Vol.2016-DPS-166 No.1
Vol.2016-CSEC-72 No.1
2016/3/3

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

工藤 直樹[1,a)]　山内 利宏[1]

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，スマートフォン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．Cordova におけるプラグインのアクセス制御はアプリケーション単位で行われているため，制御の粒度が粗い．本稿では，上記の問題に対処するため，プラグインのアクセス制御をアプリケーションが読み込む URL 単位で制御する方式を提案する．提案方式の導入により，URL 毎に許可するプラグインを設定可能になり，Cordova を利用したハイブリッドアプリケーションのセキュリティ向上が期待できる．

1. はじめに

近年，モバイルアプリケーション（以降，アプリ）の開発において，プラットフォームに依存しないアプリである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）による実装が少なく，アプリの大部分を HTML5 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，HTML5 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 側からスマートフォンデバイス（以降，デバイス）固有の資源を使用できる．一般的に，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を使用する場合，プラグインと呼ばれるインタフェースを用いる．プラグインは，Cordova アプリに読み込まれた URL が指す Web サイトやアドウェアに含まれる JavaScript コード（以降，URL 先の JavaScript コード），および Cordova アプリにより使用される．このとき，Cordova アプリが読み込む URL が指す Web サイトの中には，信頼できない Web サイトが含まれる可能性がある．また，Cordova は，プラグインのアクセス制御を Cordova アプリ単位で行っている．このため，信頼できない Web サイトが Cordova アプリによって読み込まれた場合，信頼できない JavaScript コードがプラグインを悪用できる問題がある．

上記の問題を防ぐ手段として，Cordova のホワイトリストの利用がある．しかし，Cordova のホワイトリストは，URL によるアクセス制御のみを行っているため，ホワイトリストの設定が適切でない場合，信頼できない URL が指す Web サイトを読み込み，当該 URL 先の JavaScript コードによるプラグインの使用を防止できない．また，文献 [2] によると，Cordova アプリの開発者はホワイトリストを適切に設定できておらず，Android 版の Cordova アプリのうち，約 30%の Cordova アプリのホワイトリストが適切に設定されていないと述べられている．

本稿では，プラグインのアクセス制御を Cordova アプリ単位ではなく，Cordova アプリが読み込む URL 単位で制御する方式を提案する．提案手法の導入により，URL 毎に許可するプラグインを設定可能になり，Cordova アプリのセキュリティ向上が期待できる．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)] kudo-n@swlab.cs.okayama-u.ac.jp

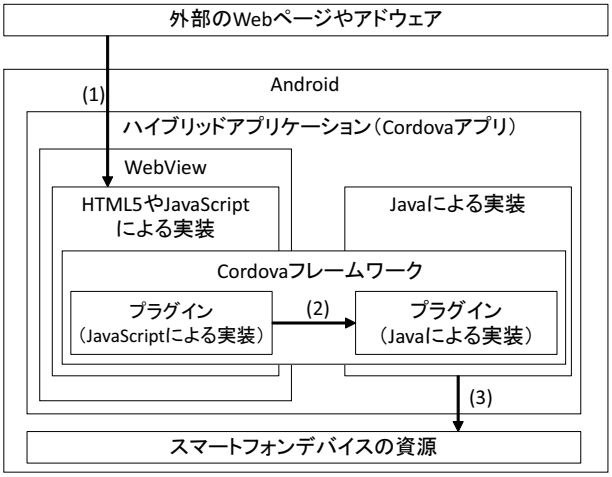

図 1 Android における Cordova を利用したハイブリッドアプリケーションの構成（参考：文献 [3]）

2. Cordova を利用したハイブリッドアプリケーションの問題点

2.1 背景

2.1.1 Cordova を利用したハイブリッドアプリケーションの構成

文献 [3] 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，デバイスの資源を使用できる．また，プラグインは，外部の Web ページやアドウェアによって利用される．外部の Web ページやアドウェアがプラグインを利用してデバイスの資源を使用する流れを以下で説明する．なお，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) 外部の Web ページやアドウェアの読み込み
Cordova アプリにおける HTML5 や JavaScript のソースコードを利用し，外部の Web ページやアドウェアを読み込む．

(2) JavaScript 側のプラグインから Java 側のプラグインへアクセス
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．

(3) Java 側のプラグインからデバイスの資源を使用
Java 側のプラグインからデバイスの資源を使用する．

2.1.2 プラグイン

プラグインは，ハイブリッドアプリが Java の機能を利用する際に用いるインタフェースであり，図 1 のうち，(2) の処理で利用される．プラグインは，JavaScript による実装と Java による実装に分かれる．JavaScript による実装では，Java の機能を利用するための JavaScript API が定義されており，また，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いる．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，デバイスの資源を使用できる．Cordova アプリの開発者は，プラグインを Cordova Plugin Registry[4] から探し出し，Cordova アプリ毎にプラグインをアプリに読み込ませる必要がある．プラグインは，Cordova から提供されているプラグインとサードパーティにより提供されているプラグインの 2 種類がある．

Cordova アプリや Cordova アプリが読み込んだ URL 先の JavaScript コードは，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 アプリでは，カメラプラグイン以外のプラグインを使用できない．

2.1.3 ホワイトリスト

Cordova アプリは，ホワイトリストを用いて Cordova アプリが読み込む URL を制限している．ホワイトリストは，図 1 のうち，(1) の処理で利用されている．ホワイトリストに許可する URL を指定するためには，設定ファイルに許可する URL を記述する．たとえば，http://www.okayama-u.ac.jp を許可したい場合，設定ファイルに以下の記述を追加する．

```
<access origin="http://www.okayama-u.ac.jp"/>
```

なお，ホワイトリストを設定していない Cordova アプリには，全ての URL の読み込みが許可される．

また，ホワイトリストで許可されない Web サイトやアドウェアは，読み込みが行われない．

2.2 Cordova におけるプラグインのメソッド呼び出しの流れ

Android 版の Cordova において，Cordova アプリを実行する際のプラグインのメソッド呼び出しの流れを**図 2** に示し，その詳細を以下で述べる．なお，図 2 における (1) の処理は，図 1 における (1) の処理にあたり，図 2 における (2)–(5) の処理は，図 1 の (2) の処理にあたる．

(1) ホワイトリストに記述された URL かの確認
URL の読み込みを行うメソッドである loadUrl API をフックし，Cordova アプリが URL の読み込みを行う前に，ホワイトリストに記述された URL であるか否か判定する．
  (A) URL の読み込みの継続
  ホワイトリストに許可された URL であった場合，loadUrl API を呼び出し，URL の読み込みが行われる．
  (B) URL の読み込みの中断
  ホワイトリストに許可されない URL であった場合，URL の読み込みを中断する．

(2) JavaScript コードの実行
読み込まれた URL が指す Web サイトに含まれる JavaScript コードを実行する．

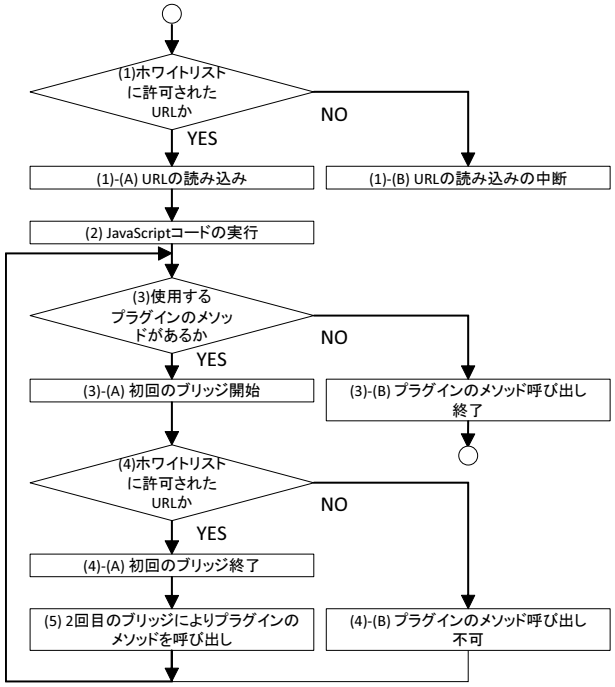

図 2 Cordova におけるプラグインのメソッド呼び出しの流れ

(3) 使用するプラグインのメソッドが存在するかの確認
読み込んだ URL 先の JavaScript コードにプラグインのメソッドを呼び出す exec メソッドが存在するか確認する．
(A) 初回のブリッジを開始
exec メソッドが存在する場合，再度ホワイトリストによる検証を行うため，初回のブリッジを開始する．
(B) プラグインのメソッド呼び出しを終了
exec メソッドが存在しない場合，プラグインのメソッド呼び出しが終了する．

(4) ホワイトリストに許可された URL かの確認
プラグインのメソッド呼び出しを行う URL がホワイトリストで許可されている URL であるか否かを再度検証する．
(A) 初回のブリッジが終了
ホワイトリストで許可されている URL であった場合，初回のブリッジが終了する．
(B) ブリッジが拒否され，プラグインのメソッド呼び出しが不可
ホワイトリストで許可されていない URL であった場合，プラグインのメソッド呼び出しは行われない．

(5) 2 回目のブリッジによるプラグインのメソッドの呼び出し
2 回目のブリッジを行い，プラグインのメソッドを呼び出す．

また，(4) 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 か否かを再度検証する理由は，URL 読み込み時のホワイトリストの判定が回避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の問題点

(問題点 1) プラグインのアクセス制御の粒度が粗いこと
プラグインのアクセス制御は，Cordova アプリの開発者によるプラグインの読み込みを基に，Cordova アプリ単位で行われる．Cordova アプリ単位によるプラグインのアクセス制御方式は，Cordova アプリが URL を読み込んだ場合，その URL 先の JavaScript コードが Cordova アプリの使用するプラグインを全て使用可能であるため，制御の粒度が粗い．

(問題点 2) ホワイトリストが適切に設定されていない Cordova アプリが多数存在すること
信頼できない URL の読み込みを制限するため，Cordova アプリでは，図 2 の (1) のように，URL の読み込みが行われる前に，ホワイトリストを用いた URL に基づくアクセス制御を行っている．このとき，ホワイトリストの設定が適切であれば，信頼できない URL の読み込みは行われず，URL 先の JavaScript コードによるプラグインのメソッド呼び出しは行われない．しかし，文献 [2] の調査によると，Android 版の Cordova アプリのうち，ホワイトリストが適切に設定されておらず，全ての URL の読み込みを許可している Cordova アプリが約 30%存在することが報告されている．また，ホワイトリストは URL に基づくアクセス制御のみを行っているため，ホワイトリストが適切に設定されていない場合，信頼できない URL 先の JavaScript コードによるプラグインの使用を制御できない．このため，Cordova のホワイトリスト制御のみでは，信頼できない URL 先の JavaScript コードによるプラグインを悪用した攻撃を防ぐことが困難である．

2.4 プラグインを悪用した攻撃

2.4.1 攻撃モデル

2.3 節で述べたように, 信頼できない URL 先の JavaScript コードがプラグインを悪用し，デバイスの資源の奪取や改ざんをする恐れがある．また，文献 [2][5][6][7][8] では，信頼できない URL の読み込みによってデバイスの資源の奪取や改ざんをされる可能性があることについて述べられている．プラグインを用いたスクリプトの例を文献 [5] より引用し，**図 3** に示す．

図 3 の例では，2–7 行目で Geolocation プラグインのうち，デバイスの現在地を表示する watchPosition メソッドを悪用し，取得した位置情報を http://128.***.213.66:5556 に送信している．

```
1 <img src=x onerror=
2 navigator.geolocation.watchPosition(
3 function(loc){
4   m = 'Latitude:' + loc.coords.latitude +
5   '\n' + 'Longitude:' + loc.coords.longitude;
6   b = document.createElement('img');
7   b.src = 'http://128.***.213.66:5556?c=' + m })>
```

図 3 プラグインを悪用するスクリプトの例 (引用：文献 [5])

図 3 のような悪意のあるスクリプトが埋め込まれる場合として，以下の 2 つが考えられる．

(1) `<iframe>`や`<img>`タグといった Cordova アプリが読み込む要素
`<iframe>`や`<img>`タグによって読み込まれるアドウェアや画像に悪意のあるスクリプトが含まれている場合，そのスクリプトが実行される．

(2) Cordova アプリが読み込む Web サイト
Cordova アプリ内で読み込まれる Web サイトに，悪意のあるスクリプトが埋め込まれていた場合，そのスクリプトが実行される．

2.4.2 Android パーミッションを用いた対処

利用者によるプラグインのアクセス制御を行う手段として，Android パーミッション（以降，パーミッション）を利用する方法がある．パーミッションは，セキュリティ上のリスクがある機能を制限するために，Android アプリケーションのインストール時に，ユーザにダイアログを表示し，その結果により，機能の許可を判断する仕組みである．パーミッションを利用することにより，Cordova アプリのインストール時に，プラグインが使用するデバイスの資源を許可するか否か設定可能であるため，プラグインによるデバイスの資源へのアクセスを制御できる．しかし，パーミッションは，アプリケーションのインストール後に，利用者によるパーミッションの設定の変更が困難である．また，アプリケーション単位で設定が適用されるため，Cordova アプリが読み込む URL 毎の制御はできない．

3. 提案方式

3.1 目的と考え方

提案方式は，2.3 節で述べた 2 つの問題を対処するために，信頼できない URL 先の JavaScript コードによるプラグインの使用を利用者の判断により防止することを目的とする．提案方式の目的を実現するために，URL 単位でプラグインのアクセス制御を行う．URL 単位でプラグインのアクセス制御を行うことにより，使用を許可するプラグインを URL 毎に設定できる．また，Cordova アプリの利用者が URL 先の JavaScript コードにより使用されるプラグインを制御できるようにするために，図 2 における (4) 2 回目のブリッジによりプラグインのメソッドを呼出し の処理の前に，提案方式の処理を追加する．

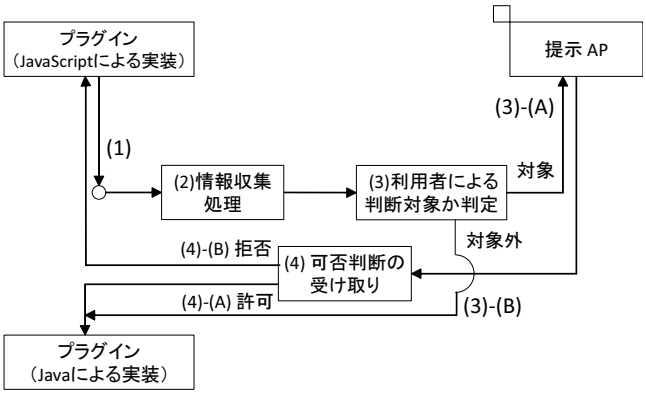

図 4 提案方式の全体像

3.2 全体像

提案方式の全体像を**図 4** に示し，以下で処理の流れを説明する．

(1) プラグインを使用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，提示 AP を呼び出し
  (B) 判断対象でない場合，Java 側のプラグインのメソッド呼び出しを継続
(4) 提示 AP から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) 利用者判断により許可された場合，Java 側のプラグインのメソッド呼び出しを継続
  (B) 利用者の判断により拒否された場合，プラグインのメソッド呼び出しをエラーとして中断

3.3 課題

3.2 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) URL とその URL 先の JavaScript コードが使用するプラグイン名の検出
URL 単位でプラグインのアクセスを制御するために，URL とその URL 先の JavaScript コードが使用するプラグイン名を検出する．

(課題 2) 利用者の判断による制御
利用者の判断によりプラグインの使用を防止するために，(課題 1) での検出結果を基に警告を提示し，判断結果によりプラグインのアクセスを制御する．

(課題 3) 利用者に提示する情報の検討
警告を提示した際，利用者がプラグインの使用の可否を判断するために必要な情報を検討する．

(課題 4) 許可された URL とプラグインの組合せを利用者の判断対象から除外
一度許可された URL とプラグイン名を再度提示する場合，プラグインのメソッド呼び出し時に，何回も同じ警告が表示されるため，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 URL とプラグインの組合せは利用者の判断対象から除外する．

3.4 課題への対処

3.4.1 (課題 1) への対処

URL と URL 先の JavaScript コードが使用するプラグインを検出するため，URL とプラグイン名を引数に持つメソッドをフックする．URL とプラグイン名は，Cordova の Java の実装でのみ使用されるため，Cordova の Java で定義されたメソッドをフックする必要がある．

Cordova の Java の実装では，Cordova の JavaScript から送信されてきた JavaScript API の内容を基に，ブリッジと Java のメソッド呼び出しを行っている．ここで，フックするポイントとしては，ブリッジを行う前やプラグインのメソッド呼び出しを行う前といった 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) ブリッジを行う前にメソッドをフックする場合
Android のバージョンにより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 によりブリッジが実装されている．

(2) プラグインのメソッド呼び出しを行う前にメソッドをフックする場合
プラグインのメソッド呼び出し前に実行される jsExec メソッドをフックする．なお，jsExec 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，目的とする URL と URL 先の JavaScript コードが使用するプラグインを検出することが可能であり，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) プラグインのメソッド呼び出しを行う前にメソッドをフックする方式を採用し，URL とプラグイン名を検出する，

3.4.2 (課題 2) への対処

3.4.1 項で検出した URL とプラグイン名を基に，警告を提示する．Cordova アプリの利用者は，提示された情報を基に，URL と URL 先の JavaScript コードによるプラグインの使用を許可するか判断する．利用者がプラグインの使用を拒否した場合，jsExec メソッドからプラグインのメソッド呼び出しに遷移しないようにし，プラグインのメソッド呼び出しを拒否することにより，信頼できない URL 先の JavaScript コードによるプラグインの使用を防止する．

3.4.3 (課題 3) への対処

利用者がプラグインの使用を許可するか判断するために，使用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，プラグインの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．また，プラグインを URL 毎に可否判断ができるようにするために，プラグインを使用する JavaScript コードが含まれる URL を提示する必要がある．以上より，利用者が URL 毎にプラグインの使用を判断するために，URL，プラグイン名，およびプラグインによって操作されるデバイスの情報を提示する．

また，Cordova Plugin Registry[4] に登録されているプラグインは，2016 年 1 月 25 日時点で，938 個存在する．これら 938 個全てのプラグインに対して，操作されるデバイスの情報を調査し，利用者に提示することは現実的でない．

ここで，プラグインには，Cordova から提供されているプラグインとサードパーティによって提供されているプラグインの 2 種類が存在する．このうち，Cordova から提供されているプラグインは，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，サードパーティ製のプラグインと比較して，使用される可能性が高いと考えられる．このため，提案方式では，プラグインによって操作されるデバイスの情報を調査する対象を全てのプラグインではなく，2016 年 1 月 25 日時点で最新版である Cordova 6.0.0 から提供されている 20 個のプラグインに限定する．

3.4.4 (課題 4) への対処

一度許可された URL とプラグインの組合せを利用者の判断対象から除外するために，許可された URL とプラグインが記述された XML ファイル（以降，プラグイン許可リスト）を作成する．また，警告を提示する前に，検出した URL とプラグインの組合せがプラグイン許可リストに存在するか確認し，存在する場合，利用者の判断対象から除外する．さらにアクセス制御の結果，プラグインの使用が許可された場合，URL と URL 先の JavaScript コードが使用するプラグインをプラグイン許可リストに追加する．

3.5 設計

提案方式の処理の流れを**図 5** に示す．図 5 のうち，提案方式の詳細を以下に示す．

(1) 利用者の判断対象に含めるか否かの判断
jsExec メソッドをフックすることにより，メソッド呼び出しを行う URL とプラグインを検出する．また，検出した組合せを利用者の判断対象に含めるかプラグイン許可リストを解析し，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，警告を提示する．
(B) 利用者の判断対象でない場合
(3) に遷移し，プラグインのメソッドを呼び出す．

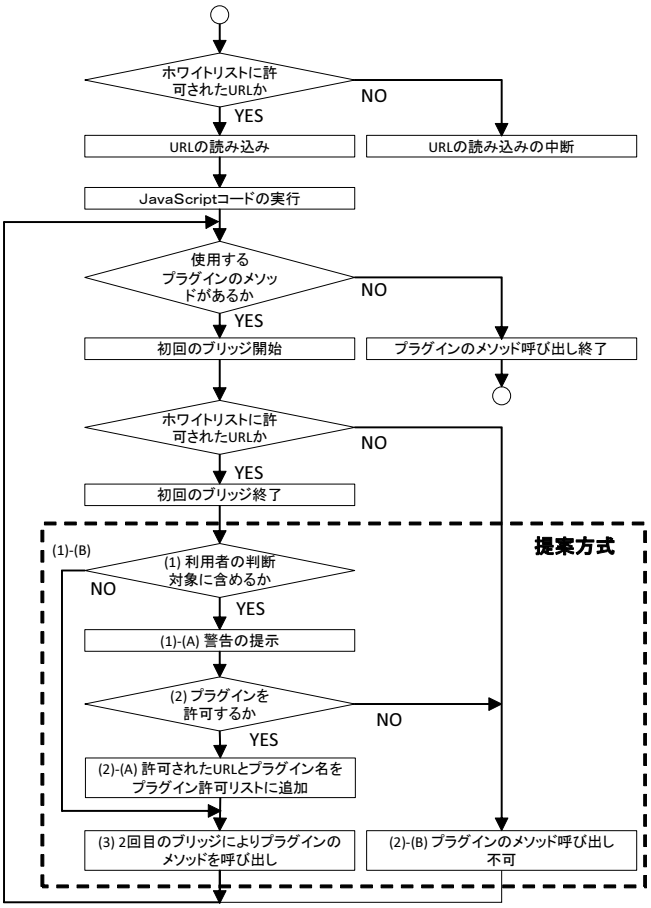

図 5 提案方式の処理の流れ

(2) プラグインの使用の可否を利用者が判断
警告に提示された URL，プラグイン名，およびプラグインが実行する内容を基に，利用者が URL 先の JavaScript コードによるプラグインの使用の可否を判断する．
(A) 許可された URL とプラグイン名をプラグイン許可リストに追加
許可された URL とプラグイン名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 URL とプラグイン名を追加する．
(B) プラグインのメソッド呼び出しが不可
提示 API の内容が拒否された場合，プラグインのメソッドは呼び出されない．

(3) 2 回目のブリッジによるプラグインのメソッドの呼び出し
許可されたプラグインのメソッドを呼び出す．

3.6 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) プラグインのアクセス制御の粒度を細かくできること
Cordova アプリ単位による制御では対処できなかった Cordova アプリが読み込む URL 毎におけるプラグインの使用が制御でき，粒度の細かいアクセス制御が可能である．

(効果 2) ホワイトリストの設定が適切でない Cordova アプリを安全に使用可能
ホワイトリストの設定が適切でない Cordova アプリが信頼できない URL を読み込んでいた場合においても，URL 先の JavaScript コードが使用するプラグインを呼び出し前に制御できる．このため，デバイスの資源の奪取や改ざんを目的とした信頼できない URL 先の JavaScript コードによるプラグインの悪用を防止できるため，ホワイトリストの設定が適切でない Cordova アプリを安全に使用できる．

(効果 3) Cordova アプリの開発者に対するホワイトリストの作成を支援可能
Cordova アプリの開発者が Cordova アプリのデバッグを行う際，アプリが読み込む URL が使用するプラグインを事前に把握できる．また，開発者はデバッグした結果を基に，ホワイトリストに記述する URL を適切に設定できる．さらに，提案方式で利用するプラグイン許可リストを編集することにより，URL 毎に許可するプラグインを設定できる．

(効果 4) Cordova アプリの利用者の判断を支援可能
Cordova アプリが URL の読み込み時に，プラグインを検出し警告を提示することにより，利用者が悪意のある URL を読み込む可能性があることを事前に知ることが可能である．また，警告の内容を基に利用者がプラグインの使用を許可するか判断できない場合，Web サイトの URL を Cordova アプリではなく，デフォルトブラウザでロードさせ，Web サイトの安全性を確認した上で使用の可否を判断することも可能である．

4. 関連研究

4.1 addJavaScriptInterface API を利用した攻撃に関する研究

文献 [9] では，Android の WebView において，addJavaScriptInterface API を利用することにより任意の JavaScript コードが実行できる問題について述べられている．この問題の対処として，文献 [10] や文献 [11] では，addJavaScriptInterface API が JavaScript 側と Java 側で利用できる共通のオブジェクトを生成することに着目し，addJavaScriptInterface API をフックし，共通オブジェクトから解析した情報を基に，悪意のある API を検知する手法について述べられている．

Cordova アプリにおいても，ブリッジを利用する際に，addJavaScriptInterface API が利用されているが，Android 4.4 以降に限定されている．このため，Android 4.3 以前の場合，文献 [10] や文献 [11] の手法は適用できない．さら

に，文献 [10] や文献 [11] は，Cordova アプリを対象にしていないことや Android 4.4 以降で addJavaScriptInterface API の仕様が変更されていることから，Android 4.4 以降においても，文献 [10] や文献 [11] の手法が適用できない可能性がある．

提案方式は，Android のバージョンによらず，Cordova アプリに対応でき，URL 毎に使用するプラグインを確実に検出できる．

4.2 ハイブリッドアプリケーションに関する研究

文献 [2] では，ハイブリッドアプリケーションにおいて`<iframe>`により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ついて述べられている．また，Cordova のホワイトリストにより URL の読み込みを拒否した場合，アドウェアの読み込みが行われないという問題について述べられている．さらに，上記の問題に対処するため，ホワイトリストが許可しない URL であっても，Java のメソッドにアクセスするブリッジのみを拒否し，URL の読み込みを許可する NoFRAK を提案している．

文献 [5] では，従来の Web アプリケーションと比較して，ハイブリッドアプリケーションでは，クロスサイトスクリプティング（以降，XSS）の攻撃可能なチャネルが広がることやハイブリッドアプリケーションが JavaScript API によってデバイス資源を使用できるために 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を指摘している．また，XSS 攻撃の一種である DOM Based XSS 攻撃に焦点を充て，XSS 攻撃を受ける可能性のある API を調査し，Cordova アプリを静的解析することにより DOM Based XSS 攻撃を検出する手法を提案している．さらに，XSS 攻撃を緩和するために，`<iframe>`や`<img>`といった HTML タグ内で指定された JavaScript コードを除外する手法を Cordova に実装している．

文献 [2] と文献 [5] では，Cordova アプリが読み込むプラグインの制御は行っていないため，信頼できない URL 先の JavaScript コードを読み込んだ場合，プラグインを悪用できてしまう．

また，Cordova におけるプラグインやプラグインが使用するデバイスの資源のアクセス制御の研究として，文献 [6]，文献 [7]，および文献 [8] がある．

文献 [6] では，URL 毎にプラグインが使用するデバイスの資源へのアクセスを制御する方式を提案している．また，文献 [7] では，URL 先の JavaScript コードが使用するプラグインを自動で検出し，検出した全てのプラグインを URL 毎にアクセス制御する方式を提案している．しかし，文献 [6] と文献 [7] の方式は，開発者による設定が適切であることを前提としており，利用者によるアクセス制御を考慮していない．このため，ホワイトリストの設定が適切でない Cordova アプリを利用する場合，文献 [6] や文献 [7] の方式では，信頼できない URL 先の JavaScript コードによるプラグインの使用を防止できない．

提案方式は，ホワイトリストの設定が適切でない Cordova アプリにおいても，利用者によるプラグインのアクセス制御が可能であり，利用者が安全に Cordova アプリを使用できる．

また，文献 [8] では，Cordova を拡張し，デバイスの情報へのアクセス制御を行うフレームワークを提案している．フレームワークは，開発者と利用者それぞれが利用でき，利用者によるデバイス情報へのアクセス制御が可能である．しかし，文献 [8] の手法は，Cordova アプリが使用するデバイス情報毎にアクセス制御を行っており，Cordova アプリが読み込む URL 毎にデバイス情報へのアクセスを制御できないため，制御の粒度が粗く，信頼できない URL 先の JavaScript コードによるデバイス情報へのアクセスを検出できない．

提案方式では，Cordova アプリが読み込む URL 毎にプラグインのアクセス制御を行うため，信頼できない URL 先の JavaScript コードによるプラグインのアクセスを検出できる．

5. おわりに

Cordova アプリの問題点と Cordova アプリを対象にした攻撃について述べた．Cordova アプリは，URL に基づくアクセス制御のみを行っており，Cordova アプリ単位でプラグインのアクセスを制御している．利用者によるプラグインを使用したデバイスの資源のアクセスを防止する手段として，パーミッションを利用した方式がある．しかし，パーミッションを利用した方式は Cordova アプリ単位でプラグインが使用するデバイスの資源をアクセス制御しているため，制御の粒度が粗く，URL 単位でプラグインの使用を制御できない．提案方式では，URL 単位でプラグインのアクセスを制御する方式の設計を示した．提案方式は，Cordova アプリが読み込む URL 先の JavaScript コードが使用するプラグインを検出できる．また，利用者の判断に基づき，アクセス制御を行うため，利用者側でプラグインの使用を制御できる．

今後の課題として，提案方式の実装と提案方式の有用性や性能の評価がある．また，iOS といった Android 以外のプラットフォームにおける提案方式の適用可能性の検討がある．

参考文献

[1] Apache Software Foundation: Apache Cordova, Apache Software Foundation (online), available from ⟨`https://cordova.apache.org/`⟩ (accessed 2016-01-25).
[2] Georgiev, M., Jana, S. and Shmatikov, V.: Breaking and Fixing Origin-Based Access Control in Hy-

brid Web/Mobile Application Frameworks, *Proceedings of the 2014 Network and Distributed System Security (NDSS '14)*, pp. 1–15 (2014).

[3] 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).

[4] Apache Software Foundation: Cordova Plugin Registry, Apache Software Foundation (online), available from ⟨`https://cordova.apache.org/plugins/`⟩ (accessed 2016-01-25).

[5] 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).

[6] 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*, pp. 309–318 (2013).

[7] 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).

[8] 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).

[9] Tongbo, L., Hao, H., Wenliang, D., Yifei, W. and Heng, Y.: Attacks on WebView in the Android System, *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC '11)*, pp. 343–352 (2011).

[10] Yu, J. and Yamauchi, T.: Access Control to Prevent Malicious JavaScript Code Exploiting Vulnerabilities of WebView in Android OS, *IEICE Transactions on Information and Systems*, Vol. E98-D, No. 4, pp. 807–811 (2015).

[11] 川端秀明，磯原隆将，竹森敬祐，窪田　歩：Android パーミッションを悪用する Script の脅威と静的解析，情報処理学会研究報告，Vol. 2011-CSEC-53, No. 3, pp. 1–6 (2011).