4ZA-08

コードインジェクション攻撃対策のための CSP 構成自動化についての提案

梅内　翼†　　小倉　加奈代†　　Bhed Bahadur Bista†　　高田　豊雄†

†岩手県立大学　ソフトウェア情報学部

1. はじめに

Web はインターネット上での情報交換のための主要なプラットフォームとなっており，今日では多数の Web アプリケーションが公開されている．しかし，Web アプリケーションには XSS 攻撃をはじめとするコードインジェクション系の攻撃の発生が懸念されている．本稿では，このような攻撃の対策として有力である Content Security Policy(CSP)の構成を，HTML ファイルと JavaScript ファイルの解析を通して自動化する手法について提案する．また，本手法を実装し，模擬的な Web アプリケーションに対して実行することで本手法が有効に動作することを検証する．

2. CSP の概要と問題点

CSP は，ユーザのブラウザに対して読み込みを許可するリソースの種別や，その提供元を制限するホワイトリスト型のセキュリティレイヤーである．本稿執筆時点においては，CSP Level 3 のワーキングドラフトが公開されており，一部の Web ブラウザにおいて実装が行われている**[1]**.

CSP は各種のディレクティブと，そのディレクティブに指定する値の組合せで構成される．本提案手法で利用するディレクティブの一覧を表 1 に示す．表 1 の各ディレクティブに適切な値を設定することで，CSP を有効に適用させることが可能である．例として，`script-src` ディレクティブに指定可能な値の種別を抜粋して表 2 に示す．

適切に構成された CSP は XSS 攻撃をはじめとするコードインジェクション攻撃への対策として有効に動作するが，手作業による構成は非常に困難であるという問題がある．本稿では，各種コンテンツの解析を通して CSP の構成を自動化することで，CSP 構成に関わる作業負担を低減させつつセキュアな CSP 構成を実現させる手法について提案する．

3. 先行研究

Javed は，Web コンテンツのクロールによって CSP 構成を自動化する CSP AiDer を提案している**[2]**

また，Fazinni らは，PHP 環境における CSP 構成の自動化を行う AutoCSP を提案している**[3]**.

本提案手法は，よりセキュアで柔軟な CSP 構成を実現可能な最新の規格を用いている点，任意のサーバ環境において実行可能な点が先行研究とは異なる．

4. 提案手法

本稿では，読み込んだ HTML コンテンツおよび JavaScript コンテンツをクライアントサイドで解析することで，CSP の構成を自動化する手法を提案する．本提案手法は，大きく分けて 3 つのフェーズからなる．以下に各フェーズの詳細を示す．また，図 1 に動作フロー図を示す．

Proposal on CSP Configuration Automation for Defense against Code Injection Attack

†Tsubasa Umeuchi, Kanayo Ogura, Bhed Bahadur Bista, Toyoo Takata, Iwate Prefectural University

表 1 利用するディレクティブ一覧

ディレクティブ	内容
`script-src`	JavaScript リソースを指定.
`style-src`	CSS リソースを指定.
`connect-src`	XMLHttpRequest や Fetch にて接続可能なオリジンを指定.
`base-uri`	`<base>` タグにて指定可能な URL を指定.

表 2 `script-src` に指定可能な値

種別	意味
`'self'`	コンテンツ提供元と同一のオリジンを許可.
URL	指定した URL を許可.
`'sha256-<base64>'`	base64 形式の SHA-256 ハッシュ値が一致するスクリプトを許可.
`'strict-dynamic'`	nonce かハッシュ値で許可されたスクリプトおよびそのスクリプトから non-“parser-inserted”な方法で読み込まれたスクリプトを許可.

第 1 フェーズ：Loading フェーズ

ユーザは CSP 構成の対象となる Web コンテンツの URL を指定する．システムは，その URL にアクセスし，対象となるコンテンツを取得する．

第 2 フェーズ：Analysis フェーズ

取得したコンテンツを HTML コンテンツと JavaScript コンテンツに分けて解析する．

HTML コンテンツの解析では，「`javascript:`スキームのリンク」，「外部を参照する JavaScript」，「外部を参照する CSS」，「インラインスクリプト」，「インラインスタイル」，「`style` 属性」，「インラインスクリプト内の URL」を抽出する．JavaScript コンテンツの解析では，「`innerHTML/outerHTML` プロパティの使用箇所」，「`document.write(ln)`関数の使用箇所」，「`eval` 関数の使用箇所」，「`Function` コンストラクタの使用箇所」，「スクリプト内の URL」を抽出する．

第 3 フェーズ：Configuration フェーズ

上記の解析を通して得られたデータを基に CSP の構成を行い，ユーザに結果を提示する．

4.1. CSP 構成の詳細

本節では，構成する CSP の詳細をディレクティブ別に説明する．

`script-src` ディレクティブ

`script-src` ディレクティブには CSP Level 3 の仕様を活用した Strict CSP**[4]**と呼ばれる nonce と`'strict-dynamic'`を用いるアプローチを改良したアプローチを採用する．具体的には，`script-src` ディレクティブに各スクリプトの SHA-256 ハッシュ値および`'strict-dynamic'`を設定する．これは，nonce を不正に読み出

す攻撃を考慮したアプローチである. なお, CSP Level 3 の仕様を実装していないブラウザに対して後方互換性を維持するために'self'および URL も設定する.

style-src ディレクティブ

style-src ディレクティブには'self', URL およびインラインスタイルの SHA-256 ハッシュ値を設定する. なお, style 属性の扱いに関してはブラウザ毎に実装が異なる. 本アプローチでは安全性を優先するため警告の表示のみを行い, style 属性の使用を許可する'unsafe-inline'キーワードの追加は行わない.

connect-src ディレクティブ

connect-src ディレクティブに設定する URL はユーザとのインタラクションを通じて確認する.

base-uri ディレクティブ

base-uri ディレクティブには<base>タグインジェクションによる XSS 攻撃対策として'none'を設定する.

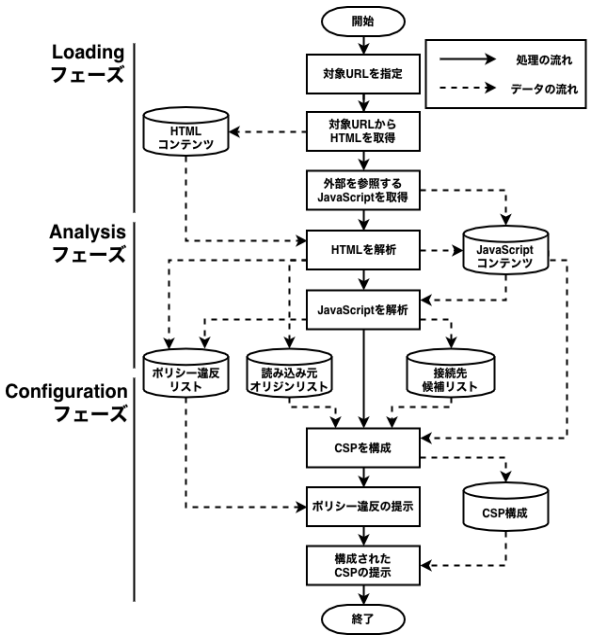

図 1 動作フロー図

4.2. 実装

実装は Python3.7 にて行った. ライブラリとして, コンテンツ読み込みのために Requests を, HTML 解析のために BeautifulSoup をそれぞれ用いた. また, URL の検出には正規表現を用いた. なお, 実装したシステムは GitHub 上で公開している**[5]**.

5. 提案手法の評価

5.1. 評価の概要

4 章で実装した提案手法について, 以下に示す表 3 の 3 つの観点に基づいて定性評価および定量評価を行う.

表 3 評価の観点

観点	内容
Security	提示された CSP を適用することで XSS 攻撃を防げているか.
Functionality	提示された CSP を適用した時, 本来の機能が阻害されていないか.
Performance	本システムが充分高速に動作するか.

5.2. 評価結果

Security

本システムによって構成された CSP を適用したところ, 評価用の全ての環境において XSS 攻撃によって注入されたスクリプトが実行されないことが確認された.

Functionality

CSP Level 3 の仕様である'strict-dynamic'に対応したブラウザとして Google Chrome 71.0 を, 対応していないブラウザとして Safari 12.0 を選択し, テストを行った. その結果, eval 関数や Function コンストラクタの実行, style 属性の使用といった本手法におけるアプローチで禁止されている機能以外の機能は正しく動作することが確認された.

Performance

本システムを検出項目を何も持たないコンテンツ(plain), 外部の JavaScript を参照するコンテンツ(external), 検出項目を全て含んだコンテンツ(all)に対して動作させ, その実行時間を計測した. なお, 本システム自体の実行時間とは直接関係のないネットワークアクセスに関する時間やユーザからの入力を待つ時間は除いている. 図 2 に示す結果の通り, 本システムが充分高速に動作することが確認された.

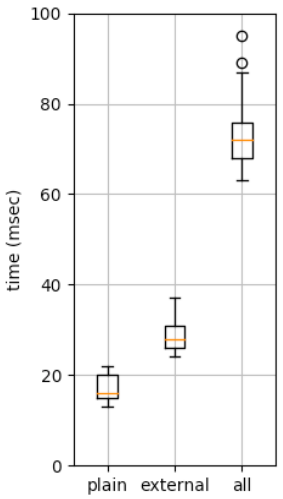

図 2 実行時間(N=30)

6. おわりに

本稿では, 各種コンテンツの解析を通して CSP 構成を自動化することで, CSP 構成に関わる作業負担を低減させつつセキュアな CSP 構成を実現させる手法を提案した. また, その手法を実装し, 模擬的な Web アプリケーションに対して実行した結果, 合理的なパフォーマンスの下で本来の機能性を保ったままコードインジェクション攻撃の対策に有効な CSP を自動構成できることを確認した.

参考文献

[1]W3C:Content Security Policy Level 3(オンライン), 入手先<https://www.w3.org/TR/CSP3/> (参照 2018/12/16).

[2] Javed A.: CSP AiDer: An Automated Recommendation of Content Security Policy for Web Applications, IEEE Symposium on Security & Privacy (2011).

[3]Fazzini, M., Saxena, P., and Orso, A.: Auto CSP: Automatically Retrofitting CSP to Web Applications, Proc. *ICSE*, pp.336-346 (2015).

[4]Google:Strict CSP(オンライン), 入手先<https://csp.withgoogle.com/docs/strict-csp.html> (参照 2018/12/20).

[5]GitHub:Szarny/CSP-Configuration-Automation-Public(オンライン), 入手先<https://github.com/Szarny/CSP-Configuration-Automation-Public> (参照 2019/01/10).