

On the Static Analysis of Hybrid Mobile Apps

A Report on the State of Apache Cordova Nation

Achim D. Brucker and Michael Herzberg

{a.brucker,msherzberg1}@sheffield.ac.uk

Department of Computer Science, The University of Sheffield, Sheffield, UK
(Parts of this research were done while the authors were working at SAP SE in Germany.)

International Symposium on Engineering Secure Software and Systems (ESSoS 2016)
April 6 - 8, 2016, London, UK



On the Static Analysis of Hybrid Mobile Apps: A Report on the State of Apache Cordova Nation

Abstract

Developing mobile applications is a challenging business: developers need to support multiple platforms and, at the same time, need to cope with limited resources, as the revenue generated by an average app is rather small. This results in an increasing use of cross-platform development frameworks that allow developing an app once and offering it on multiple mobile platforms such as Android, iOS, or Windows.

Apache Cordova is a popular framework for developing multi-platform apps. Cordova combines HTML5 and JavaScript with native application code. Combining web and native technologies creates new security challenges as, e.g., an XSS attacker becomes more powerful.

In this paper, we present a novel approach for statically analysing the foreign language calls. We evaluate our approach by analysing the top Cordova apps from Google Play. Moreover, we report on the current state of the overall quality and security of Cordova apps.

Keywords: static program analysis, static application security testing, Android, Cordova, hybrid mobile apps.

Outline

- 1 Motivation: Hybrid Mobile Apps and their Security Challenges
- 2 Real World Cordova Usage
- 3 Static Analysis for Hybrid Apps: Building a Unified Call Graph
- 4 Quality of the Unified Call Graph
- 5 Conclusions

Motivation: Hybrid Mobile Apps and their Security Challenges

What is a Hybrid App?

Native, HTML5, or hybrid



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

Platform-independent

What is a Hybrid App?

Native, HTML5, or hybrid



Native apps

Java \ Swift \ C#

- Developed for a specific platform
- All features available



Hybrid apps

HTML5, JS, and native

- Build once, run everywhere
- Access to device features through plugins



Web apps

HTML5 and JS

- Hosted on server, all platforms
- No access to device features



Platform-specific

Platform-independent

Why Apache Cordova?

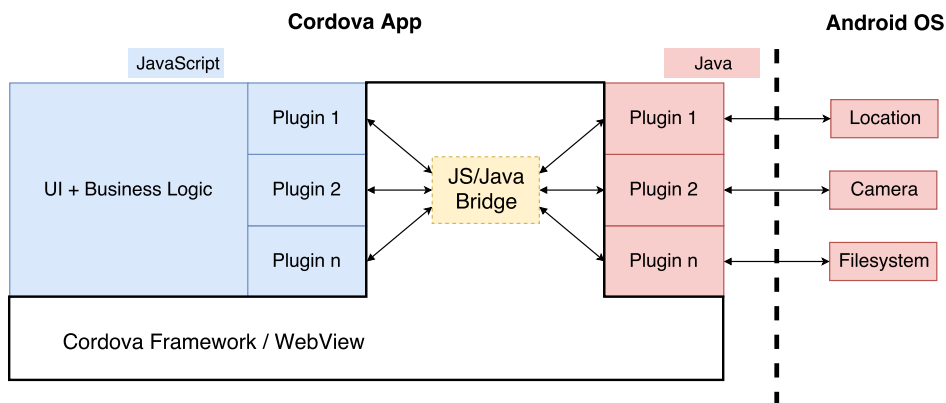


APACHE CORDOVA™

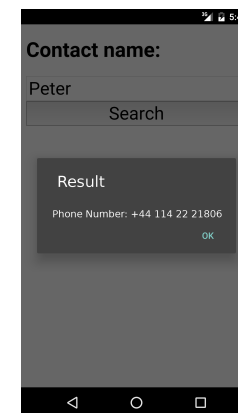
<https://cordova.apache.org/>

- Apache Cordova is most popular hybrid app framework
- Open source
- Many companies offer Apache Cordova plus commercial plugins (e.g., Adobe PhoneGap or SAP Kapsel)

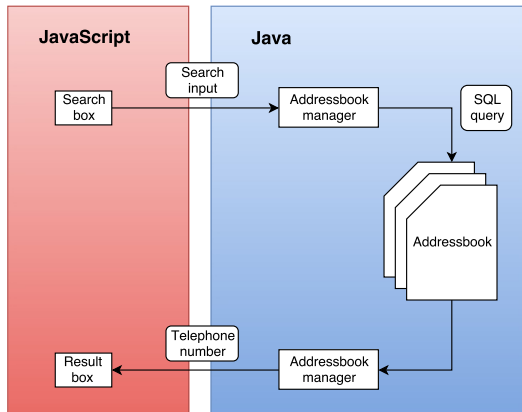
The Apache Cordova Framework for Android



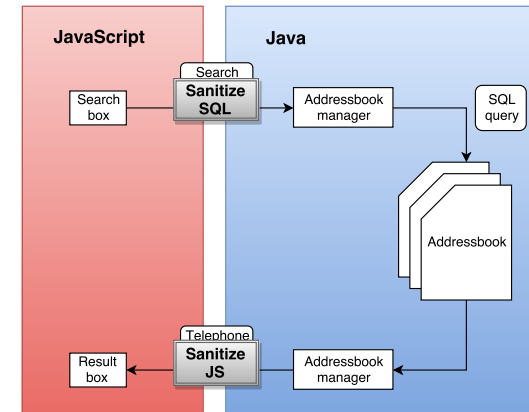
Example app



Technical view



Technical view



Example: Get Phone Number

```

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

```

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);
    }
}
    
```

Example: Get Phone Number

```

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

```

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);
    }
}
    
```

Example: Get Phone Number

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

```
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);
  }
}
```

Example: Get Phone Number

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

```
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);
  }
}
```

Example: Get Phone Number

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

```
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);
  }
}
```

Example: Get Phone Number

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

```
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);
  }
}
```

Example: Get Phone Number

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

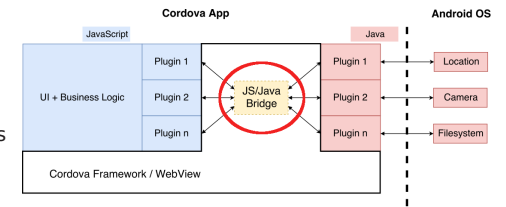
```
class ContactsPlugin extends CordovaPlugin {
    boolean execute(String action, CordovaArgs args, CallbackContext callbackContext) {
        if ("find".equals(action)) {
            String name = args.get(0).name;
            find(name, callbackContext);
        } else if ("create".equals(action))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

Outline

- 1 Motivation: Hybrid Mobile Apps and their Security Challenges
- 2 Real World Cordova Usage
- 3 Static Analysis for Hybrid Apps: Building a Unified Call Graph
- 4 Quality of the Unified Call Graph
- 5 Conclusions

First security assessment

- Problem: JS/Java Bridge is vulnerable to injection attacks
- For regular apps: Static Application Security Testing (SAST)
- But: No support for cross-language analysis
- **Our goal:**
Provide basis (call graph) to apply SAST to hybrid mobile apps



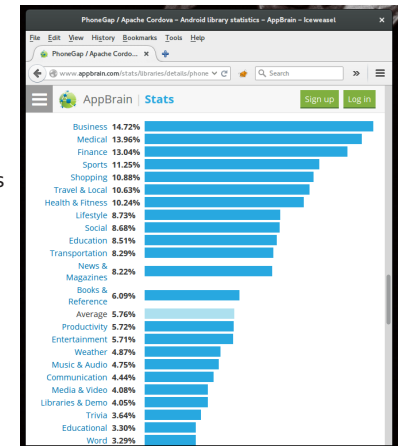
What we were interested in

Main goals:

- Understand the use of Cordova
- Learn requirements for Cordova security testing tools

Looking for answers for questions like

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



Test sets

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)
- one artificial test app (to test our tool)

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
- ...

But: Many different versions and some even modified!

Plugin

device	52%
inappbrowser	50%
dialogs	40%
splashscreen	36%
network-information	28%
file	28%
console	24%
camera	22%
statusbar	22%
PushPlugin	22%

What we have learned: app size

App size:

- mobile apps are not always small
- SAP apps seem to be larger than the average

Exceptional apps:

- No HTML/JS in APK
- Ship Cordova, but do not use it

App	Category	JS [kLoC]	Java [kLoC]
sap01	Finance	35.5	17.0
sap02	Business	345.3	53.5
sap03	Business	572.3	135.8
app01	Finance	26.3	17.8
app02	Finance	11.2	16.8
app03	Social	4.6	103.7
app04	Business	37.5	16.8
app05	Finance	20.0	44.8
app06	Finance	30.4	24.3
app07	Travel & Local	129.0	304.0
app08	Entertainment	36.7	23.0
app09	Lifestyle	36.3	44.7
app10	Finance	43.7	18.4
app11	Business	14.0	438.9
⋮	⋮	⋮	⋮

Outline

- 1 Motivation: Hybrid Mobile Apps and their Security Challenges
- 2 Real World Cordova Usage
- 3 Static Analysis for Hybrid Apps: Building a Unified Call Graph
- 4 Quality of the Unified Call Graph
- 5 Conclusions

Challenges

Based on the examined apps:

- Cordova relies heavily on dynamic mechanisms, both on JavaScript and Java side
- Developers modify their plugins and sometimes implement their own

Deep framework analysis

- Closest to the actual program
- But: Framework very expensive

Modelling framework

- Models the Cordova framework
- Analyses plugins

Modelling plugins

- Models both framework and plugins
- Analyses only UI and business logic part
- But: Developers can write own plugins

Our approach:

analyze plugins, but model the Cordova framework

- First build call graphs of Java and JavaScript separately
- Connect them using four heuristics that exploit frequent coding patterns:
 - ConvertModules
 - ReplaceCordovaExec
 - FilterJavaCallSites
 - FilterJSFrameworks
- **Result:**
Unified Call Graph

ConvertModules

```
define("com.foo.contacts", function(require, exports, module) {
  exports.find = function(successCallback, name) {
    cordova.exec(successCallback, null, "ContactsPlugin", "find", [{"name" : name}]);
  }
});
...
var successCallback = function(contact) {
  alert("Phone_number:_" + contact.phone);
}
plugins.contacts.find(successCallback, "Peter");
```

Problem:

- Not all callback functions are defined within the plugin
- Difficult to track callback functions from app code

Solution:

- Substitute dynamic mechanism with unique, global variable

ConvertModules

```
define("com.foo.contacts", function(require, exports, module) {
  plugins.contacts.find = function(successCallback, name) {
    cordova.exec(successCallback, null, "ContactsPlugin", "find", [{"name" : name}]);
  }
});
...
var successCallback = function(contact) {
  alert("Phone_number:_" + contact.phone);
}
plugins.contacts.find(successCallback, "Peter");
```

Problem:

- Not all callback functions are defined within the plugin
- Difficult to track callback functions from app code

Solution:

- Substitute dynamic mechanism with unique, global variable

ConvertModules: Results

- Most useful for
 - small plugins
 - more precise analysis
- Allows finding of callback functions in app code
- Less errors due to less ambiguity of dynamic mechanism

ReplaceCordovaExec

```
function showPhoneNumber(name) {
  var successCallback = function(contact) {
    alert("Phone_number:␣"+contacts.phone);
  }

  cordova.exec(successCallback, null, "ContactsPlugin", "find", [{"name" : name}]);
}
```

Problem:

- Callback call sites are hard to find
- No context-sensitivity

Solution:

- Stub the exec method

ReplaceCordovaExec

```
function showPhoneNumber(name) {
  var successCallback = function(contact) {
    alert("Phone_number:␣"+contacts.phone);
  }
  function stub1(succ, fail) {
    succ(null);
    fail(null);
  }
  stub1(successCallback, null, "ContactsPlugin", "find", [{"name" : name}]);
}
```

Problem:

- Callback call sites are hard to find
- No context-sensitivity

Solution:

- Stub the exec method

ReplaceCordovaExec: Results

- Necessary to find any Java to JavaScript calls
- Most apps use exec to communicate, only some bypass it
- Inexpensive way to get context-sensitivity where it is needed the most

FilterJavaCallSites

```

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);
    }
}
    
```

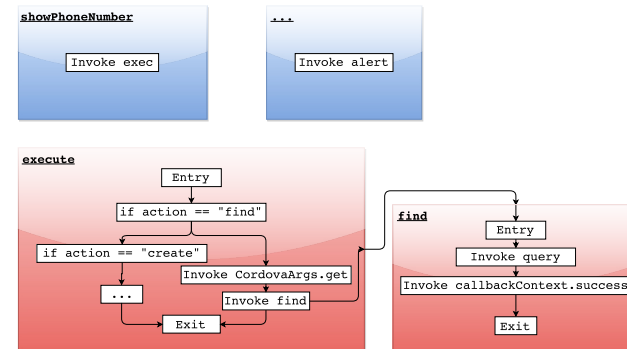
Problem:

- How to determine the targets of the callbackContext calls?
- Can we use the pattern of the action usage?

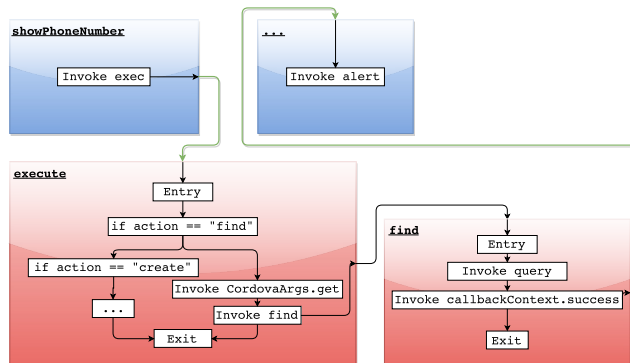
Solution:

- Determine which callbackContext calls are reachable

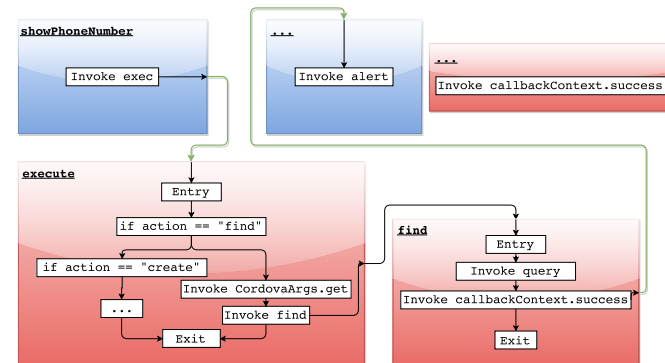
FilterJavaCallSites: details



FilterJavaCallSites: details



FilterJavaCallSites: details



FilterJavaCallSites: results

- Developers all use action variable similarly
- Therefore: Many incorrect edges avoided
- But: A few calls from Java to JavaScript are missed now
- Some store the callbackContext and call asynchronously

What we have learned: app size and cross-language calls

Cross-language calls:

- calls from Java to JS: very common
- calls from JS to Java: surprisingly uncommon

App	Category	Java2JS	JS2Java	JS [kLoC]	Java [kLoC]
sap01	Finance	2	12	35.5	17.0
sap02	Business	20814	39	345.3	53.5
sap03	Business	9531	75	572.3	135.8
app01	Finance	9	13	26.3	17.8
app02	Finance	2	10	11.2	16.8
app03	Social	2349	31	4.6	103.7
app04	Business	1	6	37.5	16.8
app05	Finance	6	26	20.0	44.8
app06	Finance	693	70	30.4	24.3
app07	Travel & Local	3430	43	129.0	304.0
app08	Entertainment	14220	67	36.7	23.0
app09	Lifestyle	51553	89	36.3	44.7
app10	Finance	8	36	43.7	18.4
app11	Business	0	0	14.0	438.9
⋮	⋮	⋮	⋮	⋮	⋮

Outline

- 1 Motivation: Hybrid Mobile Apps and their Security Challenges
- 2 Real World Cordova Usage
- 3 Static Analysis for Hybrid Apps: Building a Unified Call Graph
- 4 Quality of the Unified Call Graph
- 5 Conclusions

Recall and Precision

Recall:

Correctly reported calls
All reported calls

Precision:

Correctly reported calls
Calls actually present

App	kLoC	kNodes	Plugins	Recall	Precision	Calls
app01	43	9	5	33%	75%	17
app02	27	8	4	100%	66%	13
app03	106	18	8	1%	93%	61
app04	53	14	3	100%	100%	7
app05	64	10	7	33%	66%	29
app06	53	8	12	35%	97%	316
sap01	52	19	6	100%	66%	15
dvhma	17	7	4	100%	100%	15

Outline

- 1 Motivation: Hybrid Mobile Apps and their Security Challenges
- 2 Real World Cordova Usage
- 3 Static Analysis for Hybrid Apps: Building a Unified Call Graph
- 4 Quality of the Unified Call Graph
- 5 Conclusions

Thank you for your attention!

Any questions or remarks?

Summary

- Hybrid mobile apps are getting more popular
 - they are recommended at SAP
- Hybrid mobile apps are juicy targets
 - E.g., gain access to the app via the JS part . . .
 - . . . and use the app's permissions to steal data
- **Unified Call Graph** is a first step in bringing the full power of SAST to hybrid apps
- Quality largely depends on used call graph builders
- Future work: Data-flow analysis on top of Unified Call Graph

Bibliography



Achim D. Brucker and Michael Herzberg.

On the static analysis of hybrid mobile apps: A report on the state of apache cordova nation.

In Juan Caballero and Eric Bodden, editors, *International Symposium on Engineering Secure Software and Systems (ESSoS)*, Lecture Notes in Computer Science. Springer-Verlag, 2016.