静的解析を利用したセキュアなハイブリッドアプリの開発支援に関する研究

坂 周英[1] 清 雄一[1] 田原 康之[1] 大須賀 昭彦[1]

概要：近年モバイルアプリの開発において, ハイブリッドアプリを使用する開発者が増えてきている．ハイブリッドアプリとは通常のネイティブアプリと Web アプリを組み合わせ, 二つの利点を兼ね備えたモバイルアプリである．しかしこれはネイティブと Web 用の二つの言語で書かれたプログラムであるため,JavaScript 側からネイティブの資源にアクセスできる利点を利用した脆弱性が存在することが多く, セキュリティ面で安全でない．そのためアプリ配布後の悪意のある攻撃者からの攻撃を防ぐため, 事前に開発者が対策することが望ましい．しかしハイブリッドアプリのセキュリティについての技術はあまり進んでいない．そこで本研究ではハイブリッドアプリの静的解析を行い, アプリ開発者が情報流出の可能性が低いセキュアなアプリの作成の支援をするシステムを提案する．提案した手法により多種のハイブリッドアプリの情報流出の可能性が検出可能なことがわかった．また現在も発展し続けているハイブリッドアプリに対しても動向を追いつつ対応していく必要があることも明らかになった．

Support for development of secure hybrid applications using static analysis

SHUEI BAN[1] YUICHI SEI[1] YASUYUKI TAHARA[1] AKIHIO OHSUGA[1]

1. はじめに

近年, スマートフォンの普及が急速に進んでいる．それによりスマートフォンアプリが使用される場面も増えてきている．そのため企業にとっては大きな市場であり, 企業のサービスをより身近に便利に使用してもらうためにマーケティング用のアプリを開発するケースが増えてきている．しかし現在, ユーザーが使用しているスマートフォンのプラットフォームのシェアは Android や iOS などいくつかに分かれている．そのためスマートフォン市場全体にアプリを配布しようとすると, プラットフォームごとに別々に開発しなければならないため, その分コストが増加してしまう．

そこで今日, このような問題を解消できるハイブリッドアプリの需要が増加してきている．ハイブリッドアプリとはネイティブアプリと Web アプリを組み合わせたアプリである．ネイティブアプリとは最初に述べたようなアプリで, コストはかかるがデバイスの様々な機能を使用できる．一方 Web アプリはスマートフォンのウェブブラウザ上で動作するアプリなので, 単一の言語ですべてのプラットフォームに対応できる．またコンテンツの更新などはアプリのアップデートなどをする必要がないので, 最新の状態を保ちやすい．しかしウェブブラウザ上での動作のためプラットフォーム固有のデバイスの機能がほとんど利用できない．ハイブリッドアプリはネイティブ上でウェブブラウザの使用が可能となる機能を持つ UI 上にアプリの主要コンテンツを表示し, プラグインを通してデバイスの機能を利用できるようにしたものである．そのためアプリのメインとなる部分は Web アプリとほとんど同様に実装できる．またハイブリッドアプリ開発用のフレームワークを使用すれば, デバイスの機能を利用するためのプラグインはほとんど自分でコードを書く必要がない．このようにネイティブアプリと Web アプリの利点を組み合わせたハイブリッドアプリは低コストで多くのプラットフォームに対応できる．しかしこれはネイティブと Web 用の二つの言語で書

[1] 電気通信大学大学院情報理工学研究科情報学専攻
Department of Informatics, Graduate School of Informatics and Engineering, The University of Electro-Communications 182-8585, Tokyo, Japan

かれたプログラムであるため,JavaScript 側からネイティブの資源にアクセスできる利点を利用した脆弱性が存在することが多く, そこからスマートフォン内にある情報が流出してしまう．そのためアプリ配布後の悪意のある攻撃者からの攻撃を防ぐため, 事前に対策することが望ましい．

本研究ではハイブリッドアプリの静的解析を行い, アプリ開発者が情報流出の可能性が低いセキュアなアプリの作成の支援をするシステムを提案する．静的解析ではハイブリッドアプリ独特の構造を解析するためにいくつかの手法の組み合わせや開発者からの情報を利用する事で多種あるハイブリッドアプリの構造の解析を可能にする．また情報流出となる可能性のある経路をテイント解析により検出する．なお本研究では Android 上でのハイブリッドアプリを対象にしている．

2. ハイブリッドアプリ

本章ではハイブリッドアプリの本研究が対象とする部分について紹介する．

ハイブリッドアプリの概要は図 1 のようになっている．図のようにプラグインを介すことでデバイスの機能へのアクセスを可能にしているため, コード注入攻撃などにより,JavaScript コードなどからプラグインを利用しスマートフォン上の情報にアクセスされてしまう危険性がある．

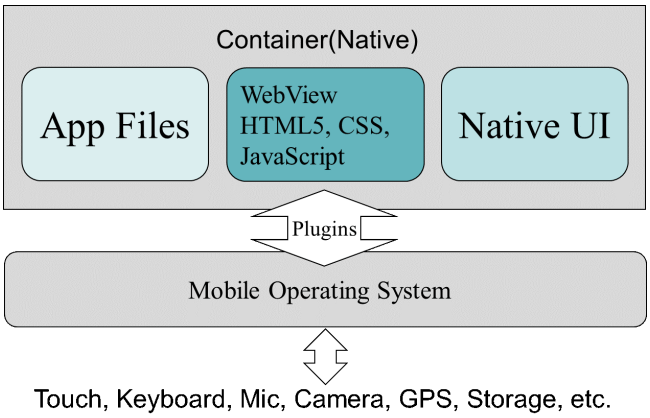

図 1 ハイブリッドアプリの構造

2.1 Java と JavaScript の呼び出し

Android のハイブリッドアプリには, 初めに説明したような WebView が JavaScript からネイティブ（Java）の機能を呼び出す方法が主に二つある．

一つ目は Java 上の WebView で二言語が直接通信するためのブリッジとなるオブジェクトを作成し, それを利用する方法である．JavaScript 側からはこのオブジェクト作成時に指定したグローバル変数を通し Java にアクセスする．

二つ目はコールバック関数による呼び出しで, 主に Cordova などのハイブリッドアプリ開発フレームワークでネイティブの機能を呼び出すプラグインに使用されている．

```
1 WebView wv = WebView(this);
2         wv.addJavascriptInterface(newA(), "bridge
              ");
3     wv.loadUrl("...");
4     ...
```

図 2 ブリッジ呼び出し Java 側

```
1 var contact = bridge.getContact();
2 ...
```

図 3 ブリッジ呼び出し JavaScript 側

```
1 class Contact extends CordovaPlugin{
2         boolean execute(string action,
              CordovaArgs args,
3     CallbackContext callbackContext)
4         ...
5     }
```

図 4 コールバック呼び出し Java 側

```
1 function shownumber(){
2         var Callback = function(contact) {
3     alert("number=" + contacts.phone);
4     }
5 }
```

図 5 コールバック呼び出し Java 側

2.2 フレームワーク, プラグイン

ハイブリッドアプリの機能をアプリ開発者が一から実現するのは非常に手間で困難である．そこでだれでも手軽にハイブリッドアプリを作成できるようにいくつか開発用フレームワークが存在する．またこれらのフレームワークは主にネイティブの機能を使用する際にプラグインを使用する．例えばハイブリッドアプリ上で連絡先の情報が使用したい場合, 専用のプラグインを追加するだけで使用可能となる．またプラグインは用途に合わせて自作可能であり多くのプラグインが作成, 配布されている．

2.3 Cordova

ここでは一例として, 現在使用率が高いハイブリッドアプリ用フレームワークである Cordova[10] 及びそのプラグインについて説明する．

先に説明したように Cordova はプラグインを使用することで, 簡単に開発者がハイブリッドアプリ上でネイティブの

機能を扱える．図 5 のように HTML では cordova.js,index.js を読み込んでいる．Cordova はこの cordova.js 上で多くのハイブリッドアプリの機能を実現している．

```
1 <script type="text/javascript" src="cordova.js
      "></scripts>
2 <script type="text/javascript" src="js/index.js
      "></scripts>
```

図 6　index.html

```
1  var button = document.getElementById("button");
2  button.addeventListener("click", function
       findContacts() {
3          ...
4
5          fields = ["displayName", "emails"];
6      navigator.contacts.find (fields,
           contactfindSuccess, contactfindError,
           options);
7
8      function contactfindSuccess (contacts) {
9          for (var i = 0; i < contacts.length; i++)
               {
10                alert("Name = " + contacts[i].
                      displayName + "\n emails =
                      contacts[ i].emails[0]["value
                      "]);
11             }
12         }
13     function contactfindError(message) {
14         allert('Failed because : ' + message);
15     }
16     ...
17 }
```

図 7　index.js

図 7 の例では連絡先情報を扱えるプラグインを使用している．contacts.displayName,contacts.emails で名前およびアドレス情報が簡単に利用できているのが分かる．実際に動かすと図 8 のようになる．

3. ハイブリッドアプリの問題点

しかしハイブリッドアプリはこのようにネイティブ＋Web を実現したため,Web アプリと同様の攻撃を受けやすいといったセキュリティ面での問題がある．通常のスマートフォンでの Web アプリであれば, ウェブブラウザがサンドボックス化しているため, 攻撃によるデバイス上の被害は少ない．しかしハイブリッドアプリは第二章で説明したようにプラグインを介してデバイスの機能にアクセスするこ

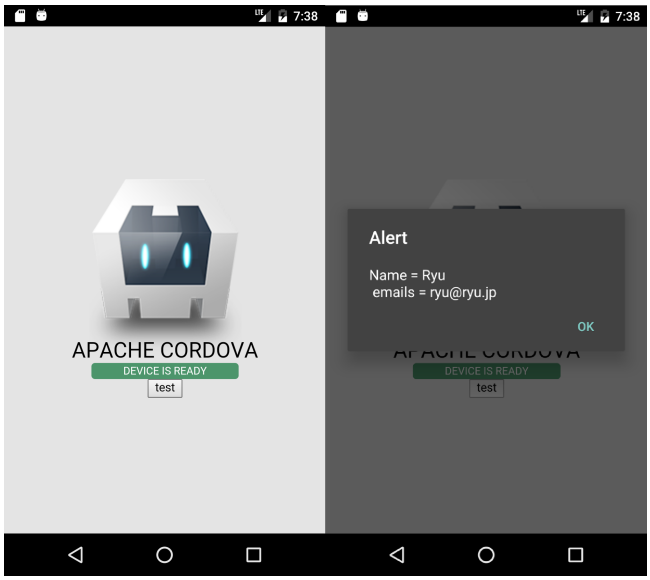

図 8　プラグイン実行画面

とができるため, そこからデバイスの情報流出といった被害拡大が考えられる．例えば図 8 のように連絡先を取得できる API をもつハイブリッドアプリがアラートではなく別の外部へ情報を渡せるモジュールに連絡先を渡せた場合、連絡先が流出してしまう可能性がある．近年需要が増えているハイブリッドアプリは, 現状ネイティブアプリに比べてセキュリティ対策が進んでおらず, 既存のネイティブアプリの解析ツールなどではアプリの構造が違うため危険となりえる部分を検出することができない．そのためハイブリッドアプリに適したセキュリティ対策が必要であり、今回はその手法を提案する．

4. 提案方式

システム構成図は以下の図のようになっている．

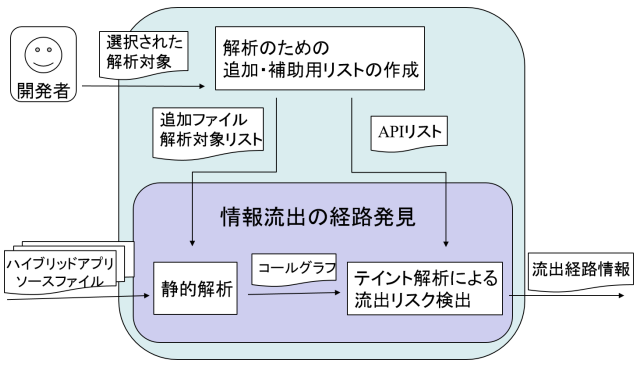

図 9　提案手法

情報流出の発見方法としては, まずハイブリッドアプリのソースファイルを読み込み静的解析を行いコールグラフを作成する．そしてそのコールグラフからテイント解析を行うことによって, 情報流出の経路を発見する．

これらを行う前に開発者には, 開発したアプリについて

の情報をいくつか選択してもらう．そして解析のそれぞれの操作の補助になるようなリストを作成し, 提案手法の精度向上を行う．以下では具体的な解析法や使用したツールなどの解説を行う．

4.1 静的解析によるコールグラフの作成

コールグラフとは図 10 のような呼び出し関係を表した図である．コールグラフ作成に至って,Java と JavaScript の言語間を介した呼び出し関係も再現する必要がある．ここでは「HybriDroid」[4] を使用した．HybriDroid は Lee ら [2] が開発しているものでハイブリッドアプリを対象とした静的解析用フレームワークである．このツールの最大の特徴は Java と JavaScript の二言語を呼び出し関係など含め同時に解析可能な部分である．

しかしながら HybriDroid は第四章で説明した一つ目の呼び出し方法のみを対象としており,Cordova などのフレームワークによって実装されたものには対応できない．そこで Brucker ら [1] の研究を参考に主に Cordova によって作成されたハイブリッドアプリにも対応していく．Brucker らは Cordova による Java と JavaScript 間の呼び出し関係を明らかにし, それらを静的解析する手法を提案した．通常 Cordova などのフレームワークをそのまま解析してしまうと, フレームワークがハイブリッドアプリの機能を実現するために使用する専用の関数などにより, 正確な結果を得ることができなかったり, 必要のない部分まで解析してしまい解析量が膨大になるようなエラーが起こることが多い．そこで Brucker らは両言語での呼び出しの部分の一部を書き換え, 静的解析をしたときに必要なコールグラフが正確に取得できるようにしたものである．

これらの手法を組み合わせることにより多くの種類のハイブリッドアプリへの対応を実現する．

4.2 テイント解析による流出リスク検出

テイント解析とはあるデータに対してタグ付けを行い, そのデータが使用や移動されるたびにその場所にも同様のタグを伝搬させていくことで, 初めにタグ付けしたデータが行く可能性のある場所や経路を抽出するものである．

本システムではテイント解析後, 危険と考えられるソース（流出元）とシンク（流出先）のペアの発見を目的とする．

例として図 10,11 で説明する．危険なソースとして A, シンクとして外部に発信できる機能を持つとして H が指定してあるとする．まず a テイントタグをつけその利用先にタグを伝搬．この時,a のテイント情報上に A と H が存在したので, 結果として流出情報の a, 流出経路の H ← E ← C ← A を発見する．

今回の提案手法ではテイントタグの付与及び伝搬は先ほど使用した HybriDroid がその機能を持つためそのまま利用する．

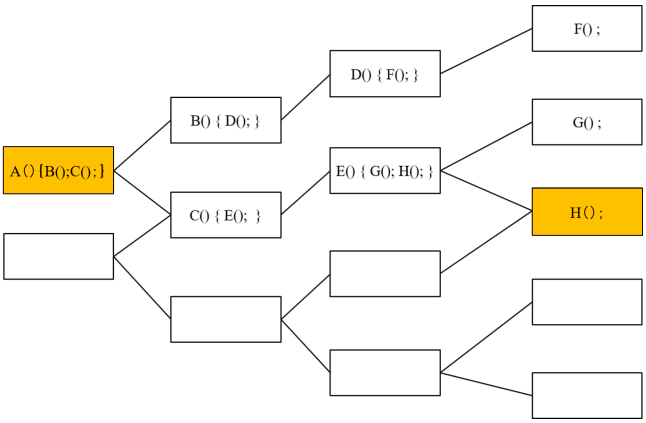

図 10 コールグラフ

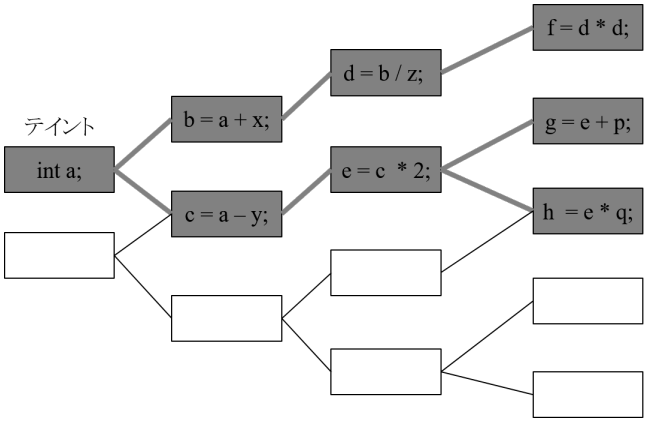

図 11 テイント解析

情報流出する可能性がある場所を特定するには危険なソースとシンクのペアを発見する必要がある．ここでいう危険なソースとはバーコードリーダーや Bluetooth といった外部からの侵入経路や, 連絡先やデバイス情報など内部のデータが参照できるような場所のことを指す．また危険なシンクとは HTML 上でコードが実行可能場所や, ウェブ上へデータを送るようなデータの流出先のことを指す．

また Jin ら [5] の研究を参考にして作成したハイブリッドアプリ上で危険なソースとシンクの API のリストを使用し, テイント上でのペアを検出する．表 1 は危険なソースの API 例で Bluetooth や NFC リーダーなどスマートフォンならではの攻撃経路も想定し流出経路情報を検出する事ができる．

表 1 危険な API リスト例

攻撃経路	API
外部	Bluetooth.getUuids
	NFC.addNdefListener
	barcodeScanner.scan
内部	contacts.find
	DirectoryEntry.getDirectory/getFile
	Media.getFormatData

4.3 アプリ開発者による解析対象の指定

第三章で述べたようにハイブリッドアプリは様々な方法によって作成されている．本研究では可能な限り多くの状況に対応, またより精度の高い解析が可能になるように, アプリ開発者が持つ情報を使用する．具体的な選択内容としては, フレームワーク, プラグイン, 動的に呼び出す JavaScript ファイルなどである．そして選択後にコールグラフの作成及びテイント解析用の解析対象リストとしてそれぞれ作成する．それぞれの作業対象にこれらを指定しておくことでより正確な検出を可能とする．

4.3.1 プラグインの選択及び処理例

テイント解析ではソースとシンクのリストとして Jin らの研究を参考した危険な API リストを使用しているが, そこに開発者が使用したプラグインが使用している API を追加する．そうすることで新しいプラグインや, 自作によるプラグインなどに対応でき, 可能性のあるプラグインすべてを検索することが可能となる．

図 12 はフレームワークの一つである Ionic のデバイス情報を取得するプラグインである．そして図 13,14 はそのプラグインが選択された場合の前処理の例である． Ionic は Cordova の機能を利用しているが, そのまま解析にかけても Ionic 特有のコードなどにより正確に解析できない. そのためこれを選択された場合 Ionic プラグインを Cordova プラグインに変換する．具体的には選択されたらプラグインの JavaScript ファイルを置き換えと呼び出しの関数の書き換えを行う．これにより既存の Cordova アプリ解析に適応可能なる．

```
1  // instal cordova plugin add cordova-plugin-
       device
2  // link https://github.com/apache/cordova-plugin-
       device
3
4  /* globals device : true */
5  angular.module('ngcordova.plugins. device', [])
6
7   .factory ('$cordovaDevice', [function () {
8          return {
9                  ...
10           };
11 }]);
```

図 12 Ionic プラグイン

4.3.2 コールグラフ作成での選択及び処理例

一つ目はコールグラフ作成の前にあらかじめフレームワークを指定しておくことで個々のフレームワーク特有のファイルや関数などの正確な特定, または余計な部分を除外することが可能となる．

二つ目はハイブリッドアプリのソースコード外にあるファイルの選択である．ハイブリットアプリの性質上,HTML や JavaScript コードなどを外部から呼び出すことができる．それらのファイルをあらかじめ解析対象に加えることで正確に解析可能な部分を増やすことが可能となる．

```
1  document.addEventListener("deviceready", function
       () {
2          var version = $cordovaDevice.getVersion()
               ;
3  }, false);
```

図 13 プラグイン前処理

```
1  document.addEventListener("deviceready", function
       () {
2          var version = device.version;
3  }, false);
```

図 14 プラグイン前処理

図 15 はテストとして選択前後で一部不要と思われる部分を除外したコールグラフの例である．

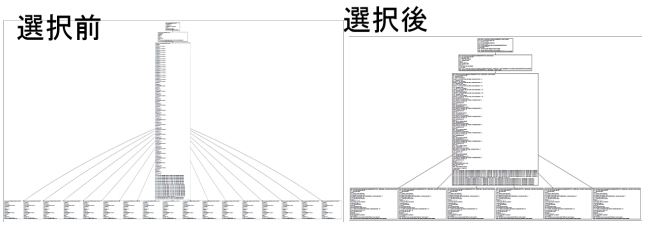

図 15 コールグラフ前処理

5. 評価

本研究が作成するシステムは既存のツールや研究を組み合わせたものとなるため, 主な解析精度や動作速度についてはそれらに依存するものが多い．ここではこのシステムが既存に加える部分について評価を行う．

解析対象の指定について, 既存のツールは目的が一般に出回っている悪質または危険なハイブリッドアプリの発見である．そのため APK ファイルを入力とし, そこから解析対象の探索をして文字解析後にローカルには存在しないファイルや動的に読み込まれる JavaScript コードを取得している．本システムでは解析前にアプリ開発者に対象をしてもらうフェーズを設けているため, 解析漏れを防ぎ精度が上昇かつ動作時間の短縮になる．

同様にテイント解析でも開発者が使用したプラグインを指定できることで, 対象のハイブリッドアプリのソースとシンクが明確になり, 情報のさらなる流出経路の発見につながる．またハイブリッドアプリの開発支援として, 使用したフレームワークやプラグインに実行可否ができるだけ

ないことが望ましい．本システムでは複数の研究やツールの組み合わせにより, 複数のハイブリッドアプリのパターンに対応できる．

本システムでのハイブリッドアプリ対応範囲は以下の表 2 のようになった．

表 2　対応範囲表

	提案手法	Hybri Droid	Brucker らの手法	Jin らの手法
Java	○	○	○	×
JavaScript	○	○	○	○
ブリッジ呼び出し	○	○	×	×
コールバック呼び出し (Cordova)	○	×	○	○
Ionic	△ [1]	×	△	×
その他フレームワーク	△ [2]	× [3]	△ [4]	×

[1] 複雑なアプリに対してはエラーが出ることがある．

[2] 新しいフレームワークやプラグインに関してはそれぞれに対しての対処を追加していくことでより多くの状況への対応を目指す．現状は未知のものに対しては除外している．

[3] プラグインを使用しない場合など一部環境については解析可能な場合がある．

[4] 一部のフレームワークを除外するフィルターを実装している．

表 2 の 4 手法に対し比較・評価を行うと, まず二言語における静的解析において,HybriDroid ではブリッジによる呼び出し,Brucker らの手法では Cordova におけるコールバックによる呼び出しに対応している．

本システムではこれらを組み合わせることで, 両方における対応を可能とした．これについては, 現在は状況に応じて手法を使い分けている．

またテイント解析においては Jin らの手法のソース, シンクを利用することで, ハイブリッドアプリにおける危険な API に対応可能としている．

更に, 開発者の情報を加えることにより, その他のフレームワークやプラグインについても対応範囲が広がっている．

6. 考察・課題

提案方式についての考察及び, 今後のハイブリッドアプリに対しての考察を行う．

6.1 提案方式

テストにより様々なハイブリッドアプリに対応することは可能な事が分かった．しかしまだそれぞれ個々の手法で別々のテストしかできていないため, すべてがテスト通りに動作する保証はない．

開発者による解析対象の選択については, 解析において余分な部分の除去ができ有用なことが分かった．また解析の対応範囲拡大においても有効なことが分かった．

今後選択対象やそれらに対する処理を追加していくことで, より一層の解析の対応範囲の拡大および解析の精度の向上が可能になると考える．またその際選択においてはフレームワークなどの一部特徴から自動で判別可能な部分もある．そのような部分については開発者の手間にならないようにできるだけ自動化したほうが良いと考える．

6.2 今後のハイブリッドアプリ

現在もハイブリッドアプリは発展段階にあり, 様々な特徴を持ったフレームワークが開発されている．例えば Ionic から,JavaScript フレームワークである Angular とより密に結合した Ionic2 などがある．中には Cordova や Ionic のように描画を WebView で行うものだけでなく, ネイティブで行われているものも開発され増え始めている．例としては React Native や Native Script といったフレームワークがある．これらは図 16 のように JavaScript エンジンによって JavaScript コードを実行しネイティブで描画を行うことによりネイティブに近い UI が高パフォーマンスで実現可能となる．

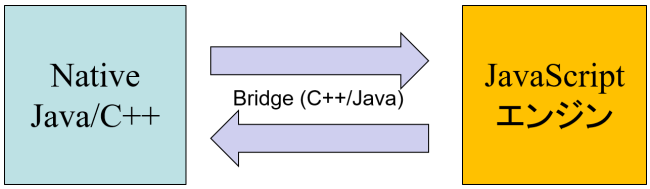

図 16　JavaScript エンジンによるネイティブの呼び出し

現在最も注目を浴びているフレームワークは先に例として挙げた Ionic2,React Native,Native Script である．中でも React Native[11] は Facebook が開発した JavaScript フレームワーク React を利用したハイブリッドアプリであり,2015 年 3 月に IOS 版,9 月に Android 版がリリースされたばかりだが,Facebook や Instagram を始めとして多くのアプリに採用されている．今後も様々な形でハイブリッドアプリは発展を続けると考えられる, そのためセキュリティ対策としてもそれに対応していく必要があると考えられる．またまだ新しく, 発展段階にあることを考えるとバグ検出としてもアプリの解析は役に立つと考えられる．

6.3 React Native

ここでは先ほど例に挙げた React Native について, これまでのハイブリッドアプリとの違いとして主に Java と JavaScript のブリッジ部分の詳細及び, 解析対象としたときに考えられる手法などを考察する．

6.3.1 描画

UI 関係の呼び出しについて説明する．まず初めに図 17 のようにネイティブ側から定義された関数や引数が

JavaScript 側に受け渡される．この逆方向も含め受け渡しには C++による JSON ベースのプロトコルが使用されている．JavaScript エンジンについては JavaScriptCore が使用されている．JavaScript 側に送られたものは, まとめて処理され図 18 のようにネイティブ側に戻される．

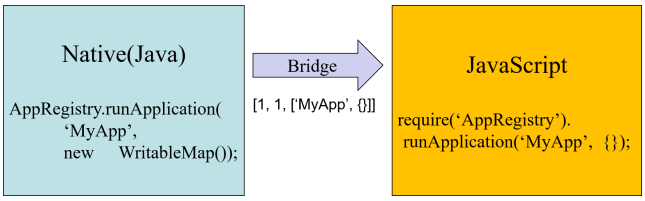

図 17　Java 側からの呼び出し

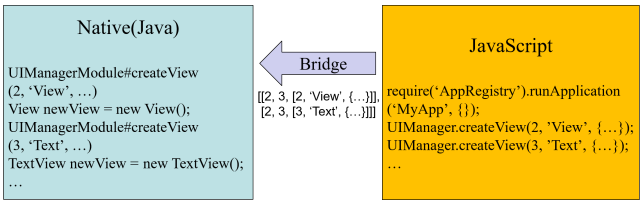

図 18　JavaScript 側からの呼び出し

解析手段としては,Java 側で JavaScript 側に受け渡すような関数を発見し, 実際のアプリの動きと同様の手順でその関数と引数を JavaScript 側で発見するようなものを実装することで, 二つの言語の解析結果（コールグラフなど）を接続可能になると考えられる．

6.3.2　デバイスの機能の呼び出し

Native モジュールは Java 側で図 19 のように主に ReactContextBaseJavaModule を継承して定義されている．

```
1  public class ToastModule extends
       ReactContextBaseJavaModule {
2
3    private static final String DURATION_SHORT_KEY
         = "SHORT";
4    private static final String DURATION_LONG_KEY =
          "LONG";
5
6    public ToastModule(ReactApplicationContext
         reactContext) {
7      super(reactContext);
8    }
9  }
```

図 19　モジュール例（ToastModule）

そして JavaScript 側から呼び出せるようにするために図 20 のように名称の定義や, アノテーションを付与する．

このアノテーションを付与することでメソッドを JavaScript から使用できるようになり, フレームワークによる引数の対応の処理なども可能になる．そして JavaScript 側からは図 21 のように呼び出す．

```
1  @Override
2    public String getName() {
3      return "ToastAndroid";
4    }
5  ...
6
7  @ReactMethod
8    public void show(String message, int duration)
         {
9      Toast.makeText(getReactApplicationContext(),
           message, duration).show();
10   }
```

図 20　名称の定義及びアノテーション付与

```
1  import ToastAndroid from './ToastAndroid';
2  ToastAndroid.show('Awesome', ToastAndroid.SHORT);
```

図 21　JavaScript からの呼び出し

解析手段としては,ReactContextBaseJavaModule,getName() やアノテーションを発見した後,JavaScript 側で指定されている処理を見つけ, 二つの言語の解析結果を接続可能になると考えられる．

引数については図 22 のようにフレームワークが処理しているので, 解析もそれに倣い検出を行う．

```
1  Boolean -> Bool
2  Integer -> Number
3  Double -> Number
4  Float -> Number
5  String -> String
6  Callback -> function
7  ReadableMap -> Object
8  ReadableArray -> Array
```

図 22　引数の対応

また上記にないような何かの結果などの JavaScript 側に受け渡す引数については図 23 のように Java 側でコールバック関数によって定義されているので, その関数を検出することで解析可能になると考えられる．

6.3.3　WebView

始めに説明したように React Native では WebView を使用せず, ネイティブを呼び出している．しかし, ネイティブの UI ではなく, 単純にネイティブの機能を使用できる WEB ブラウザのようなものを実装する場合は多い．その際は ReactNative 用の Webview ライブラリなどを使用して,

```
1  public class UIManagerModule extends
       ReactContextBaseJavaModule {
2  ...
3    @ReactMethod
4    public void measureLayout(
5        int tag,
6        int ancestorTag,
7        Callback errorCallback,
8        Callback successCallback) {
9      try {
10       measureLayout(tag, ancestorTag,
             mMeasureBuffer);
11       float relativeX = PixelUtil.toDIPFromPixel(
             mMeasureBuffer[0]);
12       float relativeY = PixelUtil.toDIPFromPixel(
             mMeasureBuffer[1]);
13       successCallback.invoke(relativeX, relativeY
             );
14     } catch (IllegalViewOperationException e) {
15       errorCallback.invoke(e.getMessage());
16     }
17   }
```

図 23 コールバック関数による引数の定義

一部今回提案していた対象のハイブリッドアプリと同じような形で実装されている．この場合は提案手法や今回使用したツールなどの解析方法が流用できる可能性が高いと考えられる．

6.4 課題

まずはシステムを完成させる必要がある．今回の手法については, 現状では開発者による解析対象の選択後, それぞれの場合によって既存の手法を使い分けている状態（Cordova の場合 Brucker らの手法適応後 HybriDroid ではなく静的解析ツールである WALA での解析を行っている状態）なので, これらの手法を参考にしながら, うまく組み合わせていく必要がある．

解析対象の選択についてはフレームワークが複数ある以上同一の方法では解析できないので, それら対応できるようにするためにもそれぞれのフレームワークに関しての仕組みをより詳しく知る必要がある．そうすることで対処できる範囲の拡大や一部自動化についても実装していきたい．

しかし考察で紹介したように, 今のハイブリッドアプリの状況からすると今回提案した手法では, 最新のハイブリッドアプリに対応できないことが多い．そのためツールとしては React Native などに対応できるものを実装するのが望ましい．そのために考察で述べた内容に加え, より詳しくフレームワークの構造を把握し, 今回の手法も参考にしてツールを実装していく必要がある．

7. おわりに

近年需要が増えているハイブリッドアプリは便利な反面, セキュリティ面で問題を抱えている．そしてまだハイブリッドアプリのセキュリティ対策はあまり進んでいない．

本研究ではハイブリッドアプリ開発者が, 安全なアプリを開発できるように, 開発者の補助となるような手法の提案を行った．提案した手法により, 様々なハイブリッドアプリに関して, 情報流出となるような箇所の検出が可能な事が分かった．また現在も発展し続けているハイブリッドアプリにも対応していく必要がある事が分かった．今後はハイブリッドアプリの動向を追うとともに, それに対応したシステムの実装の完了を目指し, 安全なハイブリッドアプリ開発においての具体的な有用度の調査などを行っていきたい．

謝辞

本研究は JSPS 科研費 16K12411, 17H04705 の助成を受けたものです．

本研究を遂行するにあたり，研究の機会と議論・研鑽の場を提供して頂き，御指導頂いた国立情報学研究所／東京大学 本位田 真一 教授をはじめ，活発な議論と貴重な御意見を頂いた研究グループの皆様に感謝致します．

参考文献

[1] A.D.Brucker, M.Herzberg, *On the Static Analysis of Hybrid Mobile Apps*, Engineering Secure Software and Systems, pp.72-88, 2016.

[2] S.Lee, J.Dolby, S.Ryu, *HybriDroid: static analysis framework for Android hybrid applications*, Automated Software Engineering, pp.250-261, 2016.

[3] X.Jin, X.Hu, K.Ying, W.Du, H.Yin, G.N.Peri, *Code injection attacks on html5-based mobile apps: Characterization, detection and mitigation*, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, p.66-77, ACM, 2014.

[4] HybriDroid, https://github.com/SunghoLee/HybriDroid.

[5] Code Injection Attacks on HTML5-based Mobile Apps, http://www.cis.syr.edu/ wedu/android/JSCodeInjection.

[6] S.Arzt, S.Rasthofer,C.Fritz, E.Bodden, *FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps*, Programming Language Design and Implementation, pp.259-269, 2014.

[7] Y.L.Chen, H.M.Ree, A.B.Jeng, T.E.Wei, *DroidCIA: A Novel Detection Method of Code Injection Attacks on HTML5-Based Mobile Apps*, Trustcom/BigDataSE/ISPA, IEEE Vol.1, pp.1014-1021, 2015.

[8] 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, 情報科学技術フォーラム講演論文集 14.4, pp.13-18, 2015.

[9] 落合 淳, 嶋村 誠, 河野健二, *Taint Analysis* によるスパイウェア検知手法の回避, 研究報告システムソフトウェアとオペレーティング・システム 2009-OS-112(8), 1-8, 2009.

[10] Cordova, https://cordova.apache.org.

[11] React Native, https://facebook.github.io/react-native/.