pm'/>
```

Next, the non-contact attempts to establish an OTR connection:

XMPP Message:
```
<message id='purpleadaf566b' type='chat' to='abe1@dukgo.com'
from='friendlykitten@null.pm/f0175ace-4cca-4cad-b637-2682446b20b7'><active
xmlns='http://jabber.org/protocol/chatstates'/><body>?OTRv23?
friendlykitten@null.pm/ has requested an Off-the-Record private conversation
&lt;https://otr.cypherpunks.ca/&gt;.  However, you do not have a plugin to
support that.
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
See https://otr.cypherpunks.ca/ for more information.</body><html
xmlns='http://jabber.org/protocol/xhtml-im'><body
xmlns='http://www.w3.org/1999/xhtml'><p>?OTRv23?
<span style='font-weight: bold;'>friendlykitten@null.pm/</span> has requested an
<a href='https://otr.cypherpunks.ca/'>Off-the-Record private conversation</a>.
However, you do not have a plugin to support that.
See <a href='https://otr.cypherpunks.ca/'>https://otr.cypherpunks.ca/</a> for
more information.</p></body></html></message>
```

From that moment onwards, the XMPP server will provide an indication of “presence” to the iOS app, which will effectively make it crash:

XMPP Message:

```
<presence from='abe1@dukgo.com/zom40991'><x xmlns='vcard-
temp:x:update'><photo/></x><c hash='sha-1' ver='azVhv4kC6GHYtDl7jT//llLIhIg='
node='https://github.com/robbiehanson/XMPPFramework'
xmlns='http://jabber.org/protocol/caps'/></presence><presence type='unavailable'
to='abe1@dukgo.com/zom40991' from='abedroid.6ed89f80@dukgo.com'/><presence
from='friendlykitten@null.pm' type='subscribe'/>
```

It was confirmed that removing the unknown contact from the “presence” XMPP message above would prevent the crash in full.

Example 2: Other XMPP messages that cause crashes

Whenever the XMPP server sends a message like the following to the iOS app, it will also crash. A malicious Zom user can send a message of this kind to any iOS app user to permanently crash the app for the targeted user:

```
<iq id='8F5F36D8-D4F4-4111-A4C7-18DED6313087' type='result'
to='abe1@dukgo.com/zom40991'
from='group33b668e4@conference.dukgo.com/abedroid.6ed89f80'><query
node='http://www.igniterealtime.org/projects/smack#NfJ3flI83zSdUDzCEICtbypursw='
xmlns='http://jabber.org/protocol/disco#info'><identity type='pc' name='Smack'
category='client'/><feature
var='http://jabber.org/protocol/bytestreams'/><feature
var='jabber:iq:privacy'/><feature var='urn:xmpp:ping'/><feature
var='http://jabber.org/protocol/commands'/><feature
var='jabber:iq:version'/><feature var='jabber:iq:last'/><feature
var='http://jabber.org/protocol/xdata-validate'/><feature
var='http://jabber.org/protocol/xhtml-im'/><feature var='vcard-temp'/><feature
var='urn:xmpp:receipts'/><feature var='urn:xmpp:time'/><feature
var='http://jabber.org/protocol/xdata-layout'/><feature
var='http://jabber.org/protocol/muc'/><feature
var='http://jabber.org/protocol/disco#items'/><feature
var='http://jabber.org/protocol/disco#info'/><feature
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
var='http://jabber.org/protocol/caps'/><feature
var='jabber:x:data'/></query></iq>
```

It is recommended to improve the exception handling of edge cases so that the legitimate application users cannot be easily disrupted by the malicious ones.

ZOM-01-011 Android: DoS via crafted message with high size value *(Medium)*

It was found that the Android app crashes due to a failed allocation when a client sends a malicious message. The hostile message contains a value which is used to determine how much memory should be allocated. If this number is too high, the application simply crashes. The following listings show the XMPP message and the hex representation of the message.

XMPP Message:

```
<message type='chat' id='purple3a4f7511'
to='sdflkdsfg.599e6cab@dukgo.com/ChatSecureZom-9a13a0e1'>
            <body>?OTR:AAIDAAAAfLvhZv0gYGrFZg==</body>
</message>
```

Message (hex):

```
00000000  00 02 03 00 00 00 7c bb  e1 66 fd 20 60 6a c5 66  |......|..f. `j.f|
```

The last four bytes are a big endian value and decode to **1617610086**. This number is almost equal to the value in the failed allocation error message.

Backtrace:

```
04-26 00:04:34.478: E/AndroidRuntime(4258): FATAL EXCEPTION: Smack-Cached
Executor 3 (0)
04-26 00:04:34.478: E/AndroidRuntime(4258): Process: im.zom.messenger, PID: 4258
04-26 00:04:34.478: E/AndroidRuntime(4258): java.lang.OutOfMemoryError: Failed
to allocate a 1617610098 byte allocation with 4194304 free bytes and 368MB until
OOM
04-26 00:04:34.478: E/AndroidRuntime(4258):    at
net.java.otr4j.io.OtrInputStream.readData(OtrInputStream.java:70)
04-26 00:04:34.478: E/AndroidRuntime(4258):    at
net.java.otr4j.io.OtrInputStream.readBigInt(OtrInputStream.java:64)
04-26 00:04:34.478: E/AndroidRuntime(4258):    at
net.java.otr4j.io.OtrInputStream.readDHPublicKey(OtrInputStream.java:101)
04-26 00:04:34.478: E/AndroidRuntime(4258):    at
net.java.otr4j.io.SerializationUtils.toMessage(SerializationUtils.java:291)
```

The bug is located in the *readData* function, wherein first a number is read and then it is used to allocate memory.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected File:

```
/Zom-Android-15.0.1-BETA-
3/otr4jandroid/src/main/java/net/java/otr4j/io/OtrInputStream.java
```

Affected Code:

```
public byte[] readData() throws IOException {
       int dataLen = readNumber(DATA_LEN);
       byte[] b = new byte[dataLen];
       read(b);
       return b;
}
```

It is recommended to check user-defined values used for critical operations, like memory allocation for sanity first. If the value is not within reasonable bounds the message should be dropped and ignored.

ZOM-01-012 iOS and Android: Automated linkification of phone URLs *(Medium)*

It was found that the iOS and Android apps will make telephone URLs tappable. A malicious Zom user could leverage this weakness to fool Zom users and take advantage of the apps dedicated for ringing premium phone numbers. The severity of this issue is reduced because two user-taps are required.

On iOS, a user can send a telephone URL, typing text like the following: ***tel://1234567890***. The app will detect this and identify it as a phone number, which means that it will make it tappable:

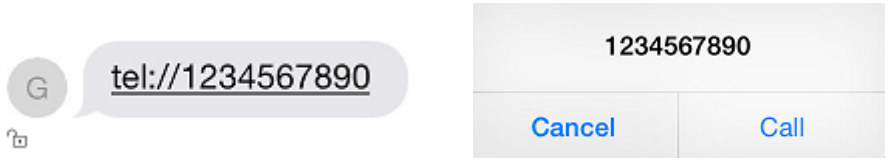

Fig.: Tapping on linkified phone numbers rings them

On Android it is enough to put a valid phone number somewhere in the text like e.g. **+49176123456789**

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

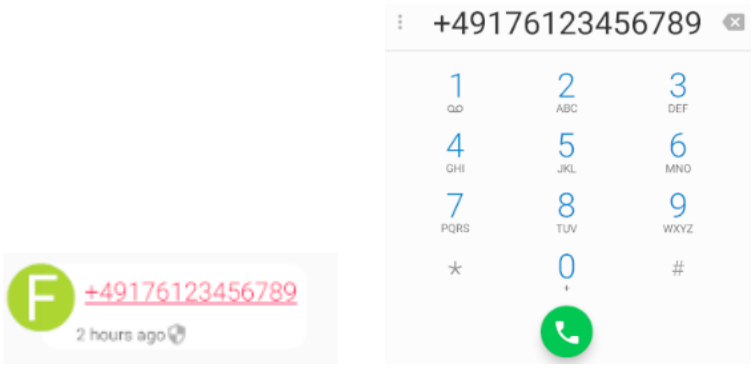

Fig.: Linkification on Android

It is recommended to disable automated linking of phone numbers[17] in order to make the accidental ringing less likely.

ZOM-01-015 iOS and Android: Unclear messages during TLS MitM *(Medium)*

The iOS and Android apps provide users with unclear messages when self-signed or otherwise invalid TLS certificates are offered by the XMPP server. In addition to this, attackers with greater resources such as nation states will be able to forge valid certificates for which the warnings will be even more difficult to distinguish for average users on iOS (unless the user checks the signature manually) and impossible to detect on Android (where certificates signed by trusted CAs are accepted transparently). Please note that when a MitM uses an invalid certificate, Android users will be constantly prompted to accept the certificate until they in fact accept it with the "*Always*" option. A malicious attacker could leverage this weakness to attempt to MitM XMPP connections and hope for the user to click through the security warnings.

This issue can be broken down as follows:

Issue 1: iOS: Minimal difference between valid vs. invalid certificate messages

When the TLS certificate for the XMPP server changes on iOS, the user will be prompted with the message presented below. This response will depend on whether the certificate is trusted or not:

[17] https://developer.apple.com/library/ios/featuredarticles/iPhoneURLScheme...neLinks/PhoneLinks.html

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

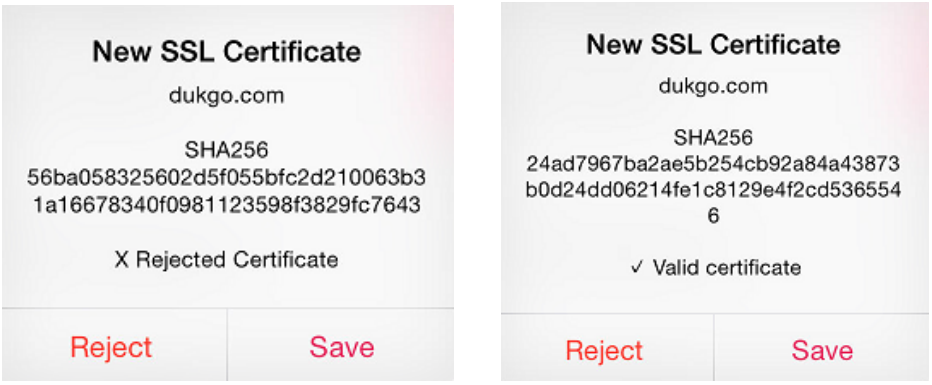

Fig.: Minimal difference between valid vs. invalid certificates on iOS

If the user taps on the "*Save*" button, the SSL certificate will be saved as trusted by the app and the user will never be prompted again when this certificate is used.

There is a slight difference for the case of an attacker using self-signed certificates. This is mainly because it seems unlikely that average users will pay attention to the minimal difference between "*Rejected*" and "*Valid*" on the messages shown. Moreover, it is even less probably that any user will check the *sha256* signature manually.

Moving on to the case of an attacker using trusted certificates, it needs to be assumed that average users will only notice a "Valid certificate" prompt without checking up on which app "*does not work*".

In both cases, the users are neither warned about the implications of tapping through such warnings, nor should they be expected to know what they really mean.

If the user is prompted for the messages on iOS, this means that the certificate is different from what it used to be. In turn, it points to the user likely being under attack. Clicking through such warnings will effectively leak user credentials, messages, contact information and other data to the attacker intercepting network communications.

Issue 2: Android: Constant user prompts to accept invalid certificates

On Android, the user will be prompted continuously to accept the certificate. The scenario in which the user taps on the "*Once*" option seems unlikely, especially given that the user will get one pop-up per minute by default. This occurs because the Android app opens the dialog every time it tries to connect to the XMPP server (i.e. even if the previous dialog has not been closed yet). Therefore it seems plausible that the user will

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

eventually tap on the "*Always*" button out of frustration. Please note that these user-dialogs will appear on the phone even if the user is not using the Zom app directly. The reason behind this is that the Zom app is running in the background on Android:

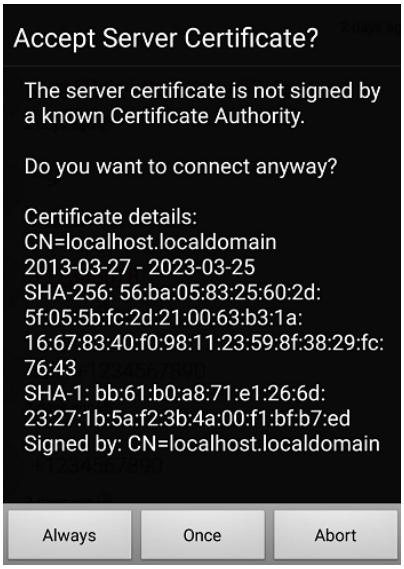

Fig.: Continuous self-signed certificate prompt on Android

Perhaps even more confusing is the case where the attacker uses a valid certificate for other domains, for example *google.com*. Once again, this will also be a continuous prompt until the user accepts the certificate:

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

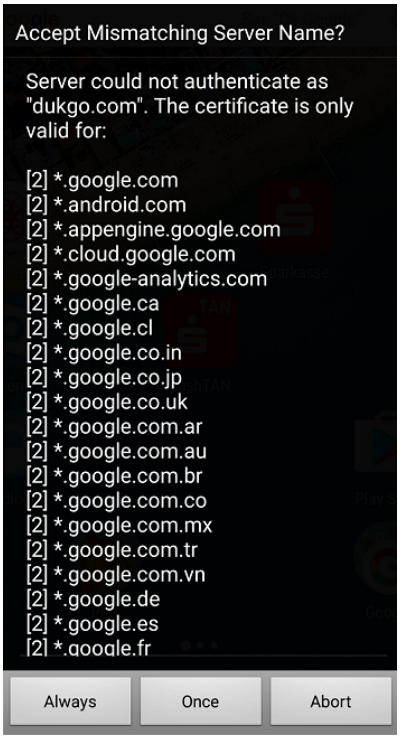

Fig.: Prompt for a valid certificate for a different domain

In the above scenarios, if the user taps on "*Abort*" or "*Once*", they will just keep getting these dialogs every minute. It will perhaps be more likely that the user will have a number of dialogs to tap through when they look at the phone, since at a rate of one dialog per minute many dialogs will pop-up while the user is not even using the phone. The attacker only needs one tap on "*Always*" for the certificate to be trusted indefinitely. In addition, if the certificate is valid, no warning will be shown to the user and the Android app will simply trust the certificate transparently. Note: This behavior is different in iOS.

It is recommended to make the error messages clearer to the average users, for example by stating that:

> *"The SSL certificate has changed and this could be an attack! Accepting this certificate could leak your credentials, messages and contact information to a third party"*

Elaborating on the defense strategies further, there should be obvious differences in messages for differing certificates. One idea would be to use the color red for warnings or simply issue more explicit and less nuanced warnings. It is also important to provide

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

users with an option to ignore a given certificate in order to avoid constant prompts. Again it needs to be ensured that prompts will not appear again if the previous prompt has not been dealt with.

ZOM-01-016 iOS: ChatSecure UserVoice Pod uses clear-text HTTP *(Medium)*

It was found that the *ChatSecure UserVoice Pod* webview attempts to embed a CSS file over clear-text HTTP. An attacker with the ability to manipulate network communications could leverage this weakness to make the phone send HTTP requests to arbitrary Internet websites, as well as the web server listening on the phone. Please note that since the attacker controls the full CSS file, another attack possibility could be to extract Forum information using CSS selectors, CSS Unicode fonts and other exfiltration vectors.

This issue can be verified when the iOS user visits a knowledge base article, the web view will then send an HTTP request as follows:

Request:

```
GET http://cdn.uservoice.com/stylesheets/vendor/typeset.css HTTP/1.1
[...]
```

Although the legitimate web server replies with a "*Page not found*" error, the plain-text communications model needs to be considered, meaning that the attacker can change this to something like the following:

Forged Response:

```
HTTP/1.1 302 Found
Location: http://t.7-a.org
[...]
```

The webview was observed to follow the redirect to *t.7-a.org* and then to *www.youtube.com* and *m.youtube.com*.

This issue is located on the *UVArticleViewController*, which contains a explicit clear-text HTTP reference to the stylesheet:

https://github.com/ChatSecure/uservoice-iphone-sdk/blob/a6cf679b3fb116d31fba47cd9ed6cdc64c327c7b/Classes/UVArticleViewController.m#L39

Affected Code:

```
NSString *html = [NSString stringWithFormat:@"<html><head><link
rel=\"stylesheet\" type=\"text/css\"
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
href=\"http://cdn.uservoice.com/stylesheets/vendor/typeset.css\"/><style>a
{ color: %@; }</style></head><body class=\"typeset\" style=\"font-family:
HelveticaNeue; margin: 1em; font-size: 15px\"><h5 style='font-weight: normal;
color: #999; font-size: 13px'>%@</h5><h3 style='margin-top: 10px; margin-bottom:
20px; font-size: 18px; font-family: HelveticaNeue-Medium; font-weight: normal;
line-height: 1.3'>%@</h3>%@</body></html>", linkColor, section,
_article.question, _article.answerHTML];
```

It is recommended to ensure that all source code and *Pods* in use access resources over TLS to avoid these issues.

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.

ZOM-01-007 XMPP Server supports plain-text authentication *(Low)*

It was found that, like the Android (ZOM-01-006) and iOS (ZOM-01-009) apps, the XMPP server also supports plain-text authentication. Although this is only after the *starttls* command was sent, the *starttls* command can be stripped against the Android app by default (ZOM-01-005). Alternatively, this attack requires the iOS or Android user to accept an invalid certificate prompt when the attacker uses TLS. The fact that the server also supports plain-text authentication makes it easier for a MitM attacker to simply forward the plain-text authentication requests that come from the apps. As a result, capturing user-credentials becomes less of a challenge.

The plain-text authentication reply, which has been forwarded, can perhaps be best illustrated with an XMPP authentication sequence:

Step 1: The client sends a new XMPP message

XMPP Request:

```
<stream:stream xmlns='jabber:client' to='dukgo.com'
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' xml:lang='en'>
```

Step 2: The server returns its capabilities

Please note how *PLAIN* authentication is offered by the server as an option.

XMPP Response:

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
<?xml version='1.0'?><stream:stream
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' from='dukgo.com'
id='d838b61d-de25-485c-8f65-867fef5d1cfe' xml:lang='en'
xmlns='jabber:client'><stream:features><mechanisms
xmlns='urn:ietf:params:xml:ns:xmpp-sasl'><mechanism>SCRAM-SHA-
1</mechanism><mechanism>PLAIN</mechanism></mechanisms><auth
xmlns='http://jabber.org/features/iq-auth'/><register
xmlns='http://jabber.org/features/iq-register'/></stream:features>
```

Step 3: The MitM removes all authentication capabilities that are not *PLAIN*

The attacker removes "*<mechanism>SCRAM-SHA-1</mechanism>*", which leaves the app with *PLAIN* as the only option to log in. The client sees the following results:

Crafted XMPP Response:

```
<?xml version='1.0'?><stream:stream
xmlns:stream='http://etherx.jabber.org/streams' version='1.0' from='dukgo.com'
id='d838b61d-de25-485c-8f65-867fef5d1cfe' xml:lang='en'
xmlns='jabber:client'><stream:features><mechanisms
xmlns='urn:ietf:params:xml:ns:xmpp-
sasl'><mechanism>PLAIN</mechanism></mechanisms><auth
xmlns='http://jabber.org/features/iq-auth'/><register
xmlns='http://jabber.org/features/iq-register'/></stream:features>
```

Step 4: The client authenticates using *PLAIN* authentication

The MitM attacker can forward this request without any modification because the XMPP server also supports *PLAIN* authentication. Otherwise the attacker would need to craft a valid *SCRAM-SHA1* authentication request when seeking to log in.

XMPP Request:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
mechanism='PLAIN'>AGFiZWRyb2lkMy45MzRlZDVkYgAreENZSzVoUHR5Mzg=</auth>
```

Step 5: The server confirms that the authentication attempt was successful

At this stage, it is confirmed that the server supports *PLAIN* authentication:

XMPP Response:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'></success>
```

This issue is less severe than its counterpart affecting the mobile apps accepting *PLAIN* authentication because the server must receive the *PLAIN* credentials when users register. The only issue here is that this makes it easier to exploit other findings in this

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

report (i.e. simply replaying the *PLAIN* authentication requests from the apps against the server). Having said that, the Zom development team should consider to remove *PLAIN* authentication from the server as an additional layer of security after the main findings in this report have been addressed.

ZOM-01-014 API: Assignment weaknesses on the device endpoint *(Info)*

It was found that the device endpoint on the *Push Server* API allows users to edit almost all fields on a previously created device. This might allow malicious users to benefit from the *registration_ids* or device IDs of the users' devices, even though the device itself remains under the same user. This issue is limited in impact because the owner could not be altered during testing.

A normal API request looks like this:

Request:

```
PUT https://push.zom.im/api/v1/device/apns/8c304fc4-95b1-4d20-adca-fdfcab34f1f1/
HTTP/1.1
Authorization: Token 68d89c02e2618259c3e6ec9745b5559e9f8dac58
[...]

{"device_id":"8c304fc4-95b1-4d20-adca-
fdfcab34f1f1","registration_id":"2f6c0f8fae517ba4ad350fb5d525785d57bbb1c4c224f36
f466ed0a23dd3e4cc"}
```

Response:

```
{"name":null,"id":"8c304fc4-95b1-4d20-adca-
fdfcab34f1f1","active":true,"registration_id":"2f6c0f8fae517ba4ad350fb5d525785d5
7bbb1c4c224f36f466ed0a23dd3e4cc","device_id":"8c304fc4-95b1-4d20-adca-
fdfcab34f1f1","date_created":"2016-04-20T23:47:04.041884Z"}
```

It was found that it is possible to update most of the fields for this model, beyond the seemingly intended functionality. This is demonstrated in the following example:

Request:

```
PUT https://push.zom.im/api/v1/device/apns/8c304fc4-95b1-4d20-adca-fdfcab34f1f1/
HTTP/1.1
Authorization: Token 68d89c02e2618259c3e6ec9745b5559e9f8dac58
[...]


{"name":"Ein Test","id":"8c304fc4-95b1-4d20-adca-
fdfcab34f1f1","active":true,"registration_id":"fake_registration_id","device_id"
:"fake_device_id","date_created":"1900-04-
20T23:47:04.041884Z","owner":"04D49550-1234-1234-8A24-F16238"}
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Response:

```
HTTP/1.1 200 OK
[...]

{"name":"Ein Test","id":"8c304fc4-95b1-4d20-adca-
fdfcab34f1f1","active":true,"registration_id":"fake_registration_id","device_id"
:"fake_device_id","date_created":"2016-04-20T23:47:04.041884Z"}
```

It is recommended to review the business need for letting users modify fields such as the registration and device IDs. If possible, the amount of fields that users can update should be restricted in the relevant controllers.

Conclusion

According to the five members of the Cure53 testing team this eleven days-long penetration test and source code audit of the Zom application has left an overall positive impression. The generally good outcome should be underlined, even though sixteen security issues have been found during this security assessment of the Zom app in spring 2016. Regarding technical details, the starting point for this test was a specific focus on mobile applications, signifying coverage of both the Android and iOS versions of the Zom application. In terms of vulnerability patterns identified across the fourteen issues and two general weaknesses, the main spot is occupied by the problems stemming from insecure connections' usage, which need to be monitored in the future. Equal attention should be placed on the API, which allowed the HTTP requests.

Given the fair complexity of the application, a second round of tests is highly recommended once the fixes to the reported findings are crafted and deployed. Additionally, in order to avoid the expansion of the attack surface, the abovementioned types of issues should be covered by automated tests of the upcoming versions. At the same time the limited number of severe problems paired with the extremely productive exchanges and workflow with regard to cooperation and communication with the Zom team can be seen as an indicator for the application being on the right track towards meeting its security goals.

Cure53 would like to thank Nathan Freitas and the entire Zom Team for their excellent project coordination, support and assistance, both before and during this assignment. We would like to further express our gratitude to the Open Technology Fund in Washington D.C., USA, for generously funding this and other penetration test projects and enabling us to publish the results.