Web アプリケーションの安全な実行方式

中村 洋介† 二村 和明† 伊藤 栄信†

†株式会社富士通研究所
211-8588 神奈川県川崎市中原区上小田中 4-1-1
{nkmr,kazuaki.nimura,itou.hidenobu}@jp.fujitsu.com

あらまし スマートフォンにおいて，マルウェアの実行を防ぐには，利用者がアプリのインストール時に表示されるアクセス許可の警告を確認し，アプリ本来の機能に不要なアクセス許可が含まれていればインストールしないという利用者のスキルに依存する手続きに頼っているため，全ての利用者に対して効果的にマルウェアの実行を防ぐことは難しい．本報告では，ネイティブアプリと遜色の無い機能を提供可能とされている Web アプリを対象とし，潜在的にマルウェアを含むアプリを実行させながら，ランタイムで不正なコードのみを実行させない，アプリの安全な実行方式を提案する．そして，Android スマートフォンを用いて提案方式を実装し，提案方式が有効であることを示す．

Secure Execution of Web Application

Yosuke Nakamura† Kazuaki Nimura† Hidenobu Ito†

†Fujitsu Laboratories LTD.
1-1, Kamikodanaka 4-chome, Nakahara, Kawasaki
211-8588, JAPAN
{nkmr,kazuaki.nimura,itou.hidenobu}@jp.fujitsu.com

Abstract For preventing unwanted behavior by malware in smart phone, it has been taken an approach that user checks the warnings of access permission which are displayed at the time of installation of an application and then user permits the access controls. However because it depends on IT literacy of users, it is doubtful whether it is suitable for all the users. In this paper we propose an approach that is automatically produce an access control list from an application as a Cloud service and run the application with the access controls at smart phone. Then we implemented the proposed approach using Android smartphone and showed the effectiveness.

1 はじめに

昨今，パーソナルコンピュータ（PC）並の性能を持ったモバイル端末であるスマートフォンの普及が進んでいる．

スマートフォンの普及につれて，利用者がマルウェアを組み込まれたアプリケーションをインストール，実行してしまう事例が増えている．特に Android において広がりを見せており，例えば，Android アプリのポータルサイトである Android Market[1]では，2010 年 8 月に AndroidOS.FakePlayer，2011 年 3 月にも Android.Geimini が見つかっている．2012 年にはスマートフォンの出荷が 5 億台に達し PC を

抜くと推測され，マルウェアの矛先が PC からスマートフォンへ向けられることで，スマートフォンのマルウェアが増えると予測される[2][3] .

このようなマルウェアを組み込まれたアプリは，Android Market で公開されているアプリを正規手段で入手し，これを逆アセンブリしマルウェアを混入してアセンブリ，再パッケージングして Android Market に公開する，という手順で公開可能なことが知られている[4]. また，アプリは Android Market 以外からインストールすることも可能で，Android Market が安全に運用されたとしてもマルウェアの入り込む余地がある.

もし，マルウェアを組み込まれたアプリをダウンロード，実行してしまうと，例えば利用者の端末から個人情報（クレジットカードの番号など）を抜き取り，悪意あるユーザのサーバへ勝手に個人情報を送信してしまう，という不正行為が行われ，利用者が被害を受けてしまう.

マルウェアの実行を防ぐ方法は，利用者がアプリのインストール時に表示されるアクセス許可の警告を確認し，アプリ本来の機能に必要ないアクセス許可が含まれていればインストールしないことであるが，これでは利用者のスキルに依存しており，全ての利用者がマルウェアの実行を防ぐことは難しい.

そこでアプリの実行中にランタイムでマルウェアの動作のみを止めることが出来る手法を提案するとともに，その実装および評価を行う.

以下，2 章では関連研究として HTML5 アプリのセキュリティ動向に触れる. 3 章では，提案アーキテクチャを示し，4 章で実装詳細を示す. 5 章で，マルウェアを組み込んだアプリを実際に動作させた結果と，各種マルウェアに対する本手法の有用性について評価した結果を示す. 最後に 6.でまとめる.

2 関連研究

1 章で Android アプリのセキュリティに触れたが，本章では Web アプリの一つである HTML5 アプリのセキュリティについて触れる.

HTML5 アプリの危険性はいくつかの文献で指摘されている. 例えば，文献[5]では HTML5 の新機能である Client-side Storage が，XSS(cross-site scripting)の脆弱性をついて攻撃される危険性があることを指摘している. また，文献[6]においても，Client-side Storage やクロスドメイン通信による攻撃の危険性が指摘されている.

また，HTML5 アプリを安全に動作させる研究も行われており，例えば文献[7]ではプログラムのソースにインラインで Capability を記述する手法を提案している。これは，HTTP のタグに Capability 要素を追加し，これで Ajax や HTTP の Get/Post，Cookie の使用や Hyperlink の Click に対し有効/無効を設定することが出来る. Capability を適切に設定することで一般的なアタックに対してアクセス制御を行うことが出来ることを示し，マルウェア対策として有効な手段であると言える. しかし，この方法は Capability という特殊な HTTP タグに対応するブラウザを Google Chrome ブラウザの拡張として実装しており，汎用的に使用できない.

3 課題と提案方法

3.1 課題

マルウェアの実行を防ぐ方法は，Android アプリでは利用者がアクセス制御を設定するしかなく，その設定は利用者に委ねていた. そのため，効果にはばらつきが出る. そこで，利用者のスキルによらない対策が必要であった. また，Web アプリにいたっては，汎用的な防御方法すらない状態であり，マルウェア対策が必要であった.

3.2 提案方法

そこで，アクセス制御の自動化手法を提案する. これは，クラウド側でアプリを解析しアクセスコントロールリスト（ACL）を作成し，端末で

ACL に従って関数の実行をランタイムでチェックすることで，マルウェアのみを実行させない方式である．本方式では，マルウェアのみを実行させないようにするため，アプリ単位の大きなアクセス制御ではなく，実行関数単位に細かくアクセス制御する．

本方式の構成を図 1 に示す．

本方式では，クラウド側に解析エンジンと解析ルール，端末側に ACL チェッカを設ける．

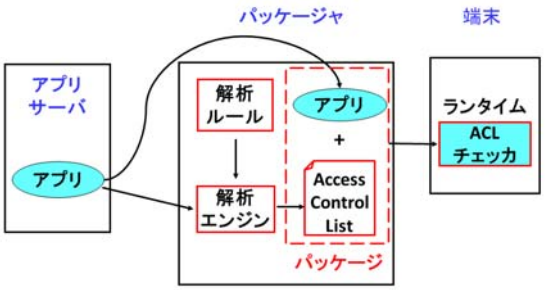

図 1. 本方式の構成

・解析エンジン：アプリを構文解析して，ACL 作成に必要なオブジェクトやメソッド，パラメータを抽出し ACL を作成するエンジン．

・解析ルール：解析エンジンで抽出するオブジェクトやメソッド，パラメータと抽出結果に従って ACL に記述する内容を紐付けたルール．

・ACL チェッカ：アプリが行うデバイスアクセスやネットワークアクセスが ACL で許可されているか否かをランタイムでチェックし，許可されていないものは実行を禁止するモジュール．

ACL チェッカがあるメソッドの実行を禁止した後もアプリの実行は継続される．ACL 対象外のメソッドや ACL で許可されたメソッドは実行を許し，ACL で許可されていないメソッドのみピンポイントで実行を禁止するため，危険性のないコードは全て実行され，危険なコードだけ実行を止めることができる．

4　実装

本章では，本方式の構成要素である ACL，端末，パッケージャそれぞれについて実装の詳細を述べる．

4.1　ACL

はじめに ACL について述べる．ACL は，W3C Working Draft の Widget Packaging and XML Configuration[8]および W3C Widget Access Request Policy[9]で定義された内容に沿って記載する．具体的には，デバイスアクセスは文献[7]の 7.13 feature 要素として，ネットワークアクセスは文献[8]の 5 章，access 要素として記載する．

ACL の記述例を図 2 に示す．この例では”http://www.aaabbb.com/”へのネットワークアクセスとカメラデバイスへのアクセスが許可されている．

```
<?xml version="1.0" encoding="UTF-8"?>
<widget xmlns="http://www.w3.org/ns/widgets"
    xmlns:wac="http://wacapps.net/ns/widgets"
    id="http://example.org/helloworld" version="1.0 Beta" height="200"
    width="300" viewmodes="floating" wac:min-version="2.0">
    <icon src="icon.png"/>
    <content src="helloworld.html" encoding="UTF-8"/>
    <access origin="http://www.aaabbb.com/*"/> ⇒ ネットワークアクセス許可
    <feature name="http://wacapps.net/api/camera"/> ⇒ カメラデバイス許可
    <name short="HelloWorld">Hello World</name>
    <description>Hello World Widget.</description>
    <license href="http://license.example.org/">Example license Copyright (c)
    2011.</license>
    <author email="myname@host" href="http://foo-
    bar.example.org/">myname</author>
</widget>
```

図 2. ACL 記述例

4.2　端末

次に，端末側の実装について述べる．端末側の実装構成を図 3 に示す．

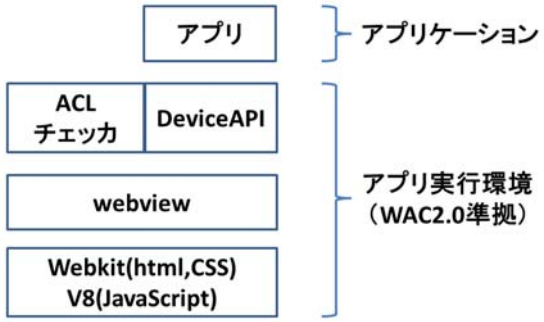

図 3. 端末側の実装構成

端末側にアプリ実行環境を開発した．アプリ実行環境は，HTML／JavaScript エンジンである Webkit／V8，レンダリングエンジンである WebView は Android 標準を流用し，アプリに

対し WAC2.0 準拠のデバイスアクセスを提供する DeviceAPI および ACL チェッカは WAC2.0[10]に準拠させるため新規に開発した. WebView は，デバイスやネットワークアクセスを捕捉するイベントハンドラとして使用する.

今回 WAC2.0 を採用したのは，WAC[11] (Wholesale Applications Community) がキャリアや OS に依存しないオープンなモバイルアプリ環境の仕様策定や産業基盤の構築を目指す業界団体であり，同団体が定めた WAC2.0 はモバイルアプリ環境の標準仕様になり得るものと考えたためである.

4.3 パッケージャ

パッケージャには 3 章に示すように解析エンジンと解析ルールを実装した.

解析ルールでは，解析対象のオブジェクトおよびメソッド，抽出されるべきパラメータと ACL に記載するルールが紐付けされる. デバイスアクセスは，WAC2.0 Widget Security and Privacy[12]の 3.4.1.4 で定義されているオブジェクトやメソッドであり，今回は camera オブジェクトの captureImage メソッドとした. ネットワークアクセスについては，文献[12]の 3.1.4.5 に記載された XMLHttpRequest オブジェクトの open メソッドとし，第二引数を抽出する.

解析エンジンは，アプリを構文解析して変数や属性をツリー構造で抽出する. この中から解析ルールに記載されたオブジェクトやメソッドを探し，4.1 章の記述ルールに沿って ACL を作成する.

4.4 対象とするアプリ

最後に、本方式で解析やアクセス制御の対象とするアプリについて触れる. スマートフォンで実行可能なアプリには，ネイティブアプリ，Web アプリ，Hybrid Mobile アプリ [13]の 3 種類が存在する.

ネイティブアプリは，現在最も一般的な形式であり実行する端末に特化したアプリである. Android アプリはこれに該当する.

Web アプリは，ブラウザ上で実行され端末にインストールする必要がないアプリである. HTML5 アプリはこれに該当する.

Hybrid Mobile アプリは，WAC や PhoneGap[14]といった開発環境で作成することができる. ネイティブアプリのように端末のデバイスなどを自由に扱うことができる Web アプリであり，今後の普及が期待される.

ネイティブアプリはプログラム解析が大変であり，ランタイムで ACL チェックするのに OS に手を加えなくてはいけないため，本方式の検証が容易ではない. 一方，Hybrid Mobile アプリはプログラム解析が比較的容易で，ランタイムの ACL チェックを OS に手を加えず実装できるため，Hybrid Mobile アプリを対象に解析エンジンや ACL チェッカの実装を行うことにした.

5 評価

本章では，理論的な評価と実験結果により本手法の有効性を検証する.

5.1 理論的評価

1 章で挙げた Android Market のアプリにマルウェアを組み込む手法に対し，本方式が効果的であることを図 4 で示す.

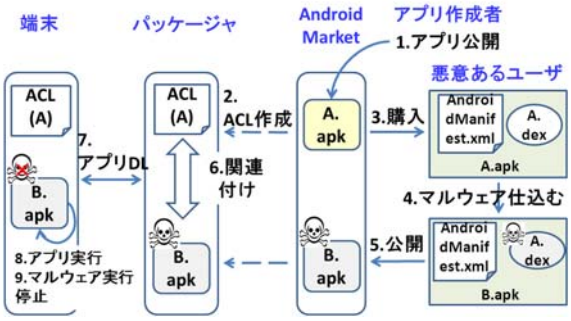

図 4. 本方式の適用例

はじめにパッケージャが公開されたアプリ A に対し，アクセスして良い URL とデバイス一覧が記載された ACL(A)を作成する. その後悪意あるユーザがマルウェアを含むアプリ B を公開

したとしても，パッケージャが dex ファイル名からアプリ B と ACL(A)を関連付け，端末にはアプリ B と ACL(A)をセットでダウンロードさせる．

端末ではアプリ B が実行されるが，ACL チェッカが ACL(A)を参照するため，アプリ B で追加されたコード（すなわちマルウェアそのもの）のデバイスやネットワークアクセスのみがアクセス制御されることになる．このように，アプリ A ではアクセスしない URL へ飛べなくなり，端末内の情報をアプリ A では意図しないサーバへアップロードすることが禁止され，アプリを動作しても安全性を保つことができる．

ただし，本方式が有効になるのは「ACL 作成時にアプリはマルウェアを含まないこと」，「作成した ACL は書き換えられていないこと」を満たす場合であり，アプリや ACL に証明書を付加するなどして正当性を担保する必要がある．

次に，マルウェアが組み込まれたアプリを端末で実行する場合の有効性を，2 章の文献[7]で挙げられた Web アプリに対する攻撃を例に，本方式で実行を防ぐことができるかどうかで検証する．

1. The orkut worm：JavaScript のインラインフレームで，onload イベントを使用して外部サーバの Javascript を読み込ませ実行させるものである．
2. Untrusted Input：Web 広告のように，ホストページのリンクを選択した時点で JavaScript が読み込まれ実行されるもの．ホストページと同じ特権が JavaScript に与えられるため，悪意ある JavaScript が実行された場合に危険である．
3. ClickJacking：ダミーの透過 iframe を重ね，ボタンやリンクをクリックしたときに異なる動作を発動させたり，異なるサイトへ飛ばしたりする．
   攻撃 1～3 は，URL が静的に JavaScript のソースコードに記載されているため解析エンジンにて抽出でき，ACL によるアクセス制御が可能である．
4. XSS(Cross Site Scripting)：Web サーバの脆弱性をつき，入力フォームに直接 JavaScript を書くなどして不正な動作をさせる．
   URL は Script 動作時に生成されるので，これを防ぐには解析エンジンを端末側にも設ける必要がある．

以上の考察を表 1 にまとめた．動的に URL を生成する XSS を防ぐには端末側でも解析が必要になるため、有効性を△としている．

攻撃	有効性
The orkut worm	○
Untrusted Input	○
ClickJacking	○
XSS	△

表 1. 本方式の有効性

5.2 実験結果

今回は，カメラデバイスで写真を取り外部サーバに写真をアップロードするマルウェアを仕込んだ Hybrid Mobile アプリを作成し，本方式を持つ端末，持たない端末それぞれで上記アプリを動作させた．アプリは HTML および JavaScript で構成し，マルウェアは WAC2.0 camera オブジェクトと XMLhttpRequest を使用している．

実験環境はサーバとスマートフォンからなる．

サーバ装置には FMV-S8350 を使用した．パッケージャとアプリサーバの機能および，マルウェアが写真をアップロードする外部サーバの機能を持つ．

スマートフォンには，Android 2.3 搭載の F-12C を使用した．この端末はアプリ実行環境を持つ．

まず，本方式を適用しない場合のアプリ実行結果は図 5 に示すとおり，スマートフォンで写真が撮影され外部サーバ上にアップロードされていて，マルウェアが実行されたことが分かる．

次に，本方式を適用した場合のアプリ実行結果を図 6 に示す．図 6 のとおり，スマートフォン上にカメラデバイスを使用できない旨の警告画面が表示され，外部サーバに写真がアップロードされておらず，マルウェアに対してアクセス制

御が行われたことが分かる．また，警告画面で OK を押した後はアプリを継続して使用可能である．

図 5.本方式を非適用時の実行結果

図 6. 本方式を適用時の実行結果

この結果から，静的にデバイスやネットワークアクセスを仕込んだマルウェアに対して本方式が有効であることを実証した．

6 まとめ

スマートフォンにおけるマルウェア対策は利用者のスキルに依存した手続きとなっているため，クラウドと連携することで端末のデバイスやネットワークへのアクセス制御を自動化し，利用者のスキルに依存しないアプリの安全な実行方式を提案した．本方式は標準仕様に準拠したアプローチで，端末の変更を少なく実現できるため，実用的な方法である．そして，Hybrid Mobile アプリを対象として本方式を実装，実験して有効性を示した．本方式では静的に埋め込まれたマルウェアは防げるが，XSS による攻撃やマルウェアが暗号化された場合に課題がある．これらは，端末側に提案したパッケージャと同じ解析エンジンを持たせることで対処可能である．

今後は， Web アプリに対する一般的な攻撃方法を実装した Hybrid Mobile アプリやマルウェア検体を使用した評価を行い，出回っているマルウェアに対する本方式の有効性を検証していく．

参考文献

[1] Android Market https://market.android.com/

[2] McAfee Labs, McAfee 脅威レポート:2011 年 第 1 四半期(2011) http://www.mcafee.com/japan/media/mcafeeb2b/international/japan/pdf/threatreport/threatreport11q1.pdf

[3] 狙われるスマートフォン,日経 TechOn,2011 http://techon.nikkeibp.co.jp/article/FEATURE/20110208/189406/

[4] 日経エレクトロニクス 2011 年 7 月 25 日号

[5] Alberto Trivero : Abusing HTML5 Structured Client-side Storage, 2008

[6] Steve Mansfield-Devine : Divide and conquer: the threats posed by hybrid apps and HTML5, Network Security(2010)

[7] Tongbo Luo, Wenliang Du : Capability-Based Access Control for Web Browsers

[8] Widget Packaging and XML Configuration, W3C Working Draft(2011) http://www.w3.org/TR/widgets/

[9] Widget Access Request Policy, W3C Candidate Recommendation(2010) http://www.w3.org/TR/widgets-access/

[10] Device Specification, WAC2.0 Approved Release Version(2011) http://specs.wacapps.net/2.0/jun2011/

[11] WAC Developer Website http://www.wacapps.net/

[12] Widget Security and Privacy, WAC2.0 Approved Release Version(2011) http://specs.wacapps.net/2.0/jun2011/core/widget-security-privacy.html

[13] Sam Alexander : Building hybrid mobile applications with PhoneGap and IBM WebSphere Portlet Factory, IBM developerWorks(2011) http://www.ibm.com/developerworks/websphere/zones/portal/portletfactory/proddoc/phonegap/

[14] Phonegap, http://www.phonegap.com/