RL-003

ハイブリッドアプリケーションの脆弱性に関する分析

Vulnerability Analysis of Hybrid Applications

西村 宗晃‡ 熊谷 裕志† 奥山 謙‡ 戸田 洋三† 久保 正樹†
Muneaki Nishimura Hiroshi Kumagai Ken Okuyama Yozo Toda Masaki Kubo

あらまし

近年、モバイルアプリケーション（以下、アプリ）の市場では、プラットフォーム固有のプログラム言語（以下、ネイティブ言語）で全体を書く代わりに中核部を HTML5 や JavaScript で作る、ハイブリッドアプリケーション（以下、ハイブリッドアプリ）と呼ばれるアプリが増えている。ハイブリッドアプリの脆弱性に関連する研究は既にいくつか存在するが[3] [4] [5]、これらの研究はアプリ開発者が取りうる対策を網羅的にまとめたものではなく、作り込まれうる脆弱性の全体像を提示できていない。

本稿では、ハイブリッドアプリフレームワーク（以下、フレームワーク）の一つである Apache Cordova（以下、Cordova）を用いて開発された、Android と iOS のハイブリッドアプリに作り込まれうる脆弱性の全体像を、アプリの構成要素 （OS、WebView、Cordova、プラグイン、アプリ固有のコード、ネットワーク通信、サーバ） に基づいて分析する。この分析作業では、各コンポーネントにこれまで発見された脆弱性、先行研究で指摘された脆弱性、著者らの独自調査により発見した新規の脆弱性を可能な限り網羅し、その影響と対策方法を提示した。そして、アプリ開発者自身が書くコードによって作り込まれる脆弱性と、それ以外のプラットフォーム等に起因する脆弱性の区別を明確にした。また、分析の結果得られた脆弱性の影響範囲を把握するため、アプリマーケットで公開されているアプリの調査を行った。

1. はじめに

近年のスマートフォンの OS では Android と iOS が市場シェアをほぼ二分している。Kanter[1]の調査によれば、2015 年 2 月時点の国内シェアは Android が 47.9%、iOS が 49.8%と拮抗している。アプリ開発者としては、より多くのユーザにアプリを提供するために両方の OS への対応が必要となる。しかし、Java や Objective-C といった異なるネネイティブ言語による二重の実装には費用がかかるため、開発を共通化するための技術が求められてきた。

このような状況で注目を集めているのがハイブリッドアプリである。ハイブリッドアプリは Web の標準技術である HTML5 や JavaScript を使用して中核部を開発するため、Android と iOS の両方でソースコードの多くを共通化できる。通常、ハイブリッドアプリの開発には効率化のためにフレームワークが採用される。本稿執筆時点で最も人気のあるフレームワークは Cordova である。AppBrain の調査[2]によれば、Google Play に公開されている Android アプリの約 6.00%に Cordova が利用されている。

一方で、Cordova には 2014 年に IBM Security Systems の研究者らにより複数の脆弱性が指摘されている[3]。また、Georgiev らの研究[4]では、ハイブリッドアプリの開発者の多くが Cordova のセキュリティモデルを適切に利用していないことが明らかにされている。さらに Jin らの研究[5]によれば、ハイブリッドアプリでは従来の Web アプリと比べてクロスサイトスクリプティング（以下、XSS）の攻撃可能界面が広がることや、XSS が悪用された際に生じる被害が深刻になりやすいことが指摘されている。

これらの状況を踏まえ、本調査では Cordova を利用した Android と iOS のハイブリッドアプリの一般的な構造を模式化し、アプリの構成要素ごとにどのような脆弱性が作り込まれうるかを分析した。また、それらの脆弱性に対してアプリ開発者が実施できる対策を明らかにした。あわせて、Google Play に公開されている各カテゴリの無料アプリトップ 500 を対象に分析を行い、本調査で提示した脆弱性を含有しうるアプリがどの程度市場に流通しているかを調査した。

本調査は経済産業省委託事業「平成 26 年度サイバーセキュリティ経済基盤構築事業（サイバー攻撃等国際連携対応調整事業）」における「脆弱性関連情報等の流通及び製品開発者における対策活動の支援」の一部として行われたものである。

本稿は 2015 年 2 月時点の調査に基づいており、一部に現状との乖離がある。調査結果の詳細は、最新の状況をできるだけ反映した上で後日公表する。

2. 背景

ハイブリッドアプリは通常のアプリと同様にマーケットで配布され、Android や iOS 等の端末にインストールして使用される。HTML5 や JavaScript で書かれたアプリ固有の実装コードは、Android の WebView や iOS の UIWebView（以下、両者を WebView とする）の上で実行される。WebView はアプリの画面に Web ブラウザを埋め込むコンポーネントであり、表示された Web コンテンツをネイティブ言語で制御する仕組みを備えている。Cordova はこの仕組みを利用して、JavaScript とネイティブ言語のコードを橋渡しする機能（ブリッジ）を提供する。カメラやセンサの制御、ローカルファイルの操作といったデバイス固有の機能はブリッジを介して JavaScript の API として実装され、プラグインという形態で Cordova 本体と分けて配布されている。プラグインは Apache Software Foundation（以下、Apache）が運営する Cordova Plugin Registry[6]に登録されており、アプリ開発者はリストからプラグインを探し出し、無料で使用できる。必要な機能を提供するプラグインが見つかれば、アプリの開発者はネイティブ言語のコードを書くことなくアプリを実装できる。JavaScript とネイティブ言語の両方の知識があれば、プラグインはアプリ開発者自身でも作成できる。

†JPCERT コーディネーションセンター
‡ソニーデジタルネットワークアプリケーションズ

2.1. Cordova のセキュリティモデル

Cordova はホワイトリストという保護機能を備えている。ホワイトリストは、指定したオリジン以外から配信されたリソースを WebView に読み込ませないための仕組みであり、読み込みを許可するオリジンをアプリの開発者が設定ファイル（config.xml）で指定する。ホワイトリストで指定されたオリジンの HTML にはブリッジへのアクセスも許可される。このため、アプリの開発者がホワイトリストを適切に指定していない場合、不正なページからブリッジを操作され、プラグインを通じてデバイス固有の機能が悪用される恐れがある。ホワイトリストの初期値にはワイルドカード（"*"）が指定されており、全てのオリジンのリソースに対するアクセスが許可されている。このためアプリの開発者がホワイトリストの初期値を変更していない場合、アプリの操作中に意図せず悪意のあるリソースを読み込んでしまう可能性がある。また、広告配信モジュールをアプリに組み込むような場合には、アプリがアクセスする必要のあるオリジンがアプリ開発者には分からないため、ホワイトリストを絞り込むことができない。こうした運用面の問題点も Georgiev ら[4]によって指摘されている。

3. 想定される脆弱性とその対策

Cordovaを利用した一般的なAndroidとiOSのハイブリッドアプリは、図 1 のようなコンポーネントから構成されるものとして模式化できる。この構成に基づき、各コンポーネントにおいてどのような脆弱性が組み込まれうるかを考察する [1]。

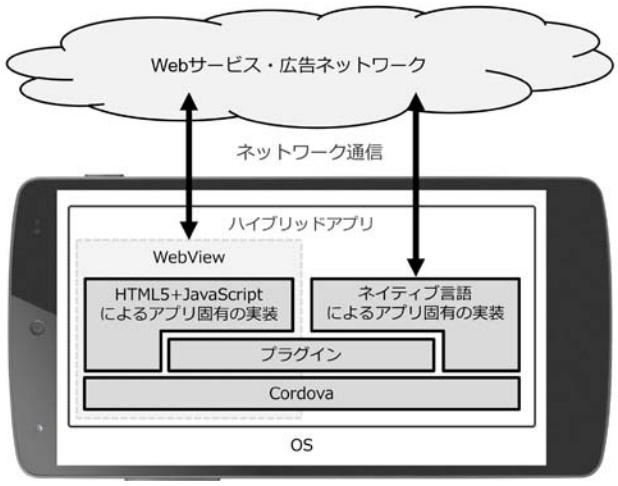

図 1: Cordova を利用したハイブリッドアプリの構成

本稿では調査結果の中でもハイブリッドアプリ固有の脆弱性を作り込みやすいと考えられる 4 つのコンポーネント（WebView、Cordova、プラグイン、アプリ固有実装）について、脆弱性の傾向とその対策を述べる。調査対象の Cordova のバージョンは、Android 版が 3.7.1、iOS 版が 3.7.0 である。

[1] 調査結果の詳細は後日公表予定である。

3.1. WebView

3.1.1. 脆弱性の概要

WebView には過去に多くの脆弱性が指摘されており、これらはハイブリッドアプリにも影響する。ハイブリッドアプリは主要機能の多くが WebView の上で実行されるため、通常のアプリよりも WebView の脆弱性を突く攻撃を受ける可能性が高い。個々の脆弱性の修正は OS のアップデートにより行われているため、ハイブリッドアプリの動作環境によってどの脆弱性の影響を受けるかは異なってくる。

例えば、Android 4.3 以下の WebView には、同一生成元ポリシーを回避して任意の Web ページの操作を可能とする脆弱性が指摘されており、発見者のブログでは攻撃コードも公開されている[8]。しかし、Google は Android 4.3 以下の WebView を今後サポートしないことを公言しており[9]、この脆弱性が公式に修正される可能性は低い。iOS においても、レンダリングエンジンの WebKit に起因する脆弱性が多数報告されているが、最新から一つ前のメジャーバージョンに当たる iOS7 にはアップデートが配信されていない[10]。

Android と iOS ともに WebView のアップデートを提供するサポート期間は明示されておらず、さらに Android では端末の製造元や機種によってもサポート期間が異なる。Google は今後、Google Play から WebView の更新を配信する方針を明らかにしている[7]が、対象となるのは Android 5.0 以上に限られる。一定期間を過ぎた iOS や 5.0 未満の Android は脆弱性を含んだ状態が続き、ハイブリッドアプリはその脆弱性の影響を受ける可能性がある。一方、脆弱性の修正されていない WebView の利用者は未だ多く存在しており、アプリ開発者としては、市場の規模を考えると古い OS に対するアプリの提供を打ち切ることは現実的に難しい。アクティブユーザの割合を見ると、Android 4.3 以前は 55.8%を占め[11]、iOS7 以下も 22%近くに上る[12]。

3.1.2. 対策

アプリ開発者が取りうる対策は、信頼できない HTML や JavaScript を WebView で開かないようアクセス制御することである。制御の実現手段の一つとして Cordova のホワイトリストの利用がある。しかし、ホワイトリストには現時点でも迂回を可能とする脆弱性が存在しており、また、data:スキームや file:スキームの URL のように、暗黙的にアクセス制限の対象外とされている URL も存在する。このため、ホワイトリストのみに依存したアクセス制御を行うことは禁物である。外部から与えられた URL を WebView で開かないようにする等、信頼できないリソースを開かないようにアプリ自体で対処することが重要となる。

また、Android ではサードパーティが提供する WebView ライブラリをアプリに組み込むことも対策の一つとなる。執筆時点で最も積極的に開発が行われているのは Intel がオープンソースで公開している Crosswalk[13] である。Crosswalk はベースに Chromium を流用しており、Google Chrome ブラウザの最新版に追従して改版されている。Android 4.0 以降の端末をサポートすることから、標準の WebView の代わりに使うことで既知の脆弱性の多くを回避できる。

しかし、Crosswalk にも幾つかの問題がある。1 つ目の問題は Cordova の正式なプラットフォームではないためプラグインの動作が保証されていないことである。2 つ目の問題は Chromium のログ出力を無効化できないことである。Android 4.0 では READ_LOGS パーミッションを使用宣言ればいかなるアプリでも他のアプリのログを取得できる。Android 4.0 に搭載されている標準の WebView とは異なり Crosswalk はログを抑止できないため、Android 4.0 ではログを通じて他のアプリに機密情報を盗まれる可能性がある。例えば、図 2 は HTTPS でホストされたページから HTTP のリソースを参照した際に表示される Mixed Content の警告ログである。この中にはユーザが開いているページの URL がクエリ文字列やフラグメントを含む形で詳細に記録されている。もし、URL にセッション ID やトークンなどの機密情報が含まれていた場合、他のアプリに漏洩する可能性がある。

```
I/chromium(32385): [INFO:CONSOLE(0)] "Mixed Content: The page at 'https://example.jp?q=SECRET#SECRET' was loaded over HTTPS, but requested an insecure prefetch resource 'http://example.jp/test.js'. This content should also be served over HTTPS."
```

図 2: Crosswalk の Mixed Content ログ

3.2. Cordova

3.2.1. 脆弱性の概要

Android の Cordova にはこれまでに 7 件の脆弱性が指摘されている（表 1）。その内訳は、ホワイトリストによるアクセス制限を迂回できるもの 6 件と、他のアプリから任意の JavaScript を実行させられるもの（CVE-2014-3500） 1 件である。前者の脆弱性として最も新しいのは WebSocket の通信にホワイトリストの制限が適用されないというもの[14]で、これは WebView の上で発生した WebSocket のリクエストをネイティブ言語で捕捉できない Android の制限に起因している。本稿執筆時点の最新版である Android 5.1 でもこの問題は修正されていない。後者の 1 件は Intent と呼ばれる Android のアプリ間通信の仕組みを用いて WebView に JavaScript を注入できるというものであり、Cordova のバージョン 3.5.1 で修正された。

iOS 版の Cordova には過去に脆弱性の指摘はされていないが、これは脆弱性が無く安全であることを意味するわけではない。本調査の過程で iOS 版の Cordova のホワイトリストによるアクセス制限を回避する方法を 3 通り発見しており、脆弱性として IPA に報告している。また、この他にも Cordova のホワイトリストの解析処理には実装の不備がある。2 階層以上のサブドメインにワイルドカードを含むオリジン（例えば、http://*.*.example.jp）をホワイトリストに指定した場合、ホワイトリストにワイルドカードを指定した場合と同様に全てのオリジンとの通信が許可されるという不正な動作となることを確認している。

3.2.2. 対策

アプリ開発者が取りうる対策は Cordova 本体のバージョンアップのみである。Apache が公開するリリースノートを定期的に確認し、脆弱性が公表された場合は速やかに Cordova の更新を適用してアプリをバージョンアップする。しかし、Cordova の更新時には後方互換を損なうような仕様変更が含まれることがあるため注意が必要である。Android 版の Cordova 3.7.1 では Content Security Policy（以下、CSP）準拠の一環としてネイティブ側から任意の JavaScript を WebView に送信することのできる sendJavascript という API が廃止された。一部のプラグインはネイティブ側の状態変化を WebView に通知する手段としてこの API を使用しているため、Cordova を更新すると動作しなくなる。Cordova をアップデートする際にはこうしたリスクに対処すべく余裕を持った開発計画を立て、リグレッションテストを忘れずに行うことが重要である。

表 1: Cordova の脆弱性一覧

CVE	概要
2014-3500	Apache Cordova on Android Crafted URL Handling Start Page Manipulation
2014-3501	Apache Cordova on Android WebView Component JavaScript Launched WebSocket Connection Handling Whitelist Restriction Bypass
2014-3502	Apache Cordova on Android Intent URL Redirection Local Information Disclosure
2012-6637	Apache Cordova / Adobe PhoneGap on Android Domain Substring Matching Whitelisting Bypass Weakness
2014-1882	Apache Cordova / Adobe PhoneGap on Android iframe Content Domain Whitelisting Bypass
N/A	Apache Cordova / Adobe PhoneGap on Android / iOS Same-origin Policy Bypass
2014-1884	Apache Cordova / Adobe PhoneGap on Windows Phone Crafted Content Request Device-resource Restriction Bypass
2014-1881	Apache Cordova / Apache PhoneGap IFRAME Script OnJsPrompt Handler Device-resource Restriction Bypass

3.3. プラグイン

3.3.1. 脆弱性の概要

プラグインの脆弱性はこれまでに 3 件報告されており（表 2）、いずれも同じ研究者によるものである。本稿執筆時点では、Cordova Plugin Registry[6]には 898 種類のプラグインが登録されている。その中の 3 つのプラグイン以外には脆弱性は存在しないのだろうか。本調査では Cordova Plugin Registry に掲載されているダウンロード数上位 20 位のプラグインを対象に簡易的な動的検査を行い、脆弱性の有無を確かめた。その結果、Android のプラグインに 1 件の脆弱性を発見した。この結果から、プラグインにはまだ潜在的な脆弱性が存在しており、今後、Cordova が普及して多くのセキュリティ研究者の関心が集まった場合、より多くの脆弱性が見つかることが懸念される。

表 2: Cordova プラグインの脆弱性一覧

CVE	脆弱性の概要
2014-0073	Apache Cordova In-App-Browser iOS Plugins Unspecified XSS
2014-0072	Apache Cordova In-App-Browser iOS Plugins trustAllHosts Setting Weakness
N/A[17]	Cordova LaunchMyApp Plug-in Remote JavaScript Injection

プラグインは JavaScript とネイティブ言語の実装を併せ持つことから、理論的には Web アプリとネイティブアプリの双方の脆弱性を作り込む可能性がある。特にプラグインにおいて作り込みやすい脆弱性としては、JavaScript インジェクションが挙げられる。Android で Pending Intent を利用して通知バーやアラームから指定された URL を WebView 上で開く仕様のプラグインを開発した場合、他のアプリから javascript スキームの URL を送信され、不正な JavaScript を実行させられる可能性がある。

3.3.2. 対策

第三者が提供するプラグインを利用する場合は、そのプラグインの開発者が提供するリリースノートを定期的に確認し、脆弱性が公表された場合は速やかにプラグインを更新する必要がある。自身でプラグインを開発する場合は、Android や iOS のセキュアコーディングの作法を習得し、プラットフォーム特有の脆弱性を作り込まないように注意する必要がある。Android の場合、日本スマートフォンセキュリティ協会が無償で配布する「Android アプリのセキュア設計・セキュアコーディングガイド」[18]が参考になる。一方、iOS については、Apple が提供する「Secure Coding Guide」[19]が参考になる。

3.4. HTML＋JavaScript のアプリ固有実装

3.4.1. 脆弱性の概要

アプリ開発者が HTML5 や JavaScript によるコードを開発する際に作り込みうる脆弱性は、大きく分けて 3 つある。1 つ目はプラグインの誤用に起因する脆弱性。2 つ目は HTML5 特有の脆弱性。そして 3 つ目は DOM Based XSS である。以下、順に考察する。

プラグインの誤用に起因する脆弱性　Cordova のプラグインの中には利用の仕方を誤るとアプリの脆弱性につながるものがある。例として、File-Transfer プラグインの不適切な使用に起因する 2 つの脆弱性について述べる。

File-Transfer プラグインは端末のローカルファイルをサーバとの間で送受信するプラグインである。プラグインが定義する upload 関数を実行することにより、fileURL という引数に指定したローカルファイルを HTTP や HTTPS でサーバに送信できる。upload 関数には trustAllHosts というブール型の引数があり、true を指定すると HTTPS のサーバ接続時にサーバ証明書検証が行われなくなる。trustAllHosts は開発時のみの利用を想定して作られたものであり、これを指定したままリリースした場合、サーバ証明書の検証不備の脆弱性をアプリに作り込んでしまう。また、upload 関数の fileURL にはサーバへ送信するローカルファイルを file スキームの URL で指定するが、ここで、'file:///sdcard/' + filename というように前方のパスは固定で、ファイル名（filename）のみを外部から指定できる仕様とした場合、攻撃者は filename に../や..%2f などの文字を含めることで上位のディレクトリに遡り、任意のディレクトリにあるファイルをサーバに送信させることができる。

HTML5 特有の脆弱性　近年標準化が議論されている機能の多くは、セキュリティやプライバシー上の懸念とその対策が仕様策定時に議論されており、利用者が意識せずとも安全に動作するものが多い。しかしながら、中にはアプリの開発者が注意しなければならない機能も存在する。例えば、HTML5 のデータベース API の 1 つである Web SQL Database は任意の SQL クエリを実行する executeSql という機能をサポートしている。この API はバインド機能をサポートしているが、開発者がこれを用いず、外部から取得した文字列を結合して SQL クエリ文字列を生成している場合、SQL インジェクションの脆弱性を作り込む可能性がある。Web SQL Database は既に廃案となり、代わりに Indexed Database API を使うことが推奨されている。しかし、Android 4.3 以下や iOS の WebView は Indexed Database API をサポートしないため、構造化されたデータを扱う際には依然として Web SQL Database が利用される。

DOM Based XSS　DOM Based XSS は XSS の一種で、外部から指定された文字列を基に JavaScript で動的に HTML を生成する際に意図せぬスクリプトが混入する脆弱性である。一般的な Web サイトでは、サーバから取得した JSON 形式のデータや HTML の入力フォームから受け取る文字列等が主な攻撃可能界面であった。しかし、ハイブリッドアプリは画像や音声ファイルのメタ情報、SMS、電話帳、Wi-Fi や Bluetooth のアクセスポイント名、端末のローカルファイル名といった様々な文字列を扱う可能性があり、これらが新たな攻撃可能界面となることが Jin らにより指摘されている[5]。iOS 版の Skype アプリでは、ユーザ名に JavaScript を含めることにより、チャット相手のアプリで任意のコードを実行させることのできる脆弱性が過去に指摘されているが[15]、同様の脆弱性がハイブリッドアプリにも作り込まれる可能性がある。

ハイブリッドアプリ上で XSS による攻撃が行われた場合に生じる被害はアプリによって様々であるが、アプリに共通して考えられる被害として、組み込まれているプラグインの不正利用が挙げられる。例えば Contacts プラグインを組み込んでいるアプリの場合、XSS 攻撃によって攻撃者のサーバに電話帳の情報を盗み出される可能性がある。Android 版の Cordova は、setAllowUniversalAccessFromFileURLs(true)という指定により file スキームのコンテキストで動作する JavaScript から任意のローカルファイルや全てのオリジンに対する制限の無いアクセスを許可している。このため、XSS の脆弱性が悪用された場合、WebView のキャッシュファイルに含まれるパスワードなどの機密情報が盗み出されたり、同一生成元ポリシーの制限を受けず、他の任意のサイトに JavaScript を注入されたりする恐れがある。iOS でも、iOS7 以前は file スキームのコンテキストで動作する JavaScript

からアプリのサンドボックス内にある任意のファイルを盗み出すことができた。iOS6 以前では、さらに電話帳をはじめとするシステム領域のファイルへのアクセスも可能であった。

3.4.2. 対策

3 つの脆弱性に共通する対策は、アプリの開発者がセキュアコーディングの知識を習得し実践することである。OWASP が公開する「HTML5 Security Cheat Sheet」[20]や JPCERT/CC が発行する調査報告書[16]は、HTML5 特有の脆弱性に対する情報源となる。しかし一方で、近年のブラウザの新機能追加にこうしたガイドは追従できておらず、継続的な更新が望まれる。

Cordova プラグインの利用誤りについても同様であり、プラグインの使い方や作法を記述したガイドライン等の位置付けの情報源は存在しておらず、ガイドの登場が望まれる。

DOM Based XSS に対しては、アプリの外部から入力された信頼できないデータを HTML に埋め込む際、文脈（コンテキスト）に応じたエスケープ処理を行うことが有効な対策となる。しかし、コンテキストを見極めて適切なエスケープ処理を施すことは熟練した開発者でも難しい。例えば図 3 の例では、${data1}～${data4}のそれぞれで別々のエスケープ処理が必要となる。特に${data4}の属性値は任意の HTML マークアップを埋め込むことができるため、XSS の原因となる要素や属性値を見極め、安全な HTML マークアップのみを抽出する必要がある。

```
<div onmouseover=”${data1}”> ${data2} </div>
<a href=”${data3}” >link</a>
<iframe srcdoc=” ${data4}”></iframe>
```

図 3: DOM Based XSS の発生するコンテキスト

このため、セキュアコーディングの教育による開発者の知識向上だけで DOM Based XSS を防ぐことは難しい。多層防御として、DOM Based XSS の発生を抑止する仕組み備えた JavaScript フレームワークや、ブラウザの CSP を組み合わせてリスクを軽減することが望ましい。また、XSS の脆弱性が悪用された場合の被害を軽減するために、不要なプラグインを組み込まないことが望ましい。

4. 市場のハイブリッドアプリに対する調査

Google Play に掲載されている各カテゴリの無料アプリトップ 500 位までのアプリ全 13,488 個を対象に静的解析を行い、これまでに述べた脆弱性の影響を受けるアプリが市場に存在する割合を調査した。調査対象のアプリのランキングは 2015 年 1 月 20 日時点のもので、対象国は日本、対象機種は Google の Nexus 5 とした。

4.1. Cordova を使用するアプリ

Cordova を使用するアプリは 13,488 個中 361 個、全体の約 2.7%であった。これは AppBrain による市場調査結果[2]の 6.00%の約半数に当たる。この乖離が生じた理由は、海外と比べて日本国内ではまだ Cordova が普及していないためであると推察される。

4.2. Android 4.3 をサポートするアプリ

Cordova を使用するアプリのうち Android 4.3 以前をサポートするものは 361 個中 360 個あり、ほぼ全数に上った。また、Crosswalk を組み込んだアプリはこの中の 7 個に過ぎず、残りの 353 アプリは Google がサポートを終了した WebView を利用している。したがって、ユーザが利用する Android 端末のバージョンが 4.3 以下である場合、これらのアプリのほぼ全てが 3.1.1 に述べた同一生成元ポリシー回避の脆弱性の影響を受け、悪意のあるサイトを参照するリンクをクリックするだけで任意のコードを実行されてしまう可能性がある。

4.3. Cordova 3.5.0 以下を使用するアプリ

Android 版の Cordova のバージョン 3.5.0 以下には、外部アプリが任意の JavaScript を実行できる危険度の高い脆弱性（CVE-2014-3500）が存在することが指摘されている。取得したアプリのうち、バージョン 3.5.0 以下の Cordova を使用するものは 361 個中 129 個あり、約 35.7%のアプリがこの脆弱性の影響を受ける可能性がある。

4.4. ホワイトリストの指定が不適切なアプリ

config.xml のホワイトリスト指定でワイルドカードを含むアプリは 361 個中 294 個あり、Cordova を使用するアプリ全体の 81.4%に上った。本結果から、多くの開発者がホワイトリストを適切に運用していないことが分かる。

4.5. File-Transfer プラグインを誤用するアプリ

File-Transfer プラグインの download 関数や upload 関数の引数に trustAllHosts=true を指定しているアプリが 2 個存在した。これらは実際にプラグインを用いて HTTPS のホストと通信する場合、サーバ証明書検証不備の脆弱性を含有することになる。Android のネイティブアプリにおいては、サーバ証明書検証不備の脆弱性を作り込んでいるアプリが 2 万件以上存在することが指摘されているが[21]、同様の脆弱性はプラグインの誤用によって容易に作り込まれる可能性がある。

5. まとめ

本調査では、一般的な Cordova アプリの構成要素ごとに混入しうる脆弱性の傾向を調査し、アプリ開発者が講じることのできる対策を明らかにした。今後は、ハイブリッドアプリの開発者に対して、リスクの周知と、本稿で考察した対策方法の普及促進が必要と考えられる。また一方で、本稿で述べた課題を解決すべくフレームワークを拡張して、安全性を保証する範囲を広げ、開発者が脆弱性を作り込みにくいアプリの開発基盤として発展することが期待される。

参考文献

[1] Kantar Worldpanel, Smartphone OS market share, http://www.kantarworldpanel.com/global/smartphone-os-market-share/ （2015 年 4 月 4 日閲覧）

[2] AppTornado GmbH, AppBrain, http://www.appbrain.com/stats/libraries/details/phonegap/phonegap-apache-cordova （2015 年 4 月 4 日閲覧）

[3] IBM, Security Intelligence, Apache Cordova Vulnerability Discovered: 10% of Android Banking Apps Potentially Vulnerable, http://securityintelligence.com/apache-cordova-phonegap-vulnerability-android-banking-apps/
[4] M. Georgiev, S. Jana, and V. Shmatikov. "Breaking and fixing origin-based access control in hybrid web/mobile application frameworks." In *Proceeding of the Network Distributed System Security Symposium (NDSS)*, 2014.
[5] X. Jin, X. Hu, K. Ying, W. Du, H. Yin and G. N. Peri. "Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation" In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. *ACM*, 2014.
[6] Apache Software Foundation, Cordova Plugin Registry, http://plugins.cordova.io/ （2015 年 4 月 4 日閲覧）
[7] Google, Android Lollipop | Android Developers, https://developer.android.com/about/versions/lollipop.html#WebView
[8] Rafay Baloch, Rafay Hacking Articles, Android Browser Same Origin Policy Bypass < 4.4 - CVE-2014-6041, http://www.rafayhackingarticles.net/2014/08/android-browser-same-origin-policy.html
[9] Google+, Adrian Ludwig, https://plus.google.com/+AdrianLudwig/posts/1md7ruEwBLF
[10] Apple, Apple security updates, https://support.apple.com/en-lb/HT201222 （2015 年 4 月 4 日閲覧）
[11] Google, Dashboards | Android Developers, https://developer.android.com/about/dashboards/ （2015 年 4 月 4 日閲覧）
[12] Apple, App Store Distribution - Support - Apple Developer, https://developer.apple.com/support/appstore/ （2015 年 4 月 4 日閲覧）
[13] Intel Open Source Technology Center, The Crosswalk Project, https://crosswalk-project.org/
[14] MITRE, CVE-2014-3501, http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-3501
[15] SUPEREVR, XSS in Skype for iOS, https://superevr.com/blog/2011/xss-in-skype-for-ios
[16] JPCERT コーディネーションセンター, HTML5 を利用した Web アプリケーションのセキュリティ問題に関する調査報告書
[17] Neil Bergman, Cordova LaunchMyApp Plug-in Remote JavaScript Injection, http://d3adend.org/blog/?p=426 （2015 年 4 月 13 日閲覧）
[18] 一般社団法人日本スマートフォンセキュリティ協会, Android アプリのセキュア設計・セキュアコーディンググガイド （2014 年 7 月 1 日版）
[19] Apple, Secure Coding Guide, https://developer.apple.com/library/ios/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html （2015 年 4 月 13 日閲覧）
[20] OWASP, HTML5 Security Cheat Sheet, https://www.owasp.org/index.php/HTML5_Security_Cheat_Sheet （2015 年 4 月 13 日閲覧）
[21] CERT/CC, Vulnerability Note VU#582497 Multiple Android applications fail to properly validate SSL certificates, http://www.kb.cert.org/vuls/id/582497 （2015 年 4 月 13 日閲覧）