Combining the Security Risks of Native and Web Development: Hybrid Apps

Achim D. Brucker and Michael Herzberg
{a.brucker, msherzberg}@shef.ac.uk
Software Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

Abstract

Cross-platform frameworks, such as Apache Cordova, are becoming increasingly popular. They promote the development of hybrid apps that combine native, i.e., system specific, code and system independent code, e.g., HTML/JavaScript. Combining native with platform independent code opens Pandora's box: all the the security risks for native development are multiplied with the security risk of web applications.

In the first half of our talk, we start our talk with short introduction into hybrid app development, present specific attacks followed by a report on how Android developers are using Apache Cordova. In the second half of the talk, we will focus on developing secure hybrid apps: both with hands-on guidelines for defensive programming as well as recommendations for hybrid app specific security testing strategies.

Outline

1. Hybrid mobile apps
2. An assessment of hybrid Apps (in Google Play)
3. Security challenges of hybrid apps
4. Secure implementation
5. Security testing of hybrid apps
6. Conclusion
Hybrid mobile apps

- Native apps: Java \ Swift \ C#
  - Developed for a specific platform
  - All features available

- Web apps: HTML5 and JS
  - Hosted on server, all platforms
  - No access to device features

Platform-specific

- Hybrid apps: HTML5, JS, and native
  - Build once, run everywhere
  - Access to device features through plugins
  - All features available

Platform-independent

Web apps: HTML5 and JS

Platform-dependent

The architecture of Apache Cordova

- UI Layer (HTML, CSS, JS)
  - Application Logic in JS
- Android WebView
- Cordova Plugins
  - Camera
  - Device Motion
  - In-App Browser
  - Network
  - Media
  - File
  - Geolocation
  - Vibration
- Custom Plugins
- Android Cordova Container
- Android Platform

Example app
class ContactsPlugin extends CordovaPlugin {

    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        String name = args.get(0).value;
        if ("find".equals(action)) {
            find(name, callbackContext);
        } else if ("create".equals(action)) {
            create(name, callbackContext);
        } else if ("delete".equals(action)) {
            delete(name, callbackContext);
        }
    }

    void find(String name, CallbackContext callbackContext) {
        Contact contact = query("SELECT * FROM CONTACTS WHERE name = " + name);
        callbackContext.success(contact);
    }

    void create(String name, CallbackContext callbackContext) {
        // Logic for creating a contact
    }

    void delete(String name, CallbackContext callbackContext) {
        // Logic for deleting a contact
    }

    void alert(String message) {
        // Logic for alerting
    }

    // Other methods
}

Example: get phone number

function showPhoneNumber(name) {
    var successCallback = function(contact) {
        alert("Phone number: " + contact.phone);
    }
    var failureCallback = ...;
    cordova.exec(successCallback, failureCallback, "ContactsPlugin", "find", [{"name": name}]);
}

class ContactsPlugin extends CordovaPlugin {

    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        String name = args.get(0).value;
        if ("find".equals(action)) {
            find(name, callbackContext);
        } else if ("create".equals(action)) {
            create(name, callbackContext);
        } else if ("delete".equals(action)) {
            delete(name, callbackContext);
        }
    }

    void find(String name, CallbackContext callbackContext) {
        Contact contact = query("SELECT * FROM CONTACTS WHERE name = " + name);
        callbackContext.success(contact);
    }

    void create(String name, CallbackContext callbackContext) {
        // Logic for creating a contact
    }

    void delete(String name, CallbackContext callbackContext) {
        // Logic for deleting a contact
    }

    void alert(String message) {
        // Logic for alerting
    }

    // Other methods
}

Example: get phone number

function showPhoneNumber(name) {
    var successCallback = function(contact) {
        alert("Phone number: " + contact.phone);
    }
    var failureCallback = ...;
    cordova.exec(successCallback, failureCallback, "ContactsPlugin", "find", [{"name": name}]);
}

class ContactsPlugin extends CordovaPlugin {

    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        String name = args.get(0).value;
        if ("find".equals(action)) {
            find(name, callbackContext);
        } else if ("create".equals(action)) {
            create(name, callbackContext);
        } else if ("delete".equals(action)) {
            delete(name, callbackContext);
        }
    }

    void find(String name, CallbackContext callbackContext) {
        Contact contact = query("SELECT * FROM CONTACTS WHERE name = " + name);
        callbackContext.success(contact);
    }

    void create(String name, CallbackContext callbackContext) {
        // Logic for creating a contact
    }

    void delete(String name, CallbackContext callbackContext) {
        // Logic for deleting a contact
    }

    void alert(String message) {
        // Logic for alerting
    }

    // Other methods
}
Example: get phone number

```javascript
function showPhoneNumber(name) {
  var successCallback = function(contact) {
    alert("Phone number: " + contact.phone);
  };
  var failureCallback = ...
  cordova.exec(successCallback, failureCallback, "ContactsPlugin", "find", {{"name": name}});
}
```

Example: get phone number

```javascript
class ContactsPlugin extends CordovaPlugin {
  boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
    if ("find".equals(action)) {
      String name = args.get(0).name;
      find(name, callbackContext);
    } else if ("create".equals(action)) ...
    void find(String name, CallbackContext callbackContext) {
      Contact contact = query("SELECT ... WHERE name" + name);
      callbackContext.success(contact);
    }
  }
}
```

One framework, many names

- Many frameworks extending Cordova
  - Adobe PhoneGap
  - SAP Kapsel
  - Onsen
  - ...
- These frameworks provide
  - additional plug-ins (access to native components)
  - additional HTML/JavaScript libraries or interfaces
Outline

1. Hybrid mobile apps
2. An assessment of hybrid Apps (in Google Play)
3. Security challenges of hybrid apps
4. Secure implementation
5. Security testing of hybrid apps
6. Conclusion

Cordova in the real world

- How many apps are using Cordova?
- How is Cordova used by app developers?
- Are cross-language calls common or not?

What we have done

Selection of apps
- all apps that ship Cordova from Google’s Top 1000:
  - 100 apps ship Cordova plugins
  - only 50 actually use Cordova (5%)
- three selected apps from SAP (using SAP Kapsel)

Development of a static analysis tool
- analysing Android apps (*.apk files)
- specialised in data-flows from Java to JavaScript and vice versa
- based on WALA
- in addition: list used plugins

Manual analysis of 8 apps (including one from SAP)
- to understand the use of Cordova
- to assess the quality of our automated analysis

What we have learned: plugin use

Plugins are used for
- accessing device information
- showing native dialog boxes and splash screens
- accessing network information
- accessing the file storage
- accessing the camera
- ...

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>device</td>
<td>52%</td>
</tr>
<tr>
<td>inappbrowser</td>
<td>50%</td>
</tr>
<tr>
<td>dialogs</td>
<td>40%</td>
</tr>
<tr>
<td>splashscreen</td>
<td>36%</td>
</tr>
<tr>
<td>network-information</td>
<td>28%</td>
</tr>
<tr>
<td>file</td>
<td>28%</td>
</tr>
<tr>
<td>console</td>
<td>24%</td>
</tr>
<tr>
<td>camera</td>
<td>22%</td>
</tr>
<tr>
<td>statusbar</td>
<td>22%</td>
</tr>
<tr>
<td>PushPlugin</td>
<td>22%</td>
</tr>
</tbody>
</table>

Source: https://www.appbrain.com/
What we have learned: app size and cross-language calls

### App size:
- mobile apps are not always small
- SAP apps seem to be larger than the average

### Cross-language calls:
- calls from Java to JS: very common
- calls from JS to Java: surprisingly uncommon

<table>
<thead>
<tr>
<th>App</th>
<th>Category</th>
<th>Java</th>
<th>JS</th>
<th>JS [kLoC]</th>
<th>Java [kLoC]</th>
</tr>
</thead>
<tbody>
<tr>
<td>app0</td>
<td>Finance</td>
<td>2</td>
<td>12</td>
<td>35.5</td>
<td>17.0</td>
</tr>
<tr>
<td>app1</td>
<td>Business</td>
<td>2087</td>
<td>39</td>
<td>345.3</td>
<td>53.5</td>
</tr>
<tr>
<td>app2</td>
<td>Business</td>
<td>9531</td>
<td>75</td>
<td>572.3</td>
<td>135.8</td>
</tr>
<tr>
<td>app3</td>
<td>Finance</td>
<td>9</td>
<td>13</td>
<td>26.3</td>
<td>12.8</td>
</tr>
<tr>
<td>app4</td>
<td>Finance</td>
<td>2</td>
<td>10</td>
<td>11.2</td>
<td>16.8</td>
</tr>
<tr>
<td>app5</td>
<td>Social</td>
<td>2349</td>
<td>31</td>
<td>4.6</td>
<td>103.7</td>
</tr>
<tr>
<td>app6</td>
<td>Business</td>
<td>1</td>
<td>6</td>
<td>37.5</td>
<td>16.8</td>
</tr>
<tr>
<td>app7</td>
<td>Finance</td>
<td>6</td>
<td>26</td>
<td>20.0</td>
<td>44.8</td>
</tr>
<tr>
<td>app8</td>
<td>Finance</td>
<td>693</td>
<td>70</td>
<td>30.4</td>
<td>24.3</td>
</tr>
<tr>
<td>app9</td>
<td>Travel &amp; Local</td>
<td>3420</td>
<td>43</td>
<td>129.0</td>
<td>304.0</td>
</tr>
<tr>
<td>app10</td>
<td>Entertainment</td>
<td>14220</td>
<td>67</td>
<td>36.7</td>
<td>23.0</td>
</tr>
<tr>
<td>app11</td>
<td>Lifestyle</td>
<td>51553</td>
<td>89</td>
<td>36.3</td>
<td>44.7</td>
</tr>
<tr>
<td>app12</td>
<td>Finance</td>
<td>8</td>
<td>36</td>
<td>43.7</td>
<td>18.4</td>
</tr>
<tr>
<td>app13</td>
<td>Business</td>
<td>0</td>
<td>0</td>
<td>14.0</td>
<td>438.9</td>
</tr>
</tbody>
</table>

Outline

1. Hybrid mobile apps
2. An assessment of hybrid Apps (in Google Play)
3. Security challenges of hybrid apps
4. Secure implementation
5. Security testing of hybrid apps
6. Conclusion

### Why is it hard to the the security of hybrid apps

- Web technologies (i.e., JavaScript)
  - lack of typing, higher order functions, asynchronous programming models
  - highly dynamic (e.g., eval(...), dynamic loading)
  - ... 

- Large Libraries and Modules
  - large (% 10kLoC) third party (FOSS, proprietary) libraries
  - both native (Java) and JavaScript
  - complex core framework
  - ... 

- Cross-Language-Analysis
  - many data-flows across language boundaries
  - datatype conversion
  - not only for accessing sensors (e.g., session plugin requires > 10 language switches)
  - ... 

What we have learned: exceptional behaviours

- Cordova use:
  - no HTML/JS in the app
  - no use of Cordova

- Plugin use:
  - often callbacks are not used
    - (missing error handling)
  - plugins are modified
  - plugins might use JNI
Weak spot: JS <-> Java bridge

Exploiting the JavaScript to Java bridge (CVE-2013-4710)

- We can expose Java methods in JavaScript
  ```javascript
  foo.addJavascriptInterface(new FileUtils(), "FUtil");
  ```
- And use them in JavaScript easily
  ```javascript
  filename = '/data/data/com.livingsocial.www/' + id + '_cache.txt';
  FUtil.write(filename, data, false);
  ```
- Which might expose much more than expected
  ```javascript
  function execute(cmd){
    return window._cordovaNative.getClass().forName('java.lang.Runtime').
      getMethod('getRuntime', null).invoke(null, null).exec(cmd);
  }
  ```
Never use http without SSL, or even iframes!

Outline
- Hybrid mobile apps
- An assessment of hybrid Apps (in Google Play)
- Security challenges of hybrid apps
- Secure implementation
- Security testing of hybrid apps
- Conclusion
Recommendations: the (hopefully) obvious parts

Cordova apps are Web applications:
- do secure JavaScript programming
- content security policy, same origin policy
- ...

**Warning:** the WebView sandbox is not as strong as on desktop Web browsers

Cordova apps are native/Java apps:
- do secure Java/Objective-C/... programming
- do not trust validations done in the JavaScript part of the plugin
- ...

Recommendations: we should not forget

Cordova apps are cordova applications:
- plugin whitelisting  
- read the Cordova security guide:

Cordova apps are mobile applications:
- permissions
- which OS versions do you need to support?
- check ssl certificates (debug mode!)
- ...

Did you know

- [Apache Cordova / CR-1234]{id='http://issues.apache.org/hudson/view/CR-1234@1'}
  [cordova-android] Skipping HTTPS certificate validation should be hidden behind a flag, and not a default in debug builds
  - [Agile Board](https://issues.apache.org/hudson/view/CR-1234@1)
    - Components: cordova-android, android
    - Labels: 
    - Priority: 
    - Status: Open
    - Votes: 
    - Created: 
    - Updated: 
  - [Description](https://github.com/apache/cordova-android/blob/1234567890abcdef/framework/src/org/apache/cordova/1234@1): Right now, when you create a debuggable Cordova build (debug behavior for commands like 'cordova run android' etc.), it by default ignores all HTTPS certificate errors, as you can see in the code below:
    ```javascript
    https://github.com/apache/cordova-android/blob/1234567890abcdef/framework/src/org/apache/cordova/1234@1```
  - [Fixes](https://issues.apache.org/hudson/view/CR-1234@1)
    - [cordova-android] Skipping HTTPS certificate validation should be hidden behind a flag, and not a default in debug builds
      - [Agile Board](https://issues.apache.org/hudson/view/CR-1234@1)
        - Components: cordova-android, android
        - Labels: 
        - Priority: 
        - Status: Open
        - Votes: 
        - Created: 
        - Updated: 
  - [Description](https://github.com/apache/cordova-android/blob/1234567890abcdef/framework/src/org/apache/cordova/1234@1): Right now, when you create a debuggable Cordova build (debug behavior for commands like 'cordova run android' etc.), it by default ignores all HTTPS certificate errors, as you can see in the code below:
    ```javascript
    https://github.com/apache/cordova-android/blob/1234567890abcdef/framework/src/org/apache/cordova/1234@1```
Recommendation: use the latest framework version

Frameworks (and the underlying OS) can have vulnerabilities:
- use the latest version of Cordova
- monitor for public known vulnerabilities (e.g., CVEs)

Or, in other words: secure your software supply chain

Framework vulnerabilities can be severe:
- Java code execution via JavaScript: CVE-2013-4710
  Avoid Cordova on Android below 4.1 & use AddJavaScriptInterface annotation
- (incomplete) overview: https://www.cvedetails.com/vulnerability-list/vendor_id-/two.pnum.oldstyle/five.pnum.oldstyle/product_id-/two.pnum.oldstyle/seven.pnum.oldstyle/one.pnum.oldstyle/five.pnum.oldstyle/three.pnum.oldstyle/Apache-Cordova.html

Outline
- Hybrid mobile apps
- An assessment of hybrid Apps (in Google Play)
- Security challenges of hybrid apps
- Secure Implementation
- Security testing of hybrid apps
- Conclusion

If you are using static analysis: Considerations

Deep framework analysis
- Closest to the actual program
- But: Framework very expensive

Modelling the framework
- Models the Cordova framework
- Analyses plugins

Modelling the plugin (interfaces)
- Models both framework and plugins
- Analyses only UI and business logic part
- But: Developers can write own plugins

If you are using static analysis: Recommendations

In case you are mostly
- developing HTML5/JavaScript:
  - Use a SAST tool for JavaScript
  - Ensure that your plugin APIs are configured as sinks/sources
- developing plugins:
  - Use a SAST tool for Java
  - Ensure that your plugin APIs are configured as sinks/sources
- developing full-featured Cordova apps
  - Method 1: scan Java and JavaScript “in isolation”
    - use a SAST tool for Java with configured sink/source for all plug-ins
  - Method 2: scan Java and JavaScript source code together
    - Ensure that Cordova framework is modelled to allow analysis of cross-language data-flows
If you are using dynamic analysis (e.g., pen testing)

- check for attacks on the native part by an "web attacker" (e.g., SQL injection)
- check for plug-ins with removed JavaScript part
  (If you can inject JavaScript, you can use those plug-ins)
- and it is always good to ensure that debug-mode is disabled

Outline

- Hybrid mobile apps
- An assessment of hybrid Apps (in Google Play)
- Security challenges of hybrid apps
- Secure Implementation
- Security testing of hybrid apps
- Conclusion

Conclusion

- Hybrid mobile apps are getting more popular
  - they are recommended by enterprise vendors
  - they are used outside of the "traditional mobile devices" (e.g., web kiosk, smart TVs)
- Securing hybrid apps is a challenge and requires expertise in
  - Web application security
  - native/Java security
  - mobile security
  - Cordova security

Thank you for your attention!
Any questions or remarks?

Contact:
Dr. Achim D. Brucker and Michael Herzberg
Department of Computer Science
University of Sheffield
Regent Court
Pamela Road
Sheffield S3 9PP, UK
{a.brucker, msherzberg}@sheffield.ac.uk
https://logicalhacking.com/blog/
Achim D. Brucker and Michael Herzberg.
On the static analysis of hybrid mobile apps: A report on the state of apache cordova nation.

Stanislav Dashevskyi, Achim D. Brucker, and Fabio Massacci.
On the security cost of using a free and open source component in a proprietary product.