Vol.2016-CSEC-74 No.24
Vol.2016-SPT-19 No.24
2016/7/15

HTMLハイブリッドアプリケーションの静的解析によるCSP自動適用手法

竹内 俊輝[1]　齋藤 彰一[1]

概要：モバイル端末上で動くアプリケーションとしてHTMLハイブリッドアプリケーションがある．これはモバイル端末のOSの種類を問わず動作することから注目を集めている．しかしHTMLハイブリッドアプリケーションにはXSS脆弱性が懸念される．そこで本論文では，XSSを防ぐことが可能であるContent Security Policy(CSP)をHTMLハイブリッドアプリケーションに対して静的解析を用い自動適用させる手法を提案する．この手法を用いいることにより，XSSを防ぐことが可能となる．また，マーケットで配布されているアプリケーションに対しこの手法を適用し，CSP適用後アプリケーションの動作を考察し，この手法の有効性について検証する．

キーワード：HTMLハイブリッドアプリケーション，XSS，CSP，静的解析，モバイル端末

1. はじめに

モバイル端末上で動くアプリケーションとしてHTMLハイブリッドアプリケーションがある．HTMLハイブリッドアプリケーションとはHTML5，JavaScript，CSSで構成されるアプリケーションである．これはモバイル端末のOSの種類を問わず動作をするクロスプラットフォームであることから注目を集めている．しかし，HTMLハイブリッドアプリケーションはXSS脆弱性が懸念される．

XSSとは攻撃者が任意のJavaScriptをWebアプリケーションで実行できる脆弱性を利用した攻撃である．攻撃者はこの攻撃によって，HTMLページの改ざんや，ブラウザに保存してあるcoookieを盗むことが可能となる．このためXSSへの対策がWebアプリケーションでは必須である．XSSに対する一般的な対策として，HTMLページへのテキストデータを入出力時にテキストデータ内の特殊文字をエスケープする方法がある．この方法はXSSに対して効果的な方法ではあるが，人為的なミスであるチェックの見逃しやエスケープ自体の失敗が発生しやすく，多くの場合で不完全な対策となる．

これに対し，XSSへの完全な防御策としてContent Security Policy(以下，CSPとする) [1]がある．CSPはHTMLのコンテンツを制限する機構で，XSSに対する包括的な保護を提供する．CSPは現在多くの主要ブラウザで利用できる [2]．CSPのアプリケーションでの利用も広がっており，TwitterやFacebookでは既に利用されている．CSPをアプリケーションで利用するには，HTMLレスポンスヘッダ内にCSPヘッダを記述し，ポリシーを宣言する．CSPヘッダで宣言したポリシーによって許可されたHTMLコンテンツのみがWebアプリケーションで利用可能となる．CSPはXSSに対し効果的な防御策であるが，CSPによりアプリケーションの動作を制限することで，アプリケーションの振る舞いや構造に影響が出ることがある．またアプリケーション開発者はHTMLファイルごとにポリシーを宣言する必要があり負担が大きい．そこで本提案では，HTMLハイブリッドアプリケーション開発負担を軽減し，CSPを効果的に利用するための機構を提案する．

本提案機構はHTMLハイブリッドアプリケーションを静的解析し，自動でポリシーの宣言を行う．さらに，ポリシーを宣言したことによって生じるHTML，JavaScriptの変更を行う．本提案機構により，アプリケーション開発者にCSPを利用しやすい環境を提供し，かつアプリケーションのXSS脆弱性を無くす．

以後，2章で提案の基盤であるHTMLハイブリッドアプリケーションと攻撃方法であるXSSについて述べ，3章で防御方法であるCSPについて述べる．4章で提案とその詳細を述べ，5章で実装について述べる．続いて6章で評価を行う．7章で関連研究について触れ，8章でまとめる．

[1] 名古屋工業大学
Nagoya Institute of Technology

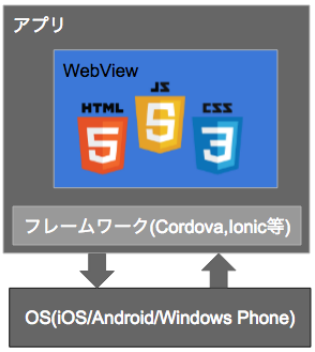

図 1 HTML ハイブリッドアプリケーションの構成

2. HTML ハイブリッドアプリケーションと XSS

本章では HTML ハイブリッドアプリケーションと XSS について述べる．初めに XSS の攻撃対象である HTML ハイブリッドアプリケーションの構成と仕組みについて述べる．次いで XSS について述べる．

2.1 HTML ハイブリッドアプリケーション

HTML ハイブリッドアプリケーションはモバイル端末の各 OS に共通の WebView を用いるため，どの OS でも共通に動作するアプリケーションである．HTML ハイブリッドアプリケーションは，HTML，JavaScript，CSS で構成される．HTML は言語の仕様として JavaScript，CSS を HTML ファイルに記述することが可能である．しかし，この仕様により，2.2 節で述べる XSS 攻撃が可能となる．多くの HTML ハイブリッドアプリケーションの一般的な構成を図 1 に示す．HTML ハイブリッドアプリケーションは，フレームワーク (例えば，Cordova や Ionic など) を用いて，OS のサービスを利用する．またネイティブコンポーネントである WebView は，アプリケーションの HTML ファイル，JavaScript ファイル，CSS ファイルを読み出し，画面描画を行う．

2.2 XSS

XSS はアプリケーションに対するコードインジェクション攻撃の一種である．XSS は 3 種類の攻撃方法に大別される．Stored XSS，Reflected XSS，DOM Based XSS である．この中でも HTML ハイブリッドアプリケーションで発生する脆弱性は DOM Based XSS である．

DOM Based XSS は，外部から入力された文字列に基いて動的に HTML を生成する際に，開発者の意図しないスクリプトが混入する脆弱性である．この脆弱性により cookie の盗難やページの改ざん等が起こるため，XSS の対策は必須である．通常の 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 GET メソッド，POST メソッドから受け取る文字列等で攻撃が行われる．しかし，HTML ハイブリッドアプリケーションではそれらに加え，画像や音声ファイルのメタ情報，SMS や電話帳，Wi-Fi，Bluetooth のアクセスポイント名などの様々な文字列によって攻撃可能であることが指摘されている [3]．

このような攻撃に対しても，テキストデータの入力時，もしくは出力時にエスケープ処理を行うことで防ぐことができる．しかし，エスケープ処理は人為的なミスが生じやすいため脆弱性を根絶するにはアプリケーション開発者への負担が大きい．CSP では XSS に対して包括的な保護を提供できる．

3. CSP

本章では CSP について述べる．はじめに CSP の概要について述べ，CSP の根幹をなすポリシーディレクティブについて述べた後，CSP の問題点について述べる．

3.1 CSP の概要

CSP は XSS を防ぐことが可能な機構であり，アプリケーションに対してホワイトリスト形式の Web コンテンツの読み込み先の制御を行う．CSP は HTTP レスポンスヘッダにヘッダ名として `Content-Security-Policy` を指定し，ヘッダのコンテンツとしてポリシーディレクティブを宣言することで利用可能となる．WebView は，アプリの HTML ファイルを読み込む際，head タグ内にあるポリシーディレクティブを受け取り，ポリシーディレクティブの宣言通りに動作するよう，HTML の実行を強制する．

3.2 ポリシーディレクティブ

Web ページのコンテンツの読み込み先を制限するための宣言である．サーバのホスト名をポリシーディレクティブに記述することで, JavaScript, 外部プラグイン, CSS などの読み込み先を指定されたサーバに限定する．ポリシーディレクティブの形式は `directive-name resource-domian` という形式をとる．`directive-name` は複数あるため，`directive-name` の数だけこの形式を繰り返し，細かく制限をすることが可能である．また繰り返す際にはポリシーディレクティブをセミコロンで区切る必要がある．

ポリシーディレクティブの meta タグを用いた設定例を図 2 に示す．図 2 で宣言されたポリシーディレクティブは，”`default-src *`”，”`script-src 'self' http://www.test.jp`”，”`style-src 'self' 'unsafe-inline'`”の 3 種類である．図 2 の例における `script-src` とは script タグの読み込み先に制限をかける場合の `directive-name` である．同様に `style-src` は style タグの読み込み先に制限をかけるための `directive-name` である．`resouce-domain` 部には読み込みを許可するホスト名を記述する．例えば，`http://www.test.jp/test/index.js` ファイルを読み込

```
<meta http-equiv="Content-Security-Policy" content="default-src *;
script-src 'self' http://www.test.jp; style-src 'self' 'unsafe-inline';">
```

図 2 ポリシーディレクティブの例

みたい場合は resouce-domain 部に http://www.test.jp を記述する．また resouce-domain 部にはホスト名以外にもキーワードを記述することが可能である．キーワードには none，self，unsafe-inline，unsafe-eval の 4 種類がある．none キーワードはすべてのホストからの読み込みを拒否する．self キーワードは自ドメインからのコンテンツの読み込みのみを許可する．HTML ハイブリッドアプリケーションの場合，アプリケーションのパッケージ内にあるコンテンツの読み込みの許可を意味する．unsafe-inline キーワードは directive-name が script-src もしくは style-src の時のみ使用可能であり，HTML ファイルでインラインスクリプトやインラインスタイルを利用できるようにする．unsafe-eval キーワードは directive-name が script-src の時のみ使用可能であり，文字列を JavaScript として評価する JavaScript のメソッド (例えば，eval や Function など) を利用できるようにする．unsafe-inline キーワードと unsafe-eval キーワードは，利用すると CSP の効果がなくなる．また図 2 で示したように，resource-domain 部にはメタ文字の*を用いることもできる．これはどのホストからの読み込みも許可するという意味である．よって図 2 で示したポリシーディレクティブは，これを記述した HTML ファイル内において JavaScript は自ドメインと www.test.jp からの読み込みを許可する．CSS は自ドメインからの読み込みと，インラインスタイルを許可する．それ以外のコンテンツに関してはどのドメインからの読み込みも許可することを意味する．

HTML ハイブリッドアプリケーションで問題となりうる DOM Based XSS は，ポリシーディレクティブの directive-name の script-src に unsafe-inline と unsafe-eval を含めないことによって防ぐことが可能となる．

3.3 CSP の問題点

アプリケーションが CSP の効果を得るためには 2 つの手順が必要となる．まず，すべての HTML ファイルに対してポリシーディレクティブを設定と宣言を行う．次に，ポリシーディレクティブの宣言に沿うように HTML と JavaScript を修正する．しかし，この 2 点をアプリケーション開発者が行う場合は時間が必要なだけでなくエラーも生じうる．またポリシーディレクティブの適切な設定には専門の知識も必要である．このためアプリケーション開発者に対する負担が大きいという問題がある．

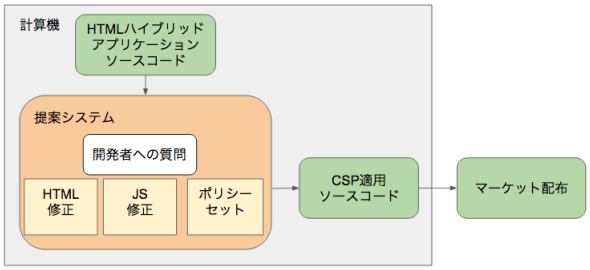

図 3 提案機構の全体図

4. 提案

本章では，既存の HTML ハイブリッドアプリケーションに CSP を自動適用する機構の動作概要とその詳細を述べる．

4.1 提案の概要

既存の HTML ハイブリッドアプリケーションに CSP を自動適用する機構を提案する．本提案機構を利用することで，ソースコードの修正をアプリケーション開発者が行うことなく CSP を適用し，HTML ハイブリッドアプリケーションにおける XSS を防ぐことが可能となる．既存の HTML ハイブリッドアプリケーションに CSP を適用する際に必要となる作業は，ポリシーディレクティブの設定と HTML と JavaScript の修正である．本提案機構ではこれらの項目を静的解析のみを用いて自動で行う．本提案機構の動作概要図を図 3 に示す．まず本提案機構は，動作時に開発者にインラインスタイルを許可に関する質問を行い，許可する場合であれば，ポリシーディレクティブの style-src に unsafe-inline キーワードを設定する．次に，アプリケーションのソースコードを CSP を適用したソースコードに変更する．アプリケーション開発者は本提案機構により変更されたソースコードをアプリのマーケットに登録することでアプリケーションの配布を行う．また本提案機構は HTML ハイブリッドアプリケーションのページ遷移を HTML ファイルで管理しているものと想定して実装を行っている．

4.2 ポリシーディレクティブの設定

ポリシーディレクティブを，XSS 脆弱性が発生しないように設定する．そのため 3.2 節で述べたように，directive-name の script-src にはキーワードの unsafe-inline と unsafe-eval を含めない．また，style-src に関しては，開発者が許可した際にはキーワード unsafe-inline を含める．アプリケーションに必要であるコンテンツが外部にある場合，ホスト名を HTML ファイルと JavaScript ファイルの解析によって取得する．

4.3 HTML の修正

4.2 節で述べたポリシーディレクティブの設定を行うことで，HTML ファイルではインラインスクリプト，インラインスタイル，イベントハンドラ，javascript:URI 形式の動作ができなくなる．このため，これらの修正が必要である．ただしインラインスタイルによって発生する XSS は Web ブラウザの一つである Internet Explorer でのみ発生する．本提案では WebView で動く HTML ハイブリッドアプリケーションを対象としているため，4.2 節で述べたように，インラインスタイルを修正するかは開発者に任せるものとする．

4.4 JavaScript の修正

4.3 節と同様に JavaScript ファイルの修正も必要である．JavaScript ファイルでは動的に script タグを生成するメソッドと文字列を JavaScript として評価するメソッドの動作ができなくなるため，これらの修正が必要である．

5. 実装

本章では提案機構の実装について述べる．ポリシーディレクティブの具体的設定方法について述べ，その後 HTML ファイルの修正方法と JavaScript ファイルの修正方法について述べる．

実装にあたり，HTML ファイルと JavaScript ファイルの解析が必要となる．HTML ファイルの解析には JSOUP [4] を用いる．JSOUP は Java で作成されたオープンソースの HTML パーサであり，DOM を利用し HTML ファイルの解析と修正に利用できる API を提供する．次に，JavaScript の解析には，独自のツールを Java を用いて開発して使用した．JavaScript ファイルの修正は，特定の関数 [*1] が修正対象となる．さらに，この修正対象の関数が引数に変数を保つ場合には，その引数を解析する必要がある．本ツールは，特定関数を取得するクラスと，JavaScript ファイル内の変数を取得するクラスを提供している．

5.1 ポリシーディレクティブの設定

ポリシーディレクティブは，基本となるポリシーディレクティブを用意し，それをアプリケーションに合わせて拡張する方式で設定する．基本となるポリシーディレクティブは `default-src *; script-src 'self'; style-src 'self';` とする．この基本のポリシーディレクティブでは，JavaScript と CSS の読み込みをアプリケーションのパッケージ内に限定する制限である．また基本のポリシーディレクティブの `script-src` にはキーワード `unsafe-inline` と `unsafe-eval` を含めないことにより，XSS の防御が可能

[*1] document.write，document.writeln，document.createElement, eval, Function, setInterval, setTimeout が本提案機構で修正対象となる関数である．

```
1: $.ajax({
2: type:'POST',
3: dataType:'json',
4: url: 'http://www.test.jp/api.php,
5: …});
```

図 4 jQuery を用いた外部ホストとの通信例

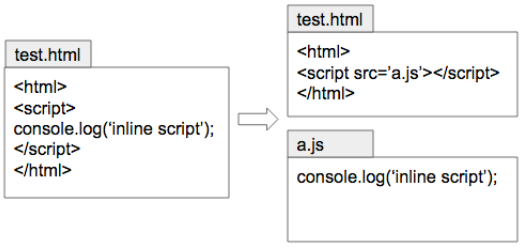

図 5 インラインスクリプトの修正方法

となる．また，`style-src` には開発者がインラインスタイルを許可した場合はキーワード `unsafe-inline` を含める．

`resource-domain` 部に記述するホスト名は，HTML ファイル及び JavaScript ファイルを解析した結果によって記述する．まず，HTML ファイルの解析においては，script タグもしくは style タグの src 属性で参照している部分が外部ホストであるかを調べる．また，JavaScript ファイルの解析においては，jQuery を用いて外部のホストと通信するホスト名を調べる．jQuery を用いた外部ホストとの通信の例を図 4 に示す．jQuery を用いて外部ホストと通信する場合には，図 4 中の 4 行目の url に記載されているホスト名を取得する．

5.2 HTML の修正

HTML ファイルの修正について述べる．インラインスクリプトとインラインスタイル，イベントハンドラ，javascript:URI の修正を行うことでポリシーディレクティブに沿う HTML ファイルとなる．まず，インラインスクリプトとは HTML ファイルに script タグを用いて JavaScript が記述した部分のことである．本機構では script タグ内の JavaScript を正規表現で抽出し，外部ファイルへ書き出す．さらにこの外部ファイルを呼び出すように変更する．修正例を図 5 に示す．図 5 のように script タグ内の JavaScript を外部ファイルに書き出した上で，script タグの src 属性を用いて指定する．

インラインスタイルは，style タグもしくは style 属性を用いて HTML ファイルに CSS を記述するものである．インラインスタイルへの対処方法はインラインスクリプトと同様に，該当の部分を正規表現を用いて外部ファイルに記述し，参照するように修正する．

次にイベントハンドラについて述べる．イベントハンドラとは，ユーザの動作や操作に対して特定の処理を与えるための命令である．本機構では図 6 のように DOM を利用した addEventListener 関数を利用し，イベントを登録

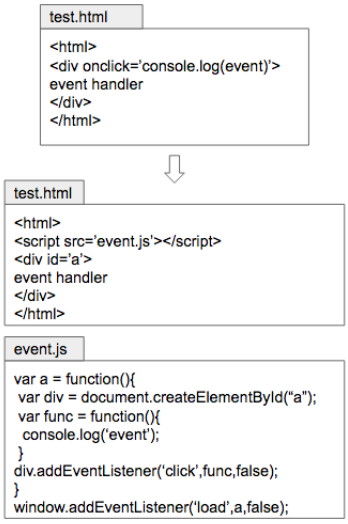

図 6 イベントハンドラの修正方法

し直す修正を行う．addEventListener で利用できるイベント名はイベントハンドラで使用する名称と差異があるが，どのイベント名を用いればよいかは W3C で定義されている [5]．

最後に javascript:URI について述べる．これは html のタグに付加可能である href 属性に「javascript:...」と記述することにより JavaScript が実行可能となるものである．この形式もイベントハンドラの時と同様の手法で修正することが可能である．但し，javascript:uri 形式とイベントハンドラが同一タグに設定されている時，優先権の処理があるため，その部分を考慮し，修正を行う．

5.3 JavaScript の修正

JavaScript ファイルの修正について述べる．動的 HTML タグ作成を利用した script タグの作成の修正と，文字列を JavaScript として評価するメソッドの修正を行うことでポリシーディレクティブに沿う JavaScript ファイルとなる．動的に HTML を作成するメソッドは，DOMAPI には document.write，document.writeln，innerHTML，outerHTML の 4 種類が存在する [3]．JavaScript ファイルがこれらのメソッドを利用し script タグを作成し，HTML ファイルに挿入する場合，作成した script タグが外部ホストに存在する JavaScript ファイルを参照する形式に修正する．document.write と document.writeln，innerHTML と outerHTML はそれぞれ同じ修正方法が利用できる．document.write を用いた場合の修正例を図 7 に，innerHTML を用いた場合の修正例を図 8 に示す．

文字列を JavaScript として評価するメソッドには setInterval，setTimeout，eval，Funtion() の 4 種類が存在する．このうち setInterval と setTimeout は引数として文字列と関数を取ることができる．文字列を引数として取る場合がポリシーディレクティブに違反するため，関数を引

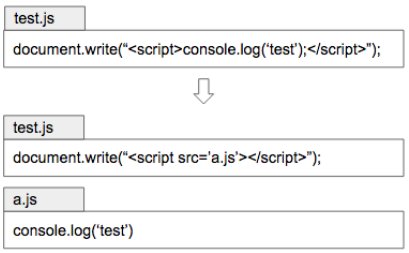

図 7 document.write 形式の修正方法

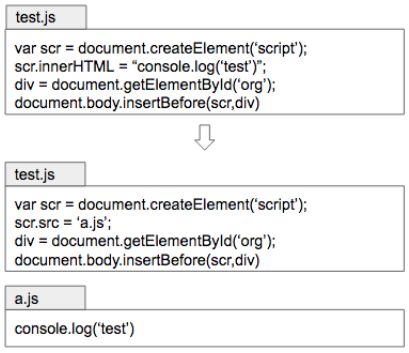

図 8 innerHTML 形式の修正方法

数として取るように修正を行う．eval は使用方法が複数あり，使用方法に合わせた修正を行う．使用方法は Richard らによると 10 種存在する [6]．4 種類において対処方法が述べられていないが，それらは eval の引数をユニコードに変換する JavaScript エンコードを行うことによって XSS の発生を防ぐことができる．この方法で eval の対処を行う際は，ポリシーディレクティブの `script-src` の `resource-domain` 部に `unsafe-eval` を用いる．

6. 評価

本章では提案機構の評価について述べる．提案機構を利用した際の HTML ハイブリッドアプリケーションのエラー出力数の結果及びソースコードの改変数の結果を示し，考察を行う．

6.1 評価方法

提案機構を用い，CSP を適用させた HTML ハイブリッドアプリケーションに対して評価を行う．また本評価では，HTML ハイブリッドアプリケーションが動作をするか否かの判定を Android エミュレータを用いて確認する．エラー出力個数に関しては Web ブラウザ Google Chrome のデベロッパーツールを利用する．ソースコードの改変数は提案機構から直接求める．なお，改変数は，インライン系 (インラインスクリプトとインラインスタイル)，イベントハンドラ，javascript:URI，document.write 系 (document.write と document.writeln と innnerHTML と outerHTML)，eval 系 (eval と Function)，setInterval 系 (setInterval と setTimeout) の 6 項目に分類する．ソース

表 1　評価環境

提案機構動作 OS	Ubuntu 14.04
提案機構開発言語	Java 1.8.0
エミュレータ	Android Emulator
エミュレータ OS	AndroidOS 5.1.1

表 2　アプリケーションの種類

アプリ名	分類名
地理院地図	地図アプリ
GreenMahjong	ゲームアプリ
Rocket Chat Cordova	チャットアプリ
Fresh Food Finder	料理店検索アプリ
Angular JS Mobile	AngularJS を用いたサンプルアプリ
2048	ゲームアプリ
hextris	ゲームアプリ
clumsy bird	ゲームアプリ

コード改変から評価までに使用した機器を表 1 に示す．評価には表 2 に示す 11 個のアプリケーションを用いた．

6.2　エラー出力結果

まず，今回評価したアプリケーションの起動直後の画面において，元のアプリケーションとの差異は見られなかった．次に，本提案を利用し，CSP を適用したアプリケーションのエラー出力数結果を表 3 に示す．エラーが発生したアプリケーションに対して，発生したエラーの内容と考察を行う．まず地理院地図アプリでは外部からのスクリプトファイルの読み込みができないことによるエラーが発生した．これは JavaScript ファイル内に通信するホスト名の記述がなく，アプリケーション操作時に動的に通信先が決定するためであった．この方式の対処としては事前にアプリケーションを動作させ，通信するホスト名を取得する必要がある．このアプリケーションは他に 4 箇所の eval が含まれていたため，ポリシーディレクティブに手動でホスト名を加え，他の 4 箇所の評価を行った．その結果，2 箇所の eval では正しく変更が行われたが，残り 2 箇所は引数の変数の中身が不明であったため，完全な変更ができなかった．この点においても動的解析を用い，事前に変数にどのような値が代入されるかを調べることにより，正しく変更できると思われる．

次に MineSweeper アプリのエラーについて述べる．このアプリケーションでは jQuery を利用した CSS 関連のエラーが発生した．アプリケーションの動作が問題なく行えることと，目に見える見た目の変化がないためエラーの直接的な原因は不明である．しかし，これはポリシーディレクティブの `style-src` の `resource-domain` 部にキーワード `unsafe-inline` を入れることでエラーが出力されなくなることと，CSS を用いた XSS は IE でのみ起こりうるため，HTML ハイブリッドアプリケーションに対する対処法としては `unsafe-inline` を挿入する方法でエラーを取り除くことができる．

表 3　エラー出力結果

アプリ名	エラー個数
地理院地図	1
GreenMahjong	0
MineSweeper	6
Rocket Chat Cordova	0
Fresh Food Finder	7
AngularJS Mobile	0
2048	0
hextris	0
clumsy bird	0

最後に Fresh Food Finder アプリについて述べる．このアプリケーションのエラーはアプリケーション内のボタンを押した時に動作する JavaScript が CSP によって制限されたため発生した．本提案機構はアプリケーションのページ遷移を HTML ファイルで管理するものと想定しているため，JavaScript のライブラリを用いたページ遷移には対応できない．このアプリではページの遷移に ViewNavigator ライブラリを用いていた．そのため，今後の課題として対応するライブラリを増やすことが挙げられる．

また，地理院地図と clumsy bird では外部にあるファイルを読み込んでいるが，これらのファイルは CSP によって制限されることがなかったため，ポリシーディレクティブの設定も正しく行われているといえる．

これらの結果から，提案機構は基本的な対策としては十分であると言える．しかし，より高度な修正のために，事前に動的解析を用い，通信する外部ホストの特定，また eval 系の変数の値の解析結果を用いたより高い制度の変更機構が必要であると考える．

6.3　ソースコード改変数結果

次にソースコードの改変数について調べた結果を表 4 に示す．9 個のアプリケーションで評価した結果，7 個のアプリケーション (Rocket Chat Cordova，2048 以外) ではソースコードの改変の必要があった．まず，GreenMahjong と Angular JS Mobile はインラインスタイルの修正である．実際に XSS の原因となる JavaScript のコード改変ではないため，提案機構の動作開始前の質問時にインラインスタイルの分割をしない選択肢をアプリケーション開発者が選択した場合は変更は必要ない．一方，地理院地図，MineSweeper，hextris，clumsy bird のソースコード改変は，JavaScript のコードを外部ファイルに正しく分割することを確認した．また，地理院地図と hextris のイベントハンドラの修正も正しく修正できたことを確認した．eval 系の修正について，6.2 節で述べたように地理院地図と Fresh Food Finder で行ったが，地理院地図の 4 箇所のうちの 2 箇所のみが正しく変更できており，残りの 2 箇所と

表 4　ソースコード改変数

アプリ名	インラインスタイル	イベントハンドラ	javascript:URI	document.write 系	eval 系	setInterval 系
地理院地図	1	1	0	0	4	0
GreenMahjong	1	0	0	0	0	0
MineSweeper	1	0	0	0	0	0
Rocket Chat Cordova	0	0	0	0	0	0
Fresh Food Finder	0	0	0	0	1	0
Angular JS Mobile	1	0	0	0	0	0
2048	0	0	0	0	0	0
hextris	1	2	0	0	0	0
clumsy-bird	10	0	0	0	0	0

Fresh Food Finder の eval 形式は正しく修正されていないため，事前の動的解析が必要である．なお，今回評価したアプリケーションで使用されていなかった javascript:URI，document.write 系，setInterval 系は自作したテストコードで修正できることを確認した．これらの結果から，提案機構で修正を行うファイルのうち HTML ファイルの修正は十分であると言える．JavaScript ファイルの修正については 6.2 節で述べたように，事前の動的解析を用いることでより高い精度の変換機構が必要である．しかし，事前に動的解析を用いなければ修正できないもの以外の修正は可能である．

7. 関連研究

本章では関連研究について述べる．Web アプリケーションに対する CSP のアプローチに関する手法を述べ，本提案との比較を行う．

7.1 deDacota

deDacota [7] は Web アプリケーションを静的解析し，データやソースコードを分割する手法である．インラインスクリプトを別の JavaScript ファイルに分割する．しかし，インラインスタイルやイベントハンドラに対しては方針を述べるのみで留まっており，実装を行っていない点で提案機構と異なる．またこの点を実装していないことより，ポリシーディレクティブの設定にも制約が出る．

7.2 AutoCSP

AutoCSP [8] は，PHP を利用した Web アプリケーションに対して自動で CSP を適用する手法である．テイント解析を行うことで，信頼できる JavaScript と信頼出来ない JavaScript に分割し処理を行うことで，Stored 型 XSS に対しより高度な防御を実現している，また PHP を利用して HTML 構文を作り出す場合にも対応している．しかし，AutoCSP では JavaScript ファイルの修正について実装していないことより，本提案機構とは異なる．

7.3 CSPAider

CSP Aider [9] は Web アプリケーションに対するポリシーディレクティブを提示する機構である．Web ページを解析することにより適切なポリシーディレクティブを設定する．しかし，CSP Aider は CSP の仕様策定段階で提唱されたものであるため，ポリシーディレクティブの形式が現行利用されているものと違う．そのため現在利用されている主なブラウザでは利用できない，またこの機構はポリシーディレクティブの提唱のみであり，HTML，JavaScript ファイルの修正が無い点が本提案機構と異なる．

7.4 セキュリティガイドラインに準拠したアプリケーション作成支援

セキュリティガイドラインに準拠したアプリケーション作成支援 [10] は OWASP によって公開されているセキュアコーディングの指針に非準拠のアプリケーションを準拠したアプリケーションとして動作させる仕組みを提案している．この方式での限界点として，悪性なスクリプトを持つ URL やサニタイズを通り抜けるペイロードなどの攻撃などの対策が十分ではない．これはセキュリティガイドラインに準拠しているためであり，セキュリティガイドラインが更新されれば軽減できるものとある．なお，本提案機構では許可の無い入力等による JavaScript は実行できないため，この提案方式と違い問題はない．

8. おわりに

本稿では，既存の HTML ハイブリッドアプリケーションを静的解析を用いて CSP に自動適用する機構を提案した．CSP は XSS に対する包括的な保護を提供する機構である．本提案では HTML ハイブリッドアプリケーションに対し，静的解析を用いて CSP を自動適用することによりアプリケーション開発者の労力を減らし，かつ CSP により XSS に防ぐことが可能である．既存の HTML ハイブリッドアプリケーションに対し，本提案機構を利用し評価した結果，ポリシーディレクティブの設定，HTML ファイルの修正は静的解析のみで確実かつ正確に修正できることを確認した．JavaScript ファイルの修正に関しては，ア

プリケーション動作時に動的に決まる処理，もしくはアプリケーションファイル内に記述がない処理など，事前の動的解析を用いなければ修正できない処理以外の修正が可能であることを確認した．今後は現在対応できないライブラリを用いたエラーに関する調査を行い，より多くのアプリケーションで提案機構が利用できる方式を検討する．また端末上で本提案機構を利用することでアプリケーション利用者が提案機構を利用できる方式を検討する．

参考文献

[1] W3C: ”Content Security Policy 2.0”, http://www.w3.org/TR/CSP.
[2] caniuse: ”Can I use content security policy?”, http://caniuse.com/contentsecuritypolicy.
[3] Jin, X., Hu, X., Ying, K., Du, W., Yin, H. and Peri, G. N.: Code injection attacks on HTML5-based mobile apps: Characterization, detection and mitigation, *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ACM, pp. 66–77 (2014).
[4] jsoup: ”jsoup: JAVA HTML Parser”, http://jsoup.org.
[5] W3C: ”UI Events Specification”, https://www.w3.org/TR/DOM-Level-3-Events/.
[6] Richards, G., Hammer, C., Burg, B. and Vitek, J.: The eval that men do, *ECOOP 2011–Object-Oriented Programming*, Springer, pp. 52–78 (2011).
[7] Doupé, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C. and Vigna, G.: deDacota: toward preventing server-side XSS via automatic code and data separation, *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, ACM, pp. 1205–1216 (2013).
[8] Fazzini, M., Saxena, P. and Orso, A.: AutoCSP: Automatically Retrofitting CSP to Web Applications, *the Proceedings of the 37th International Conference on Software Engineering (ICSE)* (2015).
[9] Javed, A.: Csp aider: An automated recommendation of content security policy for web applications, *IEEE Symposium on Security and Privacy* (2011).
[10] 鈴木富明，白井丈晴，小林真也，川端秀明，西垣正勝ほか：セキュリティガイドラインに準拠したアプリケーション作成支援に関する一提案，情報処理学会研究報告. マルチメディア通信と分散処理研究会報告， Vol. 2015, No. 8, pp. 1–8 (2015).