# Cordovaを利用したハイブリッドアプリケーションにおけるプラグインのアクセス制御方式

工藤 直樹[1]　山内 利宏[1,a)]

**概要**：モバイルアプリケーションの開発において，ハイブリッドアプリケーションが増加している．ハイブリッドアプリケーションフレームワークの 1 つである Cordova では，デバイスの資源を利用するために，プラグインと呼ばれるインタフェースを提供している．本研究では，悪意のある攻撃者がリパッケージと呼ばれる手法を用いて，プラグインを悪用する JavaScript コードを Cordova アプリに挿入し，デバイスの資源の奪取や改ざんを行う攻撃について述べる．また，これらの攻撃を防止するため，プラグインによるデバイスの資源へのアクセスを動的に制御する方式を提案し，その設計，実現方式，および評価結果について述べる．

**キーワード**：ハイブリッドアプリケーション，Android，アクセス制御

## 1. はじめに

近年，モバイルアプリケーションの開発において，ハイブリッドアプリケーション（以降，ハイブリッドアプリ）が増加している．ハイブリッドアプリは，従来のアプリと比較して，Android における Java，iOS における Objective-C や Swift といったプラットフォーム固有の言語（以降，ネイティブ言語）の実装が少なく，アプリの大部分を HTML や JavaScript といったプラットフォームに依存しない言語を用いて開発される．このため，複数のプラットフォームでアプリのコードの大部分を共有できるという利点がある．また，ハイブリッドアプリは，HTML や JavaScript を利用するために，WebView 上で実行される．

ハイブリッドアプリは，JavaScript とネイティブ言語のコードを相互通信するための機能であるブリッジを利用することにより，JavaScript 側からデバイスの資源にアクセスできる．ハイブリッドアプリの開発には，ハイブリッドアプリフレームワークを利用することが多く，人気のあるハイブリッドアプリフレームワークの 1 つとして Cordova がある．[1] Cordova を利用したハイブリッドアプリ（以降，Cordova アプリ）がブリッジを利用してデバイスの資源へアクセスする場合，プラグインと呼ばれるインタフェースを用いる．

本研究では，悪意のある攻撃者がリパッケージと呼ばれる手法を用いて，プラグインを悪用する JavaScript コードを Cordova アプリに挿入し，デバイスの資源の奪取や改ざんを行う攻撃を新たに指摘する．また，本研究では，新たに指摘したリパッケージを用いた攻撃とクロスサイトスクリプティングを用いた攻撃 [2] に対処する．リパッケージを用いた攻撃やクロスサイトスクリプティングを用いた攻撃は，デバイスの資源にアクセスするために，プラグインを利用する必要がある．なお，文献 [3] では，Cordova アプリが読み込む URL が指す Web サイトに含まれる JavaScript コードによって，プラグインを悪用できる問題を提案したが，上記の問題は発生しないことがわかったため，この問題は本研究の対象としない．

本研究では，これらの攻撃を防止するため，プラグインによるデバイスの資源へのアクセスを動的に制御する方式を提案し，その設計と実現方式について述べる．文献 [3] では，URL とプラグインの組合せに基づくアクセス制御方式を提案したものの，本研究で対象とする問題では，URL に基づく制御は不要のため，プラグインのみに基づいたアクセス制御方式を提案する．提案方式の適用により，プラグインによるデバイス資源へのアクセスを利用者の判断によって制御できるようになり，利用者が Cordova アプリをより安全に利用できる．また，提案方式の適用による有用性と性能について評価した結果を報告する．評価結果から，提案方式の適用により，プラグインを悪用した攻撃が防止できることとそのオーバヘッドが小さいことを示す．なお、本研究は、Android 版の Cordova を対象とする．

[1] 岡山大学大学院自然科学研究科
[a)] yamauchi@cs.okayama-u.ac.jp

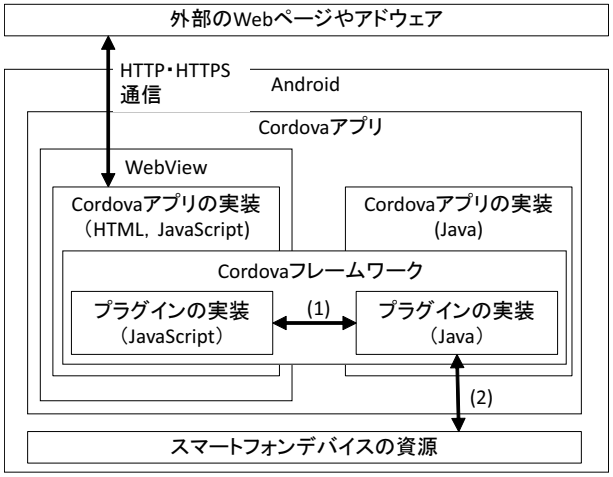


図 1　Android における Cordova を利用したハイブリッドアプリケーションの構成（参考：文献 [4]）

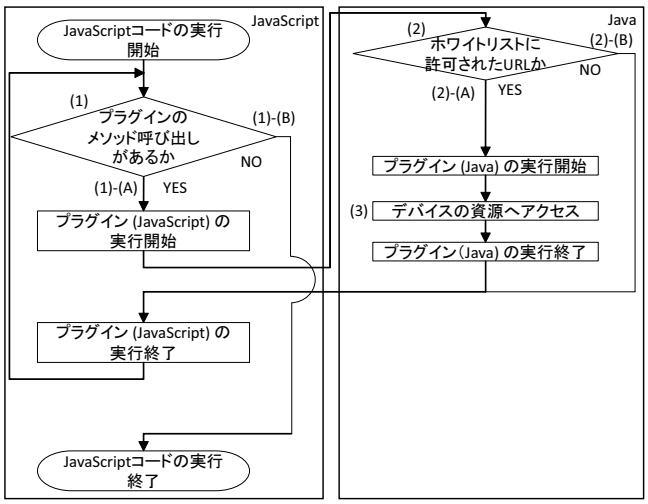


図 2　プラグインによるデバイス資源へのアクセスの流れ

## 2. 背景

### 2.1 Cordova を利用したハイブリッドアプリケーションの構成

#### 2.1.1 概要

文献 [4] を参考に，Android における Cordova アプリの構成を図 **1** に示す．図 1 のように，Cordova アプリは，プラグインと呼ばれるインタフェースを用いることにより，デバイスの資源へアクセスできる．Cordova アプリがプラグインを利用してデバイスの資源にアクセスする流れを以下で説明する．なお，ここでブリッジとは，JavaScript コードと Java コードを相互通信させるための機能である．

(1) ブリッジを用いることにより，JavaScript 側のプラグインから Java 側のプラグインへアクセスする．
(2) Java 側のプラグインからデバイスの資源へアクセスする．

#### 2.1.2 プラグイン

プラグインは，Cordova アプリがデバイスの資源にアクセスする際に用いるインタフェースである．プラグインは，JavaScript の実装と Java の実装に分かれる．JavaScript の実装では，Java のメソッドにアクセスするための JavaScript API が定義されている．一方，Java の実装では，JavaScript API によって呼び出される Java のメソッドが定義されており，Java のメソッドからデバイスの資源へアクセスが可能である．Cordova アプリはプラグインを読み込み，JavaScript API を利用することにより，JavaScript 側からデバイスの資源にアクセスできる．プラグインには，Cordova から提供されている Cordova コアプラグインとサードパーティから提供されているサードパーティプラグインの 2 種類が存在する．

Cordova アプリは，当該 Cordova アプリが読み込んでいるプラグインのみを使用できる．たとえば，カメラプラグインのみを読み込んでいる Cordova アプリは，カメラプラグイン以外のプラグインを使用できない．

### 2.2 プラグインによるデバイス資源へのアクセスの流れ

Android 版の Cordova において，Cordova アプリの JavaScript 側からプラグインによるデバイスの資源へのアクセスの流れを図 **2** に示し，その詳細を以下で述べる．

(1) JavaScript コードに，使用するプラグインのメソッド呼び出しが存在するか否かを確認する．
  (A) プラグインのメソッド呼び出しが存在する場合 JavaScript 側のプラグインの実行を開始し，JavaScript API を呼び出す．
  (B) プラグインのメソッド呼び出しが存在しない場合 JavaScript コードの実行を終了する．
(2) プラグインのメソッド呼び出し元の URL がホワイトリストに許可された URL であるか否かを確認する．
  (A) ホワイトリストに許可された URL である場合 Java 側のプラグインの実行を開始し，JavaScript API に対応する Java のメソッドを呼び出す．
  (B) ホワイトリストに許可された URL でない場合 JavaScript 側のプラグインの実行を終了する．
(3) Java のメソッドからデバイスの資源にアクセスする．その後，Java 側のプラグインの実行と JavaScript 側のプラグインの実行を終了する．

また，(2) において，プラグインのメソッド呼び出しを行う URL がホワイトリストに許可された URL であるか否かを確認する理由は，Cordova アプリ以外の外部の Web ページやアドウェアの URL によって，不正にブリッジを利用される可能性を想定しているためである．

### 2.3 Cordova を利用したハイブリッドアプリケーションにおける問題点

プラグインを利用することによって，Cordova アプリは JavaScript 側からデバイスの資源にアクセスできる．これ

により，Android や iOS といった異なるプラットフォーム間における Cordova アプリによるデバイスの資源の利用が容易となる．一方，悪意のある攻撃者がプラグインを悪用した場合，JavaScript 側からデバイスの資源の奪取や改ざんが可能となってしまう．

## 3. プラグインを悪用した攻撃

### 3.1 攻撃モデル

2.3 節で述べたように，悪意のある攻撃者がプラグインを悪用した場合，JavaScript 側からデバイスの資源の奪取や改ざんが可能となってしまう．ただし，攻撃者が悪用できるプラグインは，Cordova アプリに読み込まれているプラグインに制限される．文献 [2] や文献 [5] では，プラグインを悪用することにより，デバイスの資源の奪取や改ざんをされる恐れがあることについて述べられている．Cordova アプリにプラグインを悪用する JavaScript コードを挿入される経路としては，以下の 2 つが考えられる．

(1) Cordova アプリのリパッケージ
パッケージ済みのアプリを改ざんし，再パッケージする行為であるリパッケージを用いて，Cordova アプリに悪意のある JavaScript コードを挿入できる．なお，Cordova アプリのリパッケージは，一般的な Android アプリのリパッケージと比較して，悪意のあるコードの挿入が容易である．

(2) クロスサイトスクリプティング攻撃
文献 [2] で述べられている手法であり，Wi-Fi へのアクセスや QR コードの読み込みを契機として，クロスサイトスクリプティング攻撃を行い，Cordova アプリに悪意のある JavaScript コードを挿入できる．

なお，本研究では，悪意のある JavaScript コードが挿入可能である問題には対処せず，プラグインを悪用する JavaScript コードの挿入により，デバイスの資源の奪取や改ざんをされるという問題にのみ対処する．以降では，我々が新たに指摘する攻撃手法である Cordova アプリのリパッケージを用いた攻撃について述べる．

### 3.2 Cordova を利用したハイブリッドアプリケーションにおけるリパッケージを用いた攻撃

#### 3.2.1 Android 版の Cordova を利用したハイブリッドアプリケーションにおける apk ファイルの構成

リパッケージを用いた攻撃の流れを示す前に，Android 版の Cordova アプリにおける apk ファイルの構成について述べる．構成のうち，重要なファイルを図 **3** に示し，以下で詳細を述べる．

(1) /assets/www/以下のファイル
Cordova アプリの HTML や JavaScript のソースコードが保存されている．これらのコードは暗号化されていない場合，攻撃者に閲覧される可能性がある．

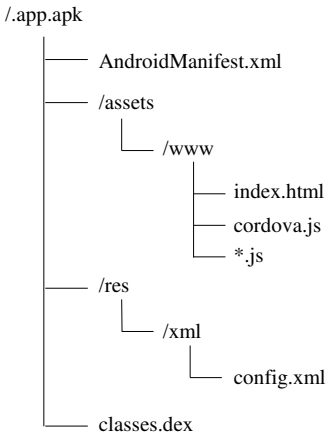


**図 3** Android 版の Cordova を利用したハイブリッドアプリケーションにおける apk ファイルの構成

#### 3.2.2 リパッケージを用いた攻撃の流れ

Cordova アプリにおけるリパッケージを用いた攻撃の流れを以下に示す．

(1) apk ファイルを展開
(2) /assets/www/以下の index.html または js ファイルを改変し，悪意のある JavaScript コードを挿入
(3) 改変したソースコードを含めて，apk ファイルをリパッケージ

(2) の改変の際に，プラグインを悪用する JavaScript コードを挿入した場合，デバイスの資源の奪取や改ざんが可能となってしまう．

#### 3.2.3 Android アプリケーションにおけるリパッケージを用いた攻撃との比較

一般的にリパッケージを用いた攻撃は，Cordova アプリではなく，Android アプリが対象にしていることが多い．リパッケージを用いて，Android アプリに悪意のあるコードを埋め込む場合，dex2jar や Java Decompiler といったツールを利用して，classes.dex から抽出した Java のバイトコードである class ファイルを改変する必要がある．また，ProGuard といった Java のバイトコードを難読化するツールの普及により，Android アプリにおけるリパッケージを用いた攻撃は徐々に困難になっている．

一方，Cordova アプリのリパッケージによる攻撃の場合，/assets/www/以下に保存されている index.html や js ファイルといったソースコードを改変する．また，現在のところ，Android アプリと比較して Cordova アプリの難読化ツールはあまり普及していないと考えられる．このため，Cordova アプリのリパッケージは，Android アプリのリパッケージと比較して，悪意のあるコードの挿入が容易である．

### 3.3 考察

3.1 節で述べたプラグインを悪用した攻撃は，プラグイ

ンを用いてデバイスの資源にアクセスすることにより，資源の奪取や改ざんを行う．プラグインを用いてデバイスの資源にアクセスする場合，Android パーミッションが要求される．Android 6.0 未満の場合，Android パーミッションはアプリのインストール時に要求され，アプリの起動時には，パーミッションを設定できない．一方，Android 6.0 以降の場合，Dangerous Permission グループに属するパーミッションは，アプリの起動時に要求される．このため，Android 6.0 以降で，Dangerous Permission グループに属するパーミッションが必要となるプラグインはデバイス資源へのアクセス前にパーミッションが要求されるため，攻撃を事前に検出可能である．

しかし，Android 6.0 以降において，アプリの起動時にパーミッションを要求するためには，AndroidManifest.xml にて定義される targetSdkVersion を 23 以上に設定する必要がある．文献 [6] では，2015 年 12 月時点での Android アプリ 60,086 個を調査した結果，targetSdkVersion が 23 未満の Android アプリが 93%存在したと述べられている．さらに，3.1 節で述べたプラグインを悪用した攻撃の対象となる Cordova アプリは，攻撃を達成するために，targetSdkVersion が 23 未満に設定されている可能性が高いと考えられる．このため，プラグインを悪用した攻撃を確実に防止するためには，プラグインによるデバイスの資源へのアクセスを制御する必要がある．

## 4. 提案方式

### 4.1 目的と考え方

提案方式は，3.1 節で述べたプラグインを悪用した攻撃を防止するため，プラグインを悪用した JavaScript コードによるデバイスの資源へのアクセスを防止することを目的とする．提案方式の目的を達成するために，利用者の判断に基づき，プラグインによるデバイスへのアクセスを動的に制御する．上記によって，プラグインを悪用した攻撃を受ける可能性のある脆弱な Cordova アプリを利用者が使用していた場合においても，プラグインによるデバイスの資源へのアクセス前に，利用者の判断に基づくアクセス制御が可能である．

ハイブリッドアプリのアクセス制御の研究として，文献 [5]，NoFrak[7]，文献 [8] があるが，これらは利用者による制御を考慮しておらず，プラグインを悪用した攻撃に対処できない．また，利用者による制御を考慮した研究として MobileIFC[9] があるが，MobileIFC の適用のためには，独自のフレームワークを組み込む必要があり，Cordova アプリのコードの修正量が多く，適用が困難である．

提案方式は，プラグインを悪用した攻撃に対処可能であり，Cordova フレームワークの修正のみで提案方式を実現できる．また，Cordova アプリのコードの修正が不要であるため，MobileIFC と比較して，適用が容易である．

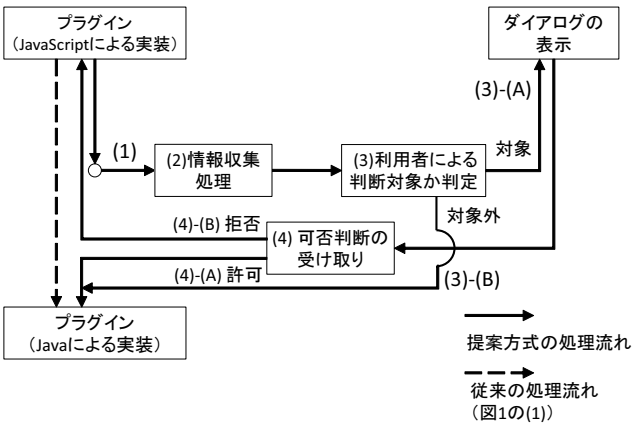


図 4　提案方式の処理流れ

### 4.2 全体像

提案方式の処理流れを図 4 に示し，その詳細を以下で述べる．

(1) Java 側のプラグインにアクセスするメソッドをフック
(2) フックしたメソッドから利用者に提示するために必要な情報を取得
(3) 利用者による判断対象か否かを判定
  (A) 判断対象であった場合，ダイアログを表示
  (B) 判断対象でない場合，Java 側のプラグインの実行を開始し，デバイスの資源にアクセス
(4) 利用者の判断結果を受け取った後，利用者の判断に従い，プラグインを制御
  (A) デバイス資源へのアクセスが許可された場合，Java 側のプラグインの実行を開始し，デバイスの資源にアクセス
  (B) デバイス資源へのアクセスが拒否された場合，JavaScript 側のプラグインの実行を終了

### 4.3 課題

4.2 節の全体像を踏まえ，提案方式の課題を以下に示す．

**(課題 1)** 利用者の判断による制御
利用者の判断を基に，プラグインによるデバイス資源へのアクセスを防止するため，ダイアログを表示し，その判断結果に基づき，アクセス制御を行う．

**(課題 2)** 利用者に提示する情報の検討
ダイアログを表示する際，利用者がプラグインによるデバイス資源へのアクセスの可否を判断するために必要な情報を検討する．

**(課題 3)** 一度許可されたプラグインを利用者の判断対象から除外
一度許可されたプラグインを再度アクセス制御の対象に含める場合，アクセスのたびにダイアログが表示されるため，利用者にとって不便である．提案方式の利便性を高めるため，制御の結果，一度許可されたプラ

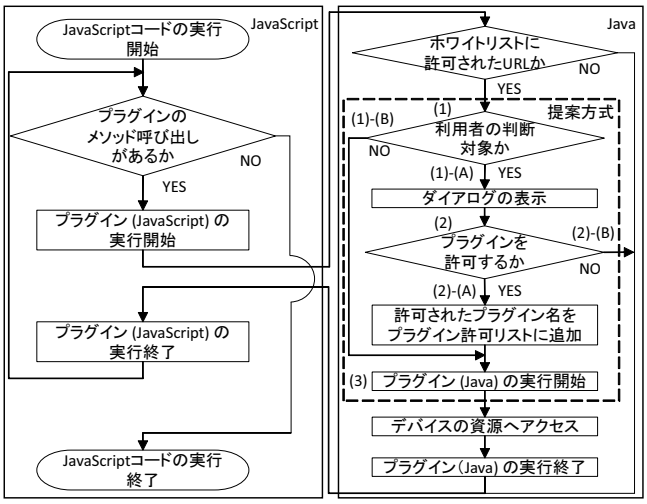


**図 5** 提案方式を適用したプラグインによるデバイス資源へのアクセスの流れ

グインを利用者の判断対象から除外する．

### 4.4 課題への対処

#### 4.4.1 利用者の判断による制御

検出したプラグイン名を基に，ダイアログを表示する．Cordova アプリの利用者は，提示された情報を基に，プラグインによるデバイス資源へのアクセスを許可するか否かを判断する．利用者がデバイス資源へのアクセスを拒否した場合，Java 側のプラグインを実行せず，JavaScript 側のプラグインの実行を終了する．

#### 4.4.2 利用者に提示する情報の検討

利用者がプラグインによるデバイス資源へのアクセスを許可するか否かを判断するために，アクセスを許可するプラグインの動作の把握が必要である．プラグインの動作を把握するために，プラグイン名とそのプラグインによって操作されるデバイスの情報を利用者に提示する．

#### 4.4.3 一度許可されたプラグインを利用者の判断対象から除外

一度許可されたプラグインを利用者の判断対象から除外するために，許可されたプラグインが記述されたファイル（以降，プラグイン許可リスト）を作成する．また，ダイアログを表示する前に，検出したプラグインがプラグイン許可リストに存在するか否かを確認し，存在する場合，利用者の判断対象から除外する．さらに，可否判断の結果，プラグインによるデバイス資源へのアクセスが許可された場合，そのプラグインをプラグイン許可リストに追加する．

### 4.5 提案方式を適用したプラグインによるデバイス資源へのアクセスの流れ

プラグインによるデバイス資源へのアクセスの流れに提案方式を適用した処理の流れを**図 5** に示す．図 5 のうち，提案方式の詳細を以下に示す．

(1) プラグイン許可リストを参照することにより，検出したプラグインを利用者の判断対象に含めるか否か判断する．
  (A) 利用者の判断対象である場合
    検出した内容を基に，ダイアログを表示する．
  (B) 利用者の判断対象でない場合
    (3) に遷移し，Java 側のプラグインの実行を開始する．
(2) ダイアログが表示するプラグイン名とプラグインが操作するデバイスの情報を基に，利用者がプラグインによるデバイス資源へのアクセスの可否を判断する．
  (A) デバイス資源へのアクセスが許可された場合
    許可されたプラグインを利用者の判断対象から除外するため，プラグイン許可リストにプラグインを追加する．
  (B) デバイス資源へのアクセスが拒否された場合
    Java 側のプラグインを実行せず，JavaScript 側のプラグインの実行を終了する．
(3) Java 側のプラグインの実行を開始し，JavaScript API に対応する Java のメソッドを呼び出す．

### 4.6 期待される効果

提案方式の適用により，以下の効果が期待できる．

**(効果 1)** プラグインを悪用した攻撃を検知可能

プラグインを悪用した攻撃を受ける可能性のある脆弱な Cordova アプリを利用者が使用していた場合においても，プラグインを悪用した攻撃を検知し，デバイスの資源へのアクセスを防止することが可能である．

**(効果 2)** Cordova アプリへの適用が容易

提案方式は，Cordova フレームワークにプラグインのアクセス制御処理を追加することによって，プラグインのアクセス制御を実現している．Cordova フレームワークの変更によってアクセス制御が実現できるため，Cordova アプリを修正することなく，提案方式を適用できる．ただし，Cordova アプリの開発者は，利用者の可否判断の結果によってデバイス資源へのアクセスが拒否されることを想定していない．このため，Cordova アプリに必要不可欠なメソッドのアクセスが拒否された場合，Cordova アプリの動作に問題が生じる可能性がある．上記の問題を起こさないために，デバイス資源へのアクセスの拒否を想定した Cordova アプリの修正が必要になる．

## 5. 実現方式と評価

### 5.1 実現方式

#### 5.1.1 実現における課題

4.5 節の設計を踏まえて，提案方式の実現課題を以下に示す．

**(実現課題 1)** デバイスの資源にアクセスするプラグインの検出
プラグインのアクセス制御を行うために，デバイスの資源にアクセスするプラグインを検出する必要がある．

**(実現課題 2)** デバイスの情報を提示するプラグインの検討
ダイアログにデバイスの情報を提示するプラグインについて検討する．

**(実現課題 3)** プラグイン許可リスト
利用者が許可したプラグインが記述されたファイルであるプラグイン許可リストの実現について検討する．

#### 5.1.2 デバイスの資源にアクセスするプラグインの検出

デバイスの資源にアクセスするプラグインを検出するため，プラグイン名を引数に持つメソッドをフックする．Cordova の Java の実装では，Cordova の JavaScript の実装から呼び出された JavaScript API を基に，ホワイトリストによる判定と Java 側のプラグインの実行を行っている．ここで，メソッドのフック箇所として，ホワイトリストによる判定前と Java 側のプラグインの実行前の 2 つの場合が考えられる．それぞれの場合について，以下で検討する．

(1) ホワイトリストによる判定前にフックする場合
Android のバージョンによってブリッジの実装が異なるため，Android のバージョン毎にフックするメソッドを変更する必要がある．具体的には，Android 4.3 以前では，onJsPrompt API，Android 4.4 以降では，addJavaScriptInterface API によってブリッジが実装されており，上記のメソッドをフックする必要がある．

(2) Java 側のプラグインの実行前にフックする場合
プラグインのメソッド呼び出し前に実行される exec メソッドをフックする．なお，exec メソッドは，Android のバージョンによらず実装が同じである．

上記のどちらの場合であっても，デバイスの資源にアクセスするプラグインを検出でき，Android のバージョンによって実装を変更する必要があるか否かのみ異なる．このため，Android のバージョンによらず提案方式を適用できる (2) Java 側のプラグインの実行前にメソッドをフックする方式を採用し，プラグイン名を検出する，

#### 5.1.3 デバイスの情報を提示するプラグインの検討

Cordova Plugin Registry[10] に登録されているプラグインは，2016 年 7 月 27 日時点で，1,470 個存在する．プラグインには，Cordova から提供されている Cordova コアプラグインとサードパーティから提供されているサードパーティプラグインの 2 種類が存在する．このうち，Cordova コアプラグインは，公式のベンダによって提供されているプラグインであるため，サードパーティプラグインと比較して，信頼性が高く，利用される可能性が高いと考えられる．このため，プラグインによって操作されるデバイスの情報の調査対象を 2016 年 7 月 27 日時点において最新版である Cordova 6.3.0 から提供されている 20 個のプラグインに絞る．

**表 1** ホスト OS の評価環境

| | |
|---|---|
| OS | Windows 8.1 Pro 64bit |
| CPU | Intel(R) Core(TM) i5-4460 3.20GHz |
| メモリ | 8GB |
| 仮想化ソフトウェア | VMware Player 7.1.3 |

**表 2** ゲスト OS の評価環境

| | |
|---|---|
| ディストリビューション | Ubuntu 16.04.1 LTS |
| カーネル | Linux 4.4.0-31-generic 64bit |
| 仮想 CPU 数 | 1 |
| メモリ | 4GB |

**表 3** Android Emulator の評価環境

| | |
|---|---|
| Android | Android 5.1.1 |
| Cordova | Cordova 6.3.0 |

#### 5.1.4 プラグイン許可リスト

プラグイン許可リストは，Cordova アプリの内部データ領域に保存する．内部データ領域にプラグイン許可リストを保存することにより，他のアプリケーションによるアクセスを防止でき，文献 [11] で述べられているような WebView を利用した攻撃を受けない．また，内部データ領域に存在するファイルは，Android のルート権限で実行しない限り，ファイルの読み書きは行われない．

### 5.2 評価

#### 5.2.1 評価内容と評価環境

提案方式の有用性と提案方式の適用によって生じるオーバヘッドを明確にするために，以下の評価を行った．

**(評価 1)** プラグインを悪用した攻撃の防止実験
プラグインを悪用した攻撃のうち，我々が指摘したリパッケージを用いた攻撃について，提案方式の適用によって攻撃を事前に防止できるか否かについて検証し，提案方式の有用性を示す．

**(評価 2)** 提案方式に要する処理時間の測定
提案方式を適用しない場合，提案方式の適用によってダイアログが表示される場合，および提案方式の適用によってダイアログが表示されない場合の 3 つにおいて，プラグインによるデバイス資源へのアクセスに要する時間を測定し，提案方式のオーバヘッドを比較評価する．

また，評価環境を**表 1**，**表 2**，および**表 3** に示す．

評価には，自作したテストアプリを使用した．このアプリは，Cordova の Web ページを表示するアプリで，アプリ内で Web ページを安全に表示する InAppBrowser プラグインとデバイスの連絡先データベースにアクセスできる Contacts プラグインが読み込まれている．

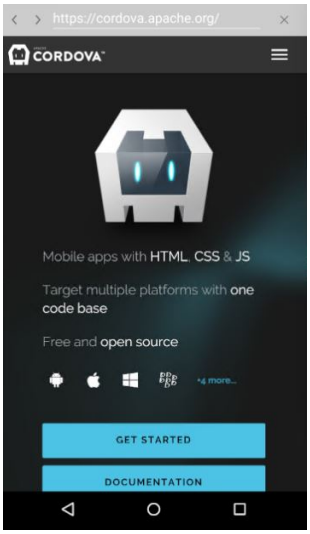


図 6　テストアプリの動作（正常時）

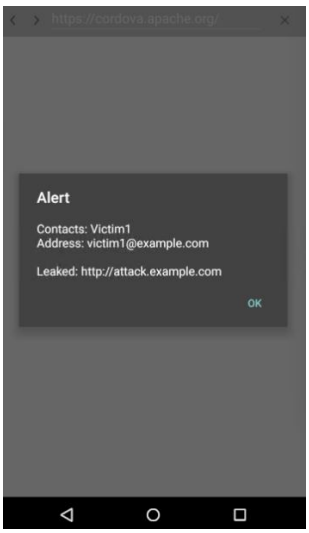


図 7　テストアプリの動作（プラグインを悪用したコードの挿入時）


図 8　提案方式の適用結果（正常時）


図 9　提案方式の適用結果（プラグインを悪用したコードの挿入時）

#### 5.2.2　プラグインを悪用した攻撃の防止実験

リパッケージを用いて，プラグインを悪用するコードを挿入したテストアプリにおいて，プラグインによるデバイス資源へのアクセスを事前に防止できるか否かについて検証する．今回，プラグインを悪用するコードの例として，Contacts プラグインを悪用し，デバイスに保存されている連絡先をアラートで表示する JavaScript コードを埋め込んだ．まず，提案方式適用前における正常時のテストアプリの動作とプラグインを悪用するコードを挿入したテストアプリの動作を**図 6** と**図 7** に示す．図 6 と図 7 を比較すると，図 7 では，Cordova の Web ページが表示される前に，Contacts プラグインを用いて，連絡先が外部に流出することを示すアラートが表示されていることがわかる．

次に，それぞれのアプリに対して，提案方式を適用した結果を**図 8** と**図 9** に示す．図 8 と図 9 を比較すると，図 8 では，InAppBrowser プラグインの可否判断を求めるダイアログが表示されるのに対し，一方，図 9 では，Contacts プラグインの可否判断を求めるダイアログが表示される．プラグインを悪用したコードは，Contacts プラグインを利用しており，ダイアログの内容とプラグイン名が一致することから，プラグインを悪用した攻撃を事前に検知できていることがわかる．また，このダイアログを許可しない場合，Contacts プラグインの使用が許可されず，利用者による制御が可能であることを確認できた．

以上から，提案方式の適用によって，プラグインを悪用した攻撃を事前に検知と制御が可能であることを示した．

#### 5.2.3　処理時間の測定

提案方式の性能を把握するために，テストアプリを 5 回実行し，プラグインによるデバイスの資源へのアクセスに要する平均処理時間を測定した．性能測定は，以下の 3 つの場合に分けて，測定を行った．

**表 4**　プラグインによるデバイス資源へのアクセスに要する処理時間の比較 (単位：ms)

| 方式 | 平均処理時間 |
|---|---|
| (1) 提案方式未適用 | 0.028 |
| (2) 提案方式適用（ダイアログ表示） | 4.115 |
| (3) 提案方式適用（ダイアログ非表示） | 1.150 |

(1)　提案方式を適用しない場合
(2)　提案方式の適用によってダイアログが表示される場合
(3)　提案方式の適用によってダイアログが表示されない場合

なお，(2) においては，ダイアログによる利用者の可否判断に要する時間は考慮せず，ダイアログの表示に要する時間と利用者による判断後のデバイス資源へのアクセスのみを測定する．測定結果を**表 4** に示す．

表 4 の結果から，プラグインによるデバイスの資源へのアクセスに要するオーバヘッドは，ダイアログ表示時約 4ms，ダイアログ非表示時約 1ms と少ないことがわかる．プラグインによるデバイス資源へのアクセスの回数は，Cordova アプリによって異なるが，100 回プラグインの呼び出しが起こると仮定した場合においても，そのオーバヘッドは，ダイアログ表示時約 409ms，ダイアログ非表示時約 112ms と小さい．このため，提案方式の適用によるオーバヘッドは小さいと言える．ただし，上記の評価は，1 つのテストアプリしか評価の対象としていないため，提案方式の性能をより正確に示すためには，多数の Cordova アプリにおいてオーバヘッドを比較測定する必要がある．上記については，今後の課題とする．

## 6.　関連研究

ハイブリッドアプリは，JavaScript からネイティブ言語

のメソッドにアクセスできるという独自の仕組みを持つことから，ハイブリッドアプリを対象にした新たな攻撃が指摘されている．[2][7] また，これらの攻撃への対処やハイブリッドアプリのセキュリティ向上を目的に，ハイブリッドアプリのアクセス制御の研究として，文献 [5]，NoFrak[7]，文献 [8]，および MobileIFC[9] がある．

文献 [5] では，プラグインのアクセス制御が Cordova アプリ単位で行われることを問題に挙げ，Cordova アプリが読み込む URL 毎にプラグインのアクセスを許可するアクセス制御方式を提案している．NoFrak[7] では，`<iframe>` によって読み込まれたアドウェアがデバイスの資源を使用できてしまうフラッキング攻撃に対処するために，URL 毎にネイティブ言語のメソッドへのアクセスを制御している．文献 [8] では，ハイブリッドアプリの JavaScript コードによるデバイス資源へのアクセスを制御するために，ハイブリッドアプリが読み込む URL 毎に開発者がアクセスパーミッションを設定できるアクセス制御方式を提案している．MobileIFC[9] では，PhoneGap アプリが読み込む URL 毎にアクセスパーミッションを開発者側と利用者側双方で設定できるフレームワークを提案している．

文献 [5]，NoFrak[7]，および文献 [8] は，開発者による設定のみを提案しており，利用者によるアクセス制御を考慮していない．このため，悪意のある攻撃者によって，プラグインを悪用する JavaScript コードを Cordova アプリに挿入された場合，利用者側でプラグインの悪用を防止できない．提案方式は，プラグインを悪用する JavaScript コードを Cordova アプリに挿入された場合においても，利用者側でプラグインによるデバイス資源へのアクセスを制御できる．また，MobileIFC[9] は，アクセス制御のために，独自のフレームワークである MobileIFC をインストールする必要があり，開発者による Cordova アプリのコードの修正量が多い．提案方式は，Cordova フレームワークにプラグインのアクセス制御を追加するだけでよく，提案方式の適用による Cordova アプリの修正は不要である．このため，MobileIFC と比較して，適用が容易である．

## 7. おわりに

悪意のある攻撃者がプラグインを悪用し，JavaScript 側からデバイスの資源の奪取や改ざんが可能となる問題について述べた．また，プラグインを悪用した攻撃として，Cordova アプリのリパッケージを用いた攻撃を新たに指摘した．リパッケージを用いた攻撃やクロスサイトスクリプティングを用いた攻撃は，デバイス資源にアクセスするために，プラグインを利用する必要がある．また，これらの攻撃を防止するために，プラグインによるデバイスの資源へのアクセスを動的に制御する方式を提案した．

提案方式は，プラグインによるデバイス資源へのアクセスを事前に検出でき，利用者の判断に基づくアクセス制御が可能である．また，提案方式は，Cordova フレームワーク内に修正を加えるだけでよいため，既存研究と比較して，提案方式の適用が容易である．提案方式の適用により，脆弱な Cordova アプリにおいても，プラグインを悪用した攻撃を検知でき，利用者が Cordova アプリをより安全に利用できる．さらに，評価結果から，提案方式の適用により，プラグインを悪用した攻撃が防止できることとそのオーバヘッドが小さいことを示した．

今後の課題として，利用者に提示する情報の検討やアクセス制御対象とするプラグインの検討がある．また，より詳細な提案方式の有用性や性能の評価がある．

### 参考文献


[1] Apache Software Foundation: Apache Cordova, Apache Software Foundation (online), available from ⟨`https://cordova.apache.org/`⟩ (accessed 2016-07-20).

[2] Xing, J., Xuchao, H., Kailiang, Y., Wenliang, D., Heng, Y. and Nagesh, P. G.: Code Injection Attacks on HTML5-based Mobile Apps: Characterization, Detection and Mitigation, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, pp. 66–77 (2014).

[3] 工藤直樹，山内利宏：Cordova を利用したハイブリッドアプリケーションにおけるプラグインのアクセス制御方式の検討，情報処理学会研究報告，Vol. 2016-CSEC-72, No. 1, pp. 1–8 (2016).

[4] 西村宗晃，熊谷裕志，奥山　謙，戸田洋三，久保正樹：ハイブリッドアプリケーションの脆弱性に関する分析，第 14 回情報科学技術フォーラム (FIT2015) 講演論文集, 第 4 分冊，pp. 13–18 (2015).

[5] Mohamed, S. and Abeer, A.: Reducing Attack Surface on Cordova-based Hybrid Mobile Apps, *Proceedings of the 2nd International Workshop on Mobile Development Lifecycle (MobileDeLi '14)*, pp. 1–8 (2014).

[6] Mutchler, P., Safaei, Y., Doupe, A. and Mitchell, J.: Target Fragmentation in Android Apps, *Proceedings of the IEEE Security Privacy Mobile Security Technologies Workshop (MoST)* (2016).

[7] Georgiev, M., Jana, S. and Shmatikov, V.: Breaking and Fixing Origin-Based Access Control in Hybrid Web/Mobile Application Frameworks, *Proceedings of the 2014 Network and Distributed System Security (NDSS '14)*, pp. 1–15 (2014).

[8] Jin, X., Wang, L., Luo, T. and Du, W.: Fine-Grained Access Control for HTML5-Based Mobile Applications in Android, *Proceedings of the 16th Information Security Conference (ISC '16)*, pp. 309–318 (2013).

[9] Kapil, S.: Practical Context-Aware Permission Control for Hybrid Mobile Applications, *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2013)*, pp. 307–327 (2013).

[10] Apache Software Foundation: Cordova Plugin Registry, Apache Software Foundation (online), available from ⟨`https://cordova.apache.org/plugins/`⟩ (accessed 2016-07-20).

[11] Son, S., Kim, D. and Shmatikov, V.: What Mobile Ads Know About Mobile Users, *Proceedings of the 2016 Network and Distributed System Security Symposium (NDSS '16)*, pp. 1–15 (2016).