Fine penetration tests for fine websites

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

Pentest-Report GreatFire FreeBooks App 08.2017

Cure53, Dr.-Ing. M. Heiderich, MSc. N. Krein, BSc. D. Weißer, Dipl.-Ing. A. Aranguren, N. Kobeissi, M. Wege

Index

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

Introduction

This report documents the findings of a penetration test against the GreatFire FreeBooks Android application, its proxy library, and the connected backend servers. The project was carried out by Cure53 in mid-August 2017 and led to the discovery of eighteen security-relevant findings.

As for the resources, a time budget allocated to this assignment entailed eight days, while personnel comprised five members of the Cure53 team. It was agreed that the investigations will be grounded in a mix-method approach. More specifically, the team engaged in classic source code audit, as well as performed assisted dynamic analysis of the sources with the aid of the Android and proxy provided by the GreatFire team. Verification of the backend server's security was performed by acquiring SSH access to two live systems, while root-access was also granted in order to enable full coverage.

It should be noted that the Cure53 team was asked to focus on the Android application in a very specific realm, namely in terms of the project keeping its security and privacy promises. Another key area was to verify whether the connected proxy setup safeguards anonymity of the users. The backend servers also received considerable attention, especially in the realms of server hardening and classic configuration reviews.

As already noted, the test yielded a total of eighteen discoveries, which were categorized into a main class of nine vulnerabilities, and a group of also nine general weaknesses. Importantly, none of the issues were of "High" or "Critical" level in terms of severity. In other words, the unveiled problems and flaws do not pose urgent risks. The majority of spotted issues revolve around classic Denial-of-Service, wherein users cannot rely on the FreeBooks application for normal use. In general, the tests revealed that GreatFire is quite secure and a low number of actually concerning vulnerabilities must be addressed in due course.

The report will now shed light on the scope in more detail. Next, each spotted issue will be described, with special attention on proposed mitigation and fixing advice. In the final section, some main threads are foregrounded to equip the maintainers of the GreatFire FreeBooks project with a broader verdict on the security situation at the application at present.

Scope

- **Module 1: Tests against FreeBooks Android App**
  - APK and sources were made available to Cure53.
- **Module 2: Tests & Audits against FreeBooks Proxy Library**
  - Source code was made available to Cure53.
- **Module 3: Tests & Audits against Server Customization**
  - Access to an example server was given to Cure53.

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. *GF-01-001*) for the purpose of facilitating any future follow-up correspondence.

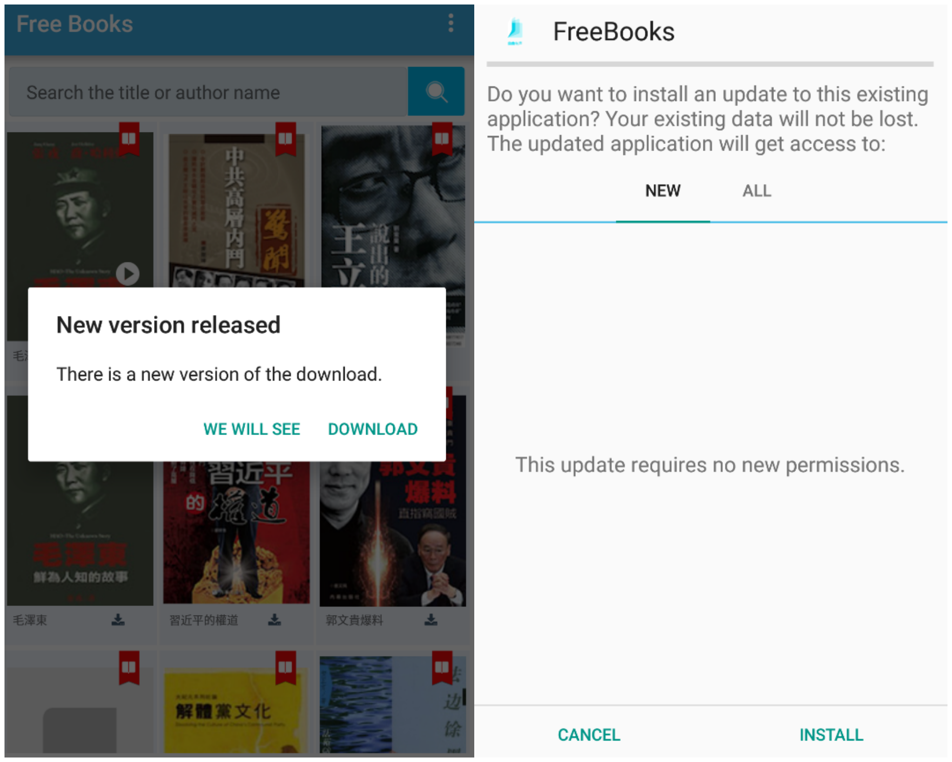
GF-01-003 Android: Possible updater RCE via racecondition on SD Card *(Medium)*

It was found that the GreatFire Free Books application downloads upgrade APKs to a predictable SD Card location. The downloaded upgrade file is first saved with a temporary filename (*FreeBooks2.0-auto.apk.tmp*). Once the download is complete, the file is renamed to *FreeBooks2.0-auto.apk* and sent to the Android installer. Since every application with SD Card permissions can read and write to the SD Card, a malicious app could leverage this weakness to modify the APK and enforce an installation of a malicious upgrade.

Another possible attack scenario would be to launch the Android installer with a different app that looks like the Free Books app while the download is still running in the background. This way the user would have no way of telling if the upgrade is malicious or not on the grounds of the UI.

This issue can be verified when the application is opened. If an update is available, an upgrade dialog is shown. When the user taps on the *Download* button, the download initiates and, ultimately, the Android installer upgrade screen will be shown.

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



New version released

There is a new version of the download.

WE WILL SEE DOWNLOAD

Do you want to install an update to this existing application? Your existing data will not be lost. The updated application will get access to:

This update requires no new permissions.

Fig.: Upgrade dialog (left) and upgrade process (right)

Under the hood, when an update is available and the user taps on the *Download* button, there are several steps that ensue.

- The app downloads the *upgrade* file as *FreeBooks2.0-auto.apk.tmp/*
- When the download completes, the file is renamed to *FreeBooks2.0-auto.apk*
- The Android installer is launched with the upgraded APK.

As this occurs on the SD Card, malicious apps can read and write files, as well as launch the Android installer at any time during this process. The Impact of this issue can vary from preventing updates (i.e. corrupting the downloaded *upgrade* files), to launching the installer with a malicious upgrade or even app.

The *Upgrade* APK location can be consulted below.

File Path:
/storage/emulated/0/Android/data/org.greatfire.freebook/files/free/FreeBooks2.0-auto.apk

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

As long as there is sufficient space available on the device, it is strongly recommended to download APKs into the protected app storage on Android. In other words, the SD Card should be avoided. What is more, the file could be given a long and random name, which is then handed over to the Android Installer. This would need to occur after the permissions have been set up and immediately after the APK download is complete[1][2]. This would fully eliminate all SD Card manipulation attacks. However, it is still recommended to delete the APK when the signature check fails to protect against malicious upgrades, even when app storage for APK downloads is available.

GF-01-006 Backend: Privilege Escalation via Cron-Restart Script *(Medium)*

While reviewing possible configuration issues on the backend servers at 43.249.37.59 and 192.168.131.231, it was noticed that unsafe usage of *cron* scripts is present, thus resulting in a potential privilege escalation vector leading to *root*. This was noticed in the following script which is run whenever a reboot happens.

Contents of */etc/cron.d/nginx*:

```
[...]
@reboot       root   cd /home/ubuntu/CollateralFreedomProxy/backend/nginx &&
./restart.sh
[...]
```

As one can observe here, the *restart.sh* script is run with *root* privileges on each reboot, while it is actually still writable by the *ubuntu* user.

```
-rwxrwxr-x 1 ubuntu ubuntu 632 Jul 17 19:59
/home/ubuntu/CollateralFreedomProxy/backend/nginx/restart.sh
```

This means that an attacker -- upon having compromised the *ubuntu* user -- can simply edit the restart script to contain a little backdoor. S/he could then just wait for the mechanism to get triggered as soon as a reboot takes place. With the presence of GF-01-007, attackers can take advantage of more simply attainable options of course, but this issue should be fixed regardless. As a general rule of thumb, it is recommended to make sure that all scripts that are eventually run as *root* can be modified exclusively by *root*. This prevents the *root* user from accidentally running unauthorized commands.

[1] http://stackoverflow.com/questions/36755301/how-to-run-install-app-in-internal-files-dir
[2] http://stackoverflow.com/questions/36907093/androidsaving-apk-files-in-internal-s...ying-get-an-parsing

GF-01-007 Backend: Sudoers Group does not require a Password *(Medium)*

Another configuration-related issue that concerns both backend servers at 43.249.37.59 and 192.168.131.231 is based on the fact that any user who is a member of the *sudoers* group can issue commands as *root* without providing further authentication. This can be seen with the following configuration file.

Contents of */etc/sudoers*:

```
[...]
%sudo  ALL=(ALL:ALL) NOPASSWD:ALL
[...]
```

As such, a compromised *ubuntu* user automatically possesses the same abilities as the *root* user itself, just from using *sudo* to run arbitrary commands as *root*. It is highly recommended to employ additional password authentication or, better yet, use a whitelist of the allowed commands that the *ubuntu* user is permitted to run as *root*.

GF-01-009 Android: User DoS and Disruption via *MediaPlayerService* *(Medium)*

It was found possible to reliably crash the GreatFire FreeBooks app by sending a crafted intent against the *MediaPlayerService*. A malicious application could leverage this weakness to continuously send intents from the background, and hence reliably prevent users from actually making use of the app. In addition to this, malicious apps can also pause and resume audiobook playback via alternatively crafted intents sent to the exported *MediaPlayerService.* Please note that while the crashing scenario always works, no matter of what the user is doing, there is a requirement of having an audiobook open for the resume and play audiobook disruption to succeed.

Scenario 1: Consistent app crash

This attack is successful without any pre-requirements and will be accomplished regardless of what the user is doing (i.e. listening to an audiobook or not). The process can be verified by running the following ADB command while the app is open on an Android emulator or device.

ADB Command:

```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.media.MediaPlayerService"
```

Corresponding logcat stack trace:

```
08-21 05:46:34.016 E/AndroidRuntime( 2819): FATAL EXCEPTION: main
08-21 05:46:34.016 E/AndroidRuntime( 2819): Process: org.greatfire.freebook,
PID: 2819
```

```
08-21 05:46:34.016 E/AndroidRuntime( 2819): java.lang.RuntimeException: Unable
to start service co.mobiwise.libraryradio.media.MediaPlayerService@a855dd6 with
Intent
{ cmp=org.greatfire.freebook/co.mobiwise.libraryradio.media.MediaPlayerService }
: java.lang.NullPointerException: Attempt to invoke virtual method 'boolean
java.lang.String.equals(java.lang.Object)' on a null object reference
[...]
08-21 05:46:34.016 E/AndroidRuntime( 2819): Caused by:
java.lang.NullPointerException: Attempt to invoke virtual method 'boolean
java.lang.String.equals(java.lang.Object)' on a null object reference
08-21 05:46:34.016 E/AndroidRuntime( 2819):    at
co.mobiwise.libraryradio.media.MediaPlayerService.onStartCommand(MediaPlayerServ
ice.java:136)
08-21 05:46:34.016 E/AndroidRuntime( 2819):    at
android.app.ActivityThread.handleServiceArgs(ActivityThread.java:3010)
08-21 05:46:34.016 E/AndroidRuntime( 2819):    ... 9 more
08-21 05:46:34.017 W/ActivityManager(  614):   Force finishing activity
org.greatfire.freebook/.BookShelf
```

The root cause for this particular crash is that the app fails to check if an intent action has been sent at all, so that when the intent action is empty (*null)*, the check on line 136 results in a stack trace on *MediaPlayerService.java*.

File:
FreeBooks-master/FreeBooks/radio/src/main/java/co/mobiwise/libraryradio/media/MediaPlayerService.java

Affected Code:

```
132     public int onStartCommand(Intent intent, int flags, int startId) {
133
134         String action = intent.getAction();
135
136         if(action.equals(NOTIFICATION_INTENT_CANCEL)){
```

Scenario 2: User disruption via Audiobook playback manipulation

In the second variant, when a user needs to have an audiobook open, it is possible for a malicious app to control the behavior of the player in accordance with the following examples.

Example 1: Play / Pause

This plays the audiobook if it is paused, or, in reverse, pauses it when it is being played.

ADB Command:
```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.media.MediaPlayerService" -a
"co.mobiwise.library.notification.media.INTENT_PLAYPAUSE"
```

Example 2: Cancel

From the user's UI perspective, this variant is similar to *pause* above, as it will pause the audio if it is playing.

ADB Command:
```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.media.MediaPlayerService" -a
"co.mobiwise.library.notification.media.INTENT_CANCEL"
```

The root cause of this issue is that the MediaPlayerService is exported to third-party apps via intent-filter declaration implicitly.

File:
AndroidManifest.xml

Affected Code:
```
<service android:name="co.mobiwise.libraryradio.media.MediaPlayerService">
   <intent-filter>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_PLAYPAUSE"/>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_OPENPLAYER"/>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_CANCEL"/>
   </intent-filter>
</service>
```

It is recommended to review the business need to expose this service to third-party apps and, if possible avoid exporting it. This could be accomplished by setting the *android:exported="false"* attribute on the affected service within the *AndroidManifest.xml* file.

Proposed Fix:
```
<service android:name="co.mobiwise.libraryradio.media.MediaPlayerService"
android:exported="false">
```

What is more, appropriate exception handling should be implemented on *MediaPlayerService.java*, so that line 136 does not crash the entire app when no intent action is provided.

GF-01-010 Android: User DoS and Disruption via *RadioPlayerService* (*Medium*)

It was found the issues affecting *MediaPlayerService* and described under GF-01-009, also apply to *RadioPlayerService.* In a similar manner, this allows malicious apps to crash or disrupt FreeBooks users, either by crashing the app at any time, or by *Pausing/Resuming* audio playback when a radio station is opened. The scenarios depicting replications of these problems can be found below.

Scenario 1: Consistent app crash.

Crashing the application in a consistent manner can be accomplished by running the following ADB command.

ADB Command:
```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.radio.RadioPlayerService"
```

Corresponding logcat crash:
```
08-21 06:34:35.008 D/AndroidRuntime( 3607): Shutting down VM
08-21 06:34:35.008 E/AndroidRuntime( 3607): FATAL EXCEPTION: main
08-21 06:34:35.008 E/AndroidRuntime( 3607): Process: org.greatfire.freebook,
PID: 3607
08-21 06:34:35.008 E/AndroidRuntime( 3607): java.lang.RuntimeException: Unable
to start service co.mobiwise.libraryradio.radio.RadioPlayerService@57b10ac with
Intent
{ cmp=org.greatfire.freebook/co.mobiwise.libraryradio.radio.RadioPlayerService }
: java.lang.NullPointerException: Attempt to invoke virtual method 'boolean
java.lang.String.equals(java.lang.Object)' on a null object reference
[...]
08-21 06:34:35.008 E/AndroidRuntime( 3607): Caused by:
java.lang.NullPointerException: Attempt to invoke virtual method 'boolean
java.lang.String.equals(java.lang.Object)' on a null object reference
08-21 06:34:35.008 E/AndroidRuntime( 3607):    at
co.mobiwise.libraryradio.radio.RadioPlayerService.onStartCommand(RadioPlayerServ
ice.java:179)
```

Root cause analysis:

This crash occurs because no intent action was provided, which makes the following code throw an exception that crashes the app. Essentially, on line 173 the action is set to null because no intent action was provided. This, in turn, causes line 179 to throw an exception as the *action* variable is *null* and therefore lacks the invoked *equals* method.

File:
FreeBooks-master/FreeBooks/radio/src/main/java/co/mobiwise/libraryradio/radio/RadioPlayerService.java

Affected Code:
```
171     public int onStartCommand(Intent intent, int flags, int startId) {
172
173         String action = intent.getAction();
174
175         /**
176          * If cancel clicked on notification, then set state to
177          * IDLE, stop player and cancel notification
178          */
179         if (action.equals(NOTIFICATION_INTENT_CANCEL)) {
180             if (isPlaying()) {
181                 isClosedFromNotification = true;
182                 stop();
183             }
184             if (mNotificationManager != null)
185                 mNotificationManager.cancel(NOTIFICATION_ID);
186         }
187         /**
188          * If play/pause action clicked on notification,
189          * Check player state and stop/play streaming.
190          */
191         else if (action.equals(NOTIFICATION_INTENT_PLAY_PAUSE)) {
192             if (isPlaying())
193                 stop();
194             else if (mRadioUrl != null)
195                 play(mRadioUrl);
196
197         }
198
199         return START_NOT_STICKY;
200     }
```

Scenario 2: User disruption via *Play/Resume* actions

This service can also be invoked by malicious apps seeking to disrupt users. Two examples of relevant behaviors are provided next.

Example 1: *Play / Pause*

In this approach, an audiobook is played if it is paused, and, coverly, it gets paused when being played.

ADB Command:
```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.media.RadioPlayerService" -a
"co.mobiwise.library.notification.radio.INTENT_PLAYPAUSE"
```

Example 2: Cancel

From the user's UI perspective, this is similar to *Pause* due to ceasing the audio being played.

ADB Command:
```
adb shell am startservice -n
"org.greatfire.freebook/co.mobiwise.libraryradio.media.RadioPlayerService" -a
"co.mobiwise.library.notification.radio.INTENT_CANCEL"
```

The very basic reason as to why these scenarios pan out is that the *RadioPlayerService* is implicitly exported to third-party apps via its *intent-filter* declaration in the *Android Manifest*.

File:
AndroidManifest.xml

Affected Code:
```
<service android:name="co.mobiwise.libraryradio.radio.RadioPlayerService">
   <intent-filter>
     <action
android:name="co.mobiwise.library.notification.radio.INTENT_PLAYPAUSE"/>
     <action
android:name="co.mobiwise.library.notification.radio.INTENT_OPENPLAYER"/>
     <action
android:name="co.mobiwise.library.notification.radio.INTENT_CANCEL"/>
   </intent-filter>
</service>
```

It is recommended to extrapolate the guidance offered under GF-01-009 to resolve the problems described here.

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

GF-01-011 Android: User Disruption via *PlayerControllerBroadcast* (*Low*)

It was found that malicious apps can *Pause* and *Resume* radio and audiobook playback at any time by sending crafted broadcasts. Unlike GF-01-009 and GF-01-010, this does not crash the app, yet could be useful for adversaries wishing to annoy the users to the point of quitting the use of GreatFire's product. In other words, this issue makes listening to an audiobook or a radio station unfeasible. Please note that this disruption is limited to a situation when a user has the audiobook or a radio station already open.

The problem can be replicated by opening a radio station or an audiobook and running any of the following ADB Commands.

Media Player ADB Commands:
```
adb shell am broadcast -a "co.mobiwise.library.ACTION_STOP_MEDIAPLAYER"
adb shell am broadcast -a "co.mobiwise.library.notification.media.INTENT_CANCEL"
adb shell am broadcast -a
"co.mobiwise.library.notification.media.INTENT_OPENPLAYER"
adb shell am broadcast -a
"co.mobiwise.library.notification.media.INTENT_PLAYPAUSE"
```

Radio Player ADB Commands:
```
adb shell am broadcast -a "co.mobiwise.library.ACTION_STOP_RADIOPLAYER"
adb shell am broadcast -a "co.mobiwise.library.notification.radio.INTENT_CANCEL"
adb shell am broadcast -a
"co.mobiwise.library.notification.radio.INTENT_OPENPLAYER"
adb shell am broadcast -a
"co.mobiwise.library.notification.radio.INTENT_PLAYPAUSE"
```

As with the above problems, the root cause for this issue can be found in the *Android Manifest*, which implicitly exports the *PlayerControllerBroadcast* receiver to third-party apps via its *intent-filter* declaration.

File:
AndroidManifest.xml

Affected Code:
```
<receiver
android:name="co.mobiwise.libraryradio.broadcast.PlayerControllerBroadcast">
   <intent-filter>
      <action android:name="co.mobiwise.library.ACTION_STOP_RADIOPLAYER"/>
      <action android:name="co.mobiwise.library.ACTION_STOP_MEDIAPLAYER"/>
      <action
android:name="co.mobiwise.library.notification.radio.INTENT_OPENPLAYER"/>
      <action
android:name="co.mobiwise.library.notification.radio.INTENT_PLAYPAUSE"/>
```

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

```
      <action
android:name="co.mobiwise.library.notification.radio.INTENT_CANCEL"/>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_PLAYPAUSE"/>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_CANCEL"/>
      <action
android:name="co.mobiwise.library.notification.media.INTENT_OPENPLAYER"/>
   </intent-filter>
</receiver>
```

It is recommended to stop exporting this receiver to third-party apps. Mitigation could be accomplished by setting the *android:exported* attribute to "*false*" in a manner demonstrated next.

Proposed Fix:

```
<receiver
android:name="co.mobiwise.libraryradio.broadcast.PlayerControllerBroadcast"
android:exported="false">
```

GF-01-012 Android: Intent Forward via *NotificationBroadcastReceiver* (*Medium*)

It was found that the *NotificationBroadcastReceiver,* available on the latest build provided for testing, is exported to third-party apps. In essence, it will forward intent data to any activity in the app, regardless of whether it is exported or not. The receiver will instantiate the class for the supplied activity's name and forwards the data to this activity. This could allow malicious apps to reach and attack functionality in other and not exported activities of the application. A lower-severity issue also exists here as the receiver can also open third-party provided URLs in the Android web browser.

Issue 1: Intent data forward

Execution of the code path in the receiver can be verified by consulting the lines below. First, a crafted broadcast must be sent, so that execution of the appropriate code path is verified.

ADB Command:

```
adb shell am broadcast -a "notification_cancelled" --es ""from"" "meow" --es
""content"" "test"
```

Completing this step results in the app attempting to instantiate a "*meow*" class, which shows a stack trace on logcat but does not crash the app as there is appropriate exception handling in place for such cases.

Logcat trace:

```
08-21 07:40:44.950 W/System.err( 3708): Caused by: 
java.lang.ClassNotFoundException: Didn't find class "meow" on path: 
DexPathList[[zip file "/data/app/org.greatfire.freebook-
1/base.apk"],nativeLibraryDirectories=[/data/app/org.greatfire.freebook-
1/lib/x86, /data/app/org.greatfire.freebook-1/base.apk!/lib/x86, /vendor/lib, 
/system/lib]]
[...]
08-21 07:40:44.950 W/System.err( 3708): 
at java.lang.Class.classForName(Native Method)
```

The exception corresponds to the following code snippet, which is also attempting to forward third-party provided data to any attacker-supplied activity within the app context (i.e. only activities in the FreeBooks app can be invoked).

File:
org/greatfire/gfapplib/services/NotificationBroadcastReceiver.java

Affected Code (decompiled):

```
    public void onReceive(Context var1_1, Intent var2_2) {
        var3_3 = var2_2.getStringExtra("web");
        if (var3_3 != null) {
            var4_4 = new Intent("android.intent.action.VIEW", 
Uri.parse((String)var3_3));
            var4_4.setFlags(268435456);
            var1_1.startActivity(var4_4);
            return;
        }
        var6_5 = var2_2.getAction();
        var7_6 = var2_2.getStringExtra("from");
        var8_7 = var2_2.getStringExtra("content");
        try {
            var10_9 = var20_8 = Class.forName(var7_6);
        }
        catch (ClassNotFoundException var9_12) {
            var9_12.printStackTrace();
            var10_9 = null;
        }
        if (var10_9 == null) ** GOTO lbl27
        try {
            if (var10_9.newInstance() instanceof Activity) {
                var14_10 = new Intent(var1_1, var10_9);
                var14_10.setFlags(268435456);
                var14_10.putExtra("notify", true);
                var14_10.putExtra("content", var8_7);
                SystemUtils.isAppRunning(var1_1, var1_1.getPackageName());
```

```
                 Log.d((String)"BroadcastReceiver", (String)("action " +
var6_5 + " " + var7_6 + " " + var10_9 + " " + var1_1.getPackageName()));
                 var1_1.startActivity(var14_10);
                 return;
             }
lbl27: // 3 sources:
             Toast.makeText((Context)var1_1,
(CharSequence)"\u53d1\u73b0\u5f02\u5e38", (int)1);
             return;
         }
```

This issue bears similarity to the intent forwarding weaknesses since it lets attackers reach activities that are not exported.

Issue 2: Browser

It was also found that the receiver, under a different code path, will open URLs received via broadcasts on the Android browser. While the security implications of this behavior are merely informational, this issue is briefly described here for the sake of completeness. The flaw at hand can be verified by running either of the following ADB commands while the app is open. This will result in the application opening the third-party provided website without any warnings or prompts on the Android browser.

ADB Commands:
```
adb shell am broadcast -a "notification_clicked" --es ""web""
"https://cure53.de"
adb shell am broadcast -a "notification_cancelled" --es ""web""
"https://cure53.de"
```

When one reviews the logcat logs carefully, it can be seen how the UID corresponds to that of the app on the phone.

Evidence of Browser Invocation by FreeBooks app:
```
08-21 07:14:53.998 I/ActivityManager(  614): START u0
{act=android.intent.action.VIEW dat=https://cure53.de/... flg=0x10000000
cmp=com.android.browser/.BrowserActivity} from uid 10090 on display 0
```

Commands:
```
su - app_90
app_90@vbox86p:/data/data/org.greatfire.freebook $ id
```

Output:
```
uid=10090(u0_a90) gid=10090(u0_a90) groups=10090(u0_a90)
```

Both issues occur because the receiver is implicitly exported to third-party apps via its *intent-filter* declaration in the *Android Manifest*.

File:
AndroidManifest.xml

Affected Code:
```
<receiver
android:name="org.greatfire.gfapplib.services.NotificationBroadcastReceiver">
  <intent-filter>
     <action android:name="notification_cancelled"/>
     <action android:name="notification_clicked"/>
  </intent-filter>
</receiver>
```

It is recommended to review the business need for this receiver and ideally remove it, or, at the very least, stop exporting it.

GF-01-013 Backend: User has Write Access to Github Repository *(Medium)*

It was found that the *ubuntu* user has write access to the Github repository wherein code for the backend and the mobile app are stored, namely:

git@github.com:greatfire/CollateralFreedomProxy.git.

An attacker who gains access to this repository could easily place a backdoor in the server-side code or even add malware to the mobile application.

Production systems should never have access to source code repositories and shall not even store anything else than the newest revision of the project. In some cases, the *Git* history could reveal confidential information, for instance when one commit contained passwords. It is recommended to deny access to the Github repository when using the private key stored in *ubuntu's .ssh* directory.

GF-01-014 Backend: Weak file permissions and lack of separation *(Medium)*

It was found that the */home/ubuntu* directory can be accessed by any user on the system. This folder contains configurations, log files and *Git* repositories. One example of consequences in this realm would be that a user with limited access could read the *nagios* configuration (*server/objects/hosts/ ec2-us-psql-master.cfg*) or web server logs (*/home/ubuntu/CollateralFreedomProxy/ backend/nginx/local/logs/error.log*).

Applications should be run only by the dedicated user with no access permissions to data unrelated to the service.

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.

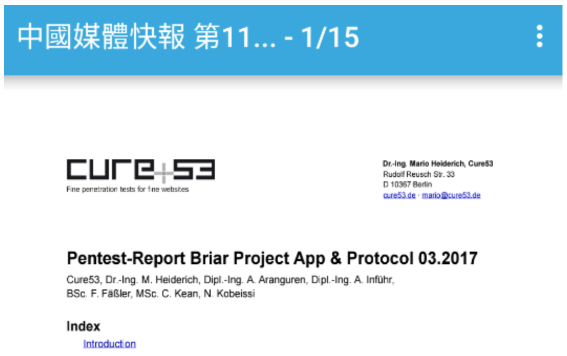
GF-01-001 Android: Unencrypted books on the SD Card might trouble users *(Low)*

It was found that the GreatFire FreeBooks Android app currently saves all books viewed by a user in clear-text on the SD Card. This provides a side-channel since malicious apps running on a victim's phone can gain insights about what a user is reading and listening to. In addition to this, if a Free Books user is stopped by the authorities, extraction of the SD Card does not require knowledge of the phone's unlock pattern. The authorities could directly learn what the user is reading and/or is listening to. In certain political contexts, the unveiled lists may put the Free Books' user into legal trouble. An additional issue exists in terms of lack of integrity. Notably, a malicious app or an attacker with physical access to the device can trivially change book files. By this logic, unexpected files are viewed when the user opens them unsuspectingly.

It can be verified that the app downloads all book material, including PDFs and MP3 files for the audiobook chapters, into a specific SD Card location. This can be accomplished by simply using the app and then reviewing the contents of the directory furnished below.

SD Card storage path:
/storage/emulated/0/Android/data/org.greatfire.freebook

Moreover, PDF and MP3 files can be trivially replaced on the SD Card, so that the rendered content is different than intended. This was confirmed during testing by replacing a PDF book with a Cure53 report in a PDF format. Note that the FreeBooks app will render the replaced file without any warnings.



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 Briar Project App & Protocol 03.2017
Cure53, Dr.-Ing. M. Heiderich, Dipl.-Ing. A. Aranguren, Dipl.-Ing. A. Inführ, BSc. F. Fäßler, MSc. C. Kean, N. Kobeissi

Index
Introduction

Fig.: Cure53 Report in a PDF format opened instead of the intended book

It is recommended to use the Android internal app storage to avoid leakage to malicious apps and attackers with physical access. When not enough storage is available on the internal Android storage, then book material should ideally be encrypted on the SD Card. This way, malicious apps and physical attackers at least cannot easily see what the user was reading or listening to. This being said, perhaps the mere presence of the *org.greatfire.freebook* application on the SD Card might be enough to get users in trouble when dealing with the authorities. Regarding the potential space issue on the internal Android storage, the app can reduce its storage usage by half if ZIP files were removed after downloading. At the time of writing, files are downloaded as ZIP files and uncompressed, but the ZIP files are not removed, hence taking approximately twice as much space as they should.

GF-01-002 Android: Backup flag might allow data theft *(Info)*

It was found that the Android app has the *backup* flag enabled. This might allow data theft in rare situations where the user has USB debugging enabled. This issue can be observed on the *Android Manifest* of the application.

File:
AndroidManifest.xml

Affected Code:
```
<application android:allowBackup="true" android:icon="@drawable/ic_launcher"
android:label="@string/app_name" android:largeHeap="true"
android:supportsRtl="true" android:theme="@style/RootTheme">
```

It is recommended to disable the *allowBackup* flag[3] on the *Android Manifest* to avoid potential data leakage in future releases.

GF-01-004 Android: Privacy concerns due to screenshot leaks *(Info)*

It was found that the GreatFire FreeBooks app fails to leverage the available platform protections to prevent information leakage via screenshots. This allows applications with screen recording or root privileges to capture all information displayed on the screen while the app is opened. A malicious app could leverage this weakness to spy on what the user reads, as well as on their general reading habits.

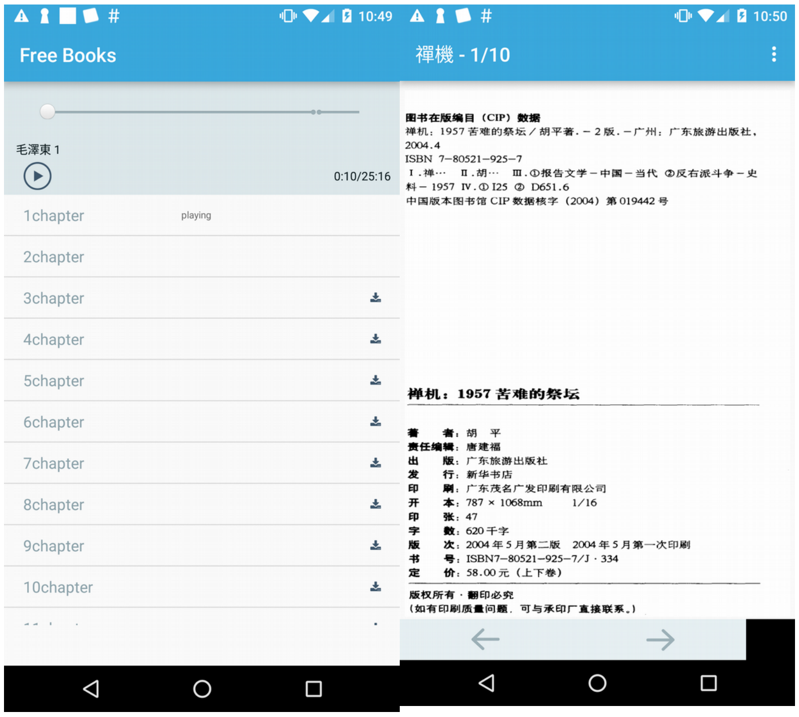
The issue can be verified by running the following commands while the emulator or a device running the FreeBooks app is open.

[3] http://developer.android.com/reference/android/R.attr.html#allowBackup

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

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

Depending on what the user is reading or listening to, different images appear. A selection of the results can be consulted below.



图书在版编目（CIP）数据
禅机：1957 苦难的祭坛／胡平著. – 2 版. – 广州：广东旅游出版社，2004.4
ISBN 7–80521–925–7
Ⅰ. 禅…　Ⅱ. 胡…　Ⅲ. ①报告文学 – 中国 – 当代　②反右派斗争 – 史料 – 1957　Ⅳ. ① I25　②　D651.6
中国版本图书馆 CIP 数据核字（2004）第 019442 号

禅机：1957 苦难的祭坛

著　　者：胡　平
责任编辑：唐建福
出　　版：广东旅游出版社
发　　行：新华书店
印　　刷：广东茂名广发印刷有限公司
开　　本：787 × 1068mm　　1/16
印　　张：47
字　　数：620 千字
版　　次：2004 年 5 月第二版　2004 年 5 月第一次印刷
书　　号：ISBN7–80521–925–7/J · 334
定　　价：58.00 元（上下卷）

版权所有 · 翻印必究
（如有印刷质量问题，可与承印厂直接联系。）

Fig.: User-activity information leakage via screenshots

It is recommended to ensure that all WebViews have the Android *FLAG_SECURE* flag[4] set. This will guarantee that even applications running with root privileges are unable to directly record information displayed by the app on screen. Ideally, this mitigation should be implemented as a *BaseActivity* that all other app activities inherit.

GF-01-005 Android: User disruption via *TagManagerPreviewActivity* (*Low*)

It was found that the Android FreeBooks app currently exports a *TagManagerPreviewActivity* activity. This appears to be an attempt to use the *GoogleTagManager* on Firebase[5], however the implementation seems incomplete. A malicious app or website could leverage this weakness to continuously show blank screens, hence disrupting the reading of FreeBooks' users. The flaw can be replicated

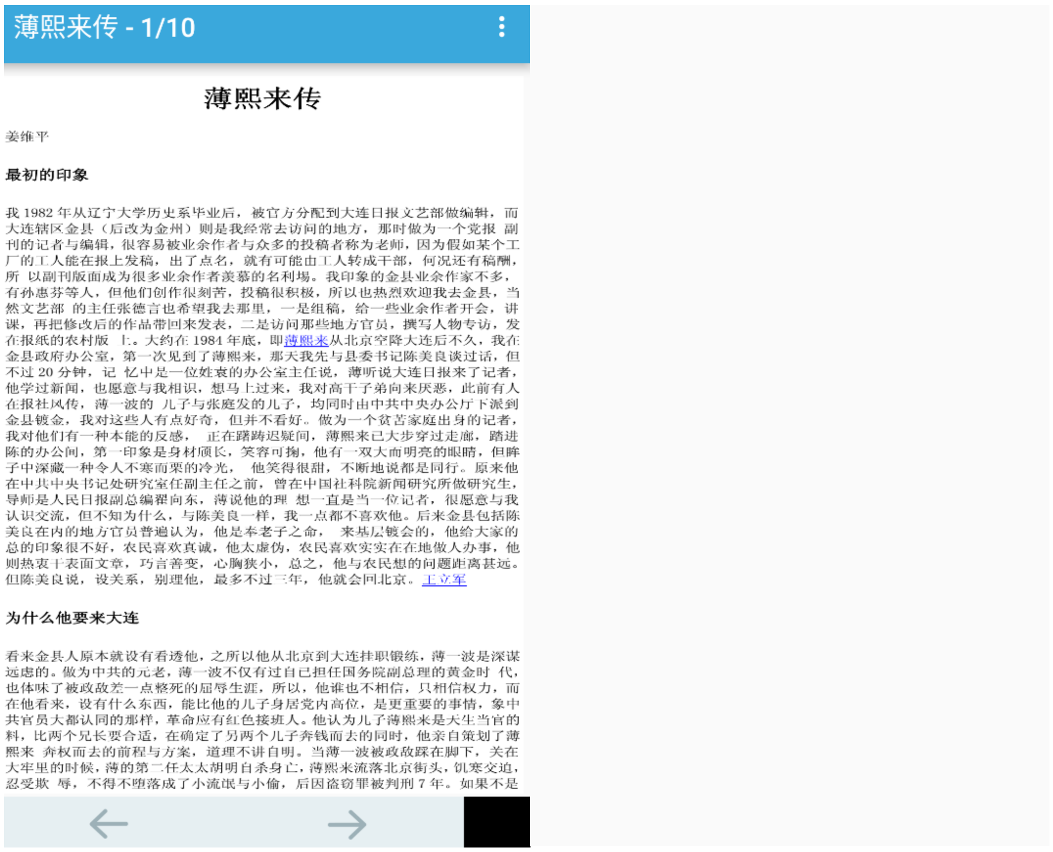
[4] http://developer.android.com/reference/android/view/Display.html#FLAG_SECURE
[5] https://developers.google.com/tag-manager/android/v5/

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

by running the following command on a phone or an emulator that has the FreeBooks app installed.

ADB Command:
```
adb shell am start -a "android.intent.action.VIEW" -n
"org.greatfire.freebook/com.google.android.gms.tagmanager.TagManagerPreviewActiv
ity"
```

This results in a blank screen being shown by the app.



薄熙来传

姜维平

最初的印象

我 1982 年从辽宁大学历史系毕业后，被官方分配到大连日报文艺部做编辑，而大连辖区金县（后改为金州）则是我经常去访问的地方，那时做为一个党报 副刊的记者与编辑，很容易被业余作者与众多的投稿者称为老师，因为假如某个工厂的工人能在报上发稿，出了点名，就有可能由工人转成干部，何况还有稿酬，所 以副刊版面成为很多业余作者羡慕的名利場。我印象的金县业余作家不多，有孙惠芬等人，但他们创作很刻苦，投稿很积极，所以也热烈欢迎我去金县，当然文艺部 的主任张德言也希望我去那里，一是组稿，给一些业余作者开会，讲课，再把修改后的作品带回来发表，二是访问那些地方官员，撰写人物专访，发在报纸的农村版 上。大约在 1984 年底，即薄熙来从北京空降大连后不久，我在金县政府办公室，第一次见到了薄熙来，那天我先与县委书记陈美良谈过话，但不过 20 分钟，记 忆中是一位姓袁的办公室主任说，薄听说大连日报来了记者，他学过新闻，也愿意与我相识，想马上过来，我对高干子弟向来厌恶，此前有人在报社风传，薄一波的 儿子与张庭发的儿子，均同时由中共中央办公厅下派到金县镀金，我对这些人有点好奇，但并不看好。做为一个贫苦家庭出身的记者，我对他们有一种本能的反感， 正在踌躇迟疑间，薄熙来已大步穿过走廊，踏进陈的办公间，第一印象是身材颀长，笑容可掬，他有一双大而明亮的眼睛，但眸子中深藏一种令人不寒而栗的冷光， 他笑得很甜，不断地说都是同行。原来他在中共中央书记处研究室任副主任之前，曾在中国社科院新闻研究所做研究生，导师是人民日报副总编翟向东，薄说他的理 想一直是当一位记者，很愿意与我认识交流，但不知为什么，与陈美良一样，我一点都不喜欢他。后来金县包括陈美良在内的地方官员普遍认为，他是奉老子之命， 来基层镀金的，他给大家的总的印象很不好，农民喜欢真诚，他太虚伪，农民喜欢实实在在地做人办事，他则热衷于表面文章，巧言善变，心胸狭小，总之，他与农民想的问题距离甚远。但陈美良说，设关系，别理他，最多不过三年，他就会回北京。王立军

为什么他要来大连

看来金县人原本就设有看透他，之所以他从北京到大连挂职锻练，薄一波是深谋远虑的。做为中共的元老，薄一波不仅有过自已担任国务院副总理的黄金时 代，也体味了被政敌差一点整死的屈辱生涯，所以，他谁也不相信，只相信权力，而在他看来，设有什么东西，能比他的儿子身居党内高位，是更重要的事情，象中共官员大都认同的那样，革命应有红色接班人。他认为儿子薄熙来是天生当官的料，比两个兄长要合适，在确定了另两个儿子奔钱而去的同时，他亲自策划了薄熙来 奔权而去的前程与方案，道理不讲自明。当薄一波被政敌踩在脚下，关在大牢里的时候，薄的第二任太太胡明自杀身亡，薄熙来流落北京街头，饥寒交迫，忍受欺 辱，不得不堕落成了小流氓与小偷，后因盗窃罪被判刑 7 年。如果不是

Fig.: A blank screen is shown while the user is reading a book

The main reason behind this issue can be found on the *Android Manifest*, which currently exports this activity to third-party websites and apps implicitly, via its *intent-filter* declaration.

File:
AndroidManifest.xml

Affected Code:
```
<activity
android:name="com.google.android.gms.tagmanager.TagManagerPreviewActivity"
android:noHistory="true">
   <intent-filter>
      <data android:scheme="tagmanager.c.org.greatfire.freebook"/>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.DEFAULT"/>
      <category android:name="android.intent.category.BROWSABLE"/>
   </intent-filter>
</activity>
```

It is recommended to review the business need to export this activity and, if possible, remove it from the *Android Manifest.* Alternatively, the activity could be used only internally if it was to be no longer exported in the Manifest. To accomplish this, an activity shall be added to the declaration as noted below.

Proposed Fix:
```
<activity
android:name="com.google.android.gms.tagmanager.TagManagerPreviewActivity"
android:noHistory="true" android:exported="false">
```

GF-01-008 Backend: Lack of Consistent Server Hardening *(Low)*

Most Linux default installations have several security options disabled due to requiring individual work or possibly affecting the system's usability for the majority of the users. There are several configuration options listed below and known for significantly improving the security of a Linux server.

General characteristic of being up-to-date
It was found that the software running on the servers is outdated. Both the kernel and the installed packages are affected by this pattern. Old software frequently contains known vulnerabilities, so it is recommended to perform a full system update and deploy a broader update strategy.

Hidepid
Every user can see all of the processes and their parameters on a Linux server. Under certain premise, this behavior might leak information or point an attacker in the right direction when it comes to escalating privileges. *Hidepid* is an option that can be

activated when the *procfs*[6] is mounted. This can be achieved with the following entry inside the server's *fstab*.

Command:
```
$ cat /etc/fstab
```

Output:
```
[...]
proc            /proc           proc        hidepid=2       0 0
```

If enabled, a non-root user can exclusively see the processes that were started by them and not others.

Dmesg Restrict

Dmesg[7] is a Linux command showing messages printed by the kernel. It contains information about the boot process and hardware, which means that in some cases it might disclose certain details to an attacker. This especially holds for an adversary who already has limited privileges on the server and can now escalate to root. There is no reason why a non-root user should see this output. It is recommended to restrict the access to kernel messages to root by adding the following line to the *sysctl* configuration:

```
kernel.dmesg_restrict = 1
```

Remote Syslog

Alongside local logging, it is advised to set up an external logging server. In case the server is compromised, an attacker can easily remove all evidence from the log files, thus making it difficult to even detect the attack, not to mention preventing the maintainers from understanding the attack that just took place. The consequences would be alleviated had the logs been stored on another server.

File Change Monitoring

An attacker who compromised a server most likely seeks to stay on the system as long as possible. This can be achieved by manipulation of e.g. executables on the server. It is recommended to verify the integrity of the installed packages with the regular use of a file change monitor. This would aid detection of manipulations.

su Restriction

The *su* command is used to log in as another user. If the system is not configured properly, everyone is allowed to take advantage of this *util* and can authenticate as users

[6] https://en.wikipedia.org/wiki/Procfs
[7] https://en.wikipedia.org/wiki/Dmesg

whose passwords are known. For example, a compromised service, which runs on the server, enables an attacker to use or guess passwords for other accounts. This could directly assist the goals of escalating privileges. As there is no common use-case for a service-user to be privy to *su,* it is advised to disallow this kind of usage. The issue can be solved in two steps. Firstly, all users who should be allowed to use *su* must be added to the *wheel* group. Secondly, the *su*-configuration needs some changes. A specific line shown next needs to be added.

File:
/etc/pam.d/su
```
auth        required    pam_wheel.so group=wheel
```

GF-01-015 Proxy: Static AES Initialization Vector and Encryption Key *(Low)*

It was found that a static AES initialization vector, along with an ever so slightly obfuscated static encryption key, are being used to encrypt/decrypt data. This is not a recommended good practice as the encrypted data can be easily and quite reliably detected in transit and storage. On the one hand, it is acknowledged that this might have been a conscious design decision to take the load off the backend infrastructure. On the other hand, the impact of a properly designed encryption scheme with a prepended and asymmetrically encrypted, but cyclically changing symmetric encryption key and dependent data, would not be much higher than the current cryptographically unsafe low-impact scheme.

File:
/android-library/jni/aes-wrap.c

Affected Code:
```
static unsigned char aesIV[]  = {0xe9, 0x82, 0xdd, 0x94, 0x9a, 0xea, 0xa7, 0x44,
0x12, 0x2a, 0x18, 0xd9, 0xc5, 0x46, 0x91, 0x76};
static unsigned char aesKEY[] = {0x6e, 0xfe, 0xc7, 0xe2, 0x01, 0x73, 0x9c, 0x93,
0x95, 0xf3, 0x90, 0xcb, 0x74, 0x3d, 0x4e, 0xda,
                                 0x88, 0x6e, 0x8b, 0xa2, 0xc0, 0x51, 0x0a,
0x4c, 0x15, 0x56, 0x9f, 0x2b, 0xfe, 0xaa, 0xdc, 0x10};

[...]

void init_aes_key(void* vm){
  aesKEY[1] = aesKEY[1] + 0x02;
  aesKEY[2] = aesKEY[1] - 0x02;
  aesKEY[3] = aesKEY[4] >> 1;
  aesKEY[9] = aesKEY[3] + 0x09;
}
```

It is strongly recommended to use an established encryption scheme as implemented by libraries such as NaCl or libsodium instead of relying on the handcrafted and easily breakable cryptographic code. Please refer to the *Notes on Cryptographics Implementations* included later in this document for additional details.

GF-01-016 Proxy: Oversize AES Initialization Vector Buffer Size *(Low)*

It was found that the *constant,* which is employed to specify the size for copying the initialization vector into the respective programmatically used buffers, suffers from having wrong dimension. Therefore too much data from the end of the static initialization vector (right now the beginning of the static encryption key) is copied to the equally oversized buffer. While this does not directly lead to an exploitable scenario, it still poses a looming problem and should be addressed.

File:
/android-library/jni/aes-wrap.c

Affected Code:
```
char* aes_message_decode(const char *mix_encrypt_msg) {
[...]
  unsigned char ivenc[AES_BLOCK_SIZE] = {0};
  const char *raw = mix_encrypt_msg;
  int i = 0, j = 0, idx = 0;
[...]

  while(j < AES_BLOCK_SIZE) {
       ivenc[j] = *(raw + j);
       key[i] = ivenc[j];
       //printf("key = %s, %c, j = %d, i = %d\n", key, ivenc[j], j, i);
       i += 2, j ++;
  }
[...]

char* aes_message_encode(const char *textbuf) {
[...]
  unsigned char ivenc[AES_BLOCK_SIZE] = {0};
  int i = 0, j = 0, idx = 0;
[...]
  unsigned char aes_key[32] = {0};
[...]

  for(j=0, i=0; j < AES_BLOCK_SIZE; j++, i+=2) {
       ivenc[j] = aes_key[i];
  }
[...]
```

```
unsigned char* restore_from_cache(char* filename, unsigned long *nsize) {
[...]
      unsigned char ivenc[AES_BLOCK_SIZE] = {0};

      memcpy(ivenc, aesIV, AES_BLOCK_SIZE);
[...]

int save_to_cache(char* filename, unsigned char* membuf, int nsize) {
[...]
      unsigned char ivenc[AES_BLOCK_SIZE] = {0};

      memcpy(ivenc, aesIV, AES_BLOCK_SIZE);
[...]
```

It is recommended to change the *constant* with respect to block size used. Any correct version may be considered and it will look similar to: *sizeof(aesIV).* Another constant like *AES_IV_SIZE* instead of the inappropriate application of *AES_BLOCK_SIZE* can also be used in this realm.

GF-01-017 Proxy: Low resolution non-random Message Key Generation (*Low*)

It was found that the random number used for the generation of the message keys is based upon a simple and no longer recommended MD5-hash of the system's *time()* and is therefore not considered cryptographically strong. Additionally, the low resolution system's *time()* was turned into a space-reduced *ctime()*-string, resulting in an even lower random variation. The resulting MD5-hash of 128 bits is then widened to the 256 bits dictated by the existing implementation by further turning it into a hexdump of the binary value, which fails to increase the resulting cryptographic value in any way.

File:
/android-library/jni/aes-wrap.c

Affected Code:
```
char* aes_message_encode(const char *textbuf) {
[...]
  int i = 0, j = 0, idx = 0;
[..]
  unsigned char aes_key[32] = {0};
[...]
  time_t tm = time(NULL);
  char *stm = ctime(&tm);
  unsigned char md5[16]  = {0};

  MD5(stm, strlen(stm), md5);
  for(i=0; i<16; i++) {
      sprintf(aes_key+(i<<1), "%02x", md5[i]);
  }
[...]
```

It is recommended to use any of the cryptographic random number generators available in a UNIX-based system, like */dev/urandom* or the *AES-CTR* DRBG. Though usage of */dev/random* would have been proposed in the past, it is no longer recommended because it is prone to a particular set of problems.

GF-01-018 Proxy: Hardcoded Buffer Size value across different Modules (*Low*)

It was found that a hardcoded buffer size value is being used across different modules. While this is quite problematic within a single module in its own right, it is certainly not advisable across different modules. It should especially be stopped on a buffer allocated in one module's function and later on taken on for the externally supplied data, as it is being read into by the called function residing in a completely different module.

Affected Files:
/android-library/jni/aes-wrap.c
/android-library/jni/proxy-bridge.c

Affected Code:

```
int get_cacert_pembuf(char *apkfile, char *pembuf) {
[...]
  unzFile *zf = unzOpen(apkfile);
[...]
        unzOpenCurrentFile(zf);
        err = unzReadCurrentFile(zf, pembuf, 242842);
        if(err > 0) {
        DD("get_cacert_pembuf.nread = %d, totalsize = 242842\n", err);
[...]

JNIEXPORT int JNICALL Java_org_greatfire_XPatch_setApkInfo(JNIEnv *env, jobject
thiz, jstring apkpath, jstring apppath, int versioncode, jstring vername,
jstring pkgname) {
  const jbyte *utf8 = NULL;
[...]
  char* pembuf = NULL;
[...]
        utf8 = (* env)->GetStringUTFChars(env, vername, NULL);
[...]
        pembuf = calloc(242842, sizeof(char));
        if(pembuf) {
[...]
        get_cacert_pembuf(utf8, pembuf);
[...]
```

It is recommended for the length of the buffer to be passed from *Java_org_greatfire_XPatch_setApkInfo()* to the called *get_cacert_pembuf()* with an additional *pembufsize* paramete. The later should be used there to read into the allocated buffer and make sure that the buffer previously allocated on the heap is not overrun.

Notes on Cryptographic Implementations

The Android client uses an AES-based symmetric encryption scheme in order to encrypt books before delivering them to users. The scheme relies on a custom AES wrapper that includes static keys and initialization vectors. The same static key and IV are used to encrypt a given book across all users.

This approach cannot be seen as satisfactory in terms of confidentiality or integrity guarantees around the data. An attacker can decrypt all books on the fly and may also violate integrity guarantees. This can result in the attacker, for example, sending malicious payloads disguised as books. In practice, no security gains whatsoever are actually provided by the current design. A proposal for an improved design, which actually fosters confidentiality and integrity, is supplied next.

1. The FreeBooks server generates a long-term, static X5519 key pair.
2. The FreeBooks server's public key is statically embedded inside the FreeBooks client binary.
3. Every FreeBooks user generates an X25519 public key pair on signup.
4. The FreeBooks user generates a shared secret using the newly generated secret key and the server's embedded public key.
5. The FreeBooks user communicates this new device's public key to the server (which lasts for the lifetime of the device.)
6. The server can now additionally calculate the same shared secret and use it to encrypt a bundle of books to the client.

The Strengths of the Proposal:

1. Confidentiality and integrity are preserved.
2. Traffic analysis is less potent.
3. Pinning of the server's public key acts as a kind of certificate Pinning mechanism, which helps prevent Man-in-the-Middle attacks.

The Weaknesses of the Proposal:

1. The proposed design relies on a trust-on-first-use authentication with no out-of-band authentication. However, the latter is likely not necessary given the use-case.

2. The proposed design probably requires more CPU usage for the server, which now has to encrypt towards a different public key for each user. This overhead can be lessened by having the server re-encrypt the single symmetric key used to encrypt the book instead of re-encrypting the entire book towards each client who requests it.

Conclusions

The findings of this Cure53 summer 2017 security assessment of the GreatFire project overall point to a relatively well-handled user-privacy and general safety at the Free Books application. Five members of the Cure53 team spent a total of eight days on completing this assessment, ultimately discovering eighteen relevant issues. Given the scope and available resources, the testing team reached a fairly good coverage. Still, there is a potential for expanding the investigation in the future, as the sheer volume of sources and configuration files warrants additional and more in-depth reviews.

Though the number of eighteen findings might appear concerning, the so-called levels of criticality must be accounted for in issuing a final verdict. In fact, the GreatFire Free Books project does not suffer from high-level risks, but is rather prone to minor flaws. Exactly half of the discoveries belong to the realm of general weaknesses, which means that the usual urgency of deploying fixes is minimized. The main themes pertinent to the actual nine security vulnerabilities revolve around DoS problems, few issues regarding server security with the worst case being privilege escalation, as well as potential for the attackers to get their hands on the *Git* repositories with non-public source code. What is vital to note is that none of these seemingly threatening issues can be achieved without gaining access to the GreatFire's infrastructure beforehand.

Besides the already commented on cryptographic implementation arena, two more aspects that deserve further summaries are C and Java code, as well as the Android application itself. As for the former, it shall be emphasized that the complete coverage of the Java and C source code during audit still left the Proxy Bridge as somewhat of an uncharted territory, especially due to this component's complexity. The Cure53 testers shared the general impression that the code was often hard to read and seemed obfuscated, thus slowing down the usual pace of conducting an audit. For that reason, it is recommended to refactor the code and optimize readability for future quality checks and reviews. It was further noticed that many code elements were handcrafted by the FreeBooks maintainers rather than rely on taking advantage of the available and suitable libraries. This impression extended to the cryptographic realm, as it came as a surprise for Cure53 that no libraries were used (i.e. neither libsodium nor NaCl).

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

As a general rule of thumb, the benefit of using the existing frameworks and tools is tremendous, much outweighing the potential adequacy of home-made code and components. In effect, the final verdict is somewhat of a paradox. On the one hand, there were no major security-relevant findings in the C/Java code realm. On the other hand, the code is not optimal, with clear advice to focus on refactoring and redesigns. Shifting towards the existing modern libraries can vastly raise the stability and robustness of the project at hand, with more details on this matter supplied above in the *Notes on Cryptographic Implementations* section.

Moving on to the arena of the Android application, the first and foremost observation is that it exposes a large attack surface to third-party apps. What is more, BROWSABLE activities extended the potential of different attack scenarios to websites. This resulted in a relatively high total number of findings as several activities, services and broadcast receivers were exported. Among the notable issues, problems related to the SD card usage constituted one pattern. Because the SD Card location easily allows for extraction, its analysis revealed possible race conditions leading to RCE due to downloading upgrade APKs to the SD Card (GF-01-003). Privacy concerns may also arise in situations where the authorities stop a FreeBooks user, extract the card and access the list of possibly “forbidden” books (see GF-01-001). With the MediaPlayerService and *RadioPlayerService* being the two exported media services, it was discovered that attackers may crash FreeBooks or interfere with audiobook and radio station playbacks, as noted in GF-01-009 and GF-01-010. Broadcast Receivers were also found to be processing intents from third-party applications. This behavior could be abused to reach functionality in non-exported activities (GF-01-012) or, once again, for playback interference (GF-01-011). As already noted, though, none of the issues pose tremendously direct risk of harm.

In sum, the GreatFire FreeBooks application stood relatively strong with respect to the actual degree of severity assigned to the spotted issues. The fixing process around the discoveries alone, however, is not sufficient. This is because the main problems stemmed from the complexity and suboptimal characteristics of the audited code. In that sense, there is much room for more bird’s eye view reassessments of the project and corresponding generalist optimizations at different planes are advised.

Cure53 would like to thank Martin Johnson and Charlie Smith from the GreatFire team for their excellent project coordination, support and assistance, both before and during this assignment.