SURE 静岡大学学術リポジトリ
Shizuoka University REpository

An Efficient implementation of Whitelist-based XSS Attack Detection

メタデータ	言語: jpn 出版者: 公開日: 2018-02-07 キーワード (Ja): キーワード (En): 作成者: 井上, 佳祐, 本多, 俊貴, 向山, 浩平, 大木, 哲史, 西垣, 正勝 メールアドレス: 所属:
URL	http://hdl.handle.net/10297/00024643

Copyright© 2018 The Institute of Electronics, Information and Communication Engineers

SCIS 2018 2018 Symposium on Cryptography and Information Security
Niigata, Japan, Jan. 23 - 26, 2018
The Institute of Electronics, Information and Communication Engineers

ホワイトリスト型 XSS 攻撃検知の効果的な実現方法に関する検討
An Efficient implementation of Whitelist-based XSS Attack Detection

井上 佳祐* Keisuke Inoue　本多 俊貴* Toshiki Honda　向山 浩平* Kohei Mukaiyama

大木 哲史* Tetsushi Ohki　西垣 正勝* Masakatsu Nishigaki

あらまし インターネットの普及に伴い，Web アプリケーションに対するサイバー攻撃が増加している．本稿では，XSS 攻撃に焦点を当て，その対策を検討する．現在，XSS 攻撃の対策として，Web アプリケーションの開発，運用，コードレビューに関するガイドラインが策定されている．また，Web アプリケーションに対する入力文字列のサニタイジングも一般的になっている．しかし，XSS 攻撃の方法は多岐に渡っており，また，不正者によって日々新たな脆弱性や攻撃法が探索されている現状において，XSS 攻撃を完全に無害化することは容易ではない．このような多様な攻撃に対しては，ホワイトリスト型の対策を採用し，予め規定された動作のみを許可するという防御戦略が有効である．しかし，必要十分なホワイトリストを作成するための方法論が確立しておらず，これがホワイトリスト型 XSS 攻撃対策の効果を限定的なものにしてしまっていた．その解決のために，本稿では，Web アプリケーションのスクリプトに対して，「仕様書に示されたとおりの動作」をホワイトリストとして定義することで，XSS 攻撃を検知する手法を提案する．本手法では，Web アプリケーションの開発工程の中で必ず行われるテストに着目し，スクリプトのホワイトリストをテストケースから自動生成する方法を確立する．これにより，従来の Web アプリケーションの開発工程を変更することなく，仕様書と逸脱することのないホワイトリストを，それぞれの Web アプリケーションのスクリプトごとに作成することが可能となる．本提案手法をテストツールである「Selenium」を用いて実装・評価し，提案手法の有効性を示す．

キーワード XSS，ホワイトリスト，攻撃検知，テストケース

1 はじめに

コンピュータ性能の向上とネットワークの広帯域化に伴い, Web 上のコンテンツは従来の静的なコンテンツを主とした Web ページから，動的なコンテンツを扱う Web アプリケーションへと変移している．現在，大多数の Web サイトが，jQuery[1]をはじめとした JavaScript ライブラリや WordPress[2]などの CMS を利用するようになり，クライアント側・サーバ側を問わず，プログラムで動的に生成される仕組みを含んでいる．その結果，従来（Web アプリケーションではない）のアプリケーションで発生するような脆弱性が, Web サービスにおいても発生するようになり, Web アプリケーションに対するサイバー攻撃の急増へと繋がった．

クロスサイトスクリプティング（XSS）攻撃も Web アプリケーションに対するサイバー攻撃の 1 つである．XSS は，Web サイトがユーザから受け取る入力値を適切なものかどうかをチェックする機能や，有害な入力値を無害化する機能の欠陥によって生まれる Web アプリケーションの脆弱性である．この脆弱性を悪用することで，攻撃者は任意のスクリプトをユーザのブラウザで実行させることが可能である．攻撃された結果，セッション ID などを奪取されてしまい，その Web サイトにおいて攻撃者によるユーザへのなりすましを許したり，攻撃者がユーザに対してドライブバイダウンロード攻撃を行

* 静岡大学，〒432-8011 静岡県浜松市中区城北 3-5-1，Shizuoka University, 3-5-1, Johoku, Naka-ku Hamamatsu, Shizuoka 432-8011, Japan.

うきっかけとして利用される可能性がある．

この脆弱性が発生する主な原因は，Web アプリケーション開発時に行うべき XSS 対策に漏れが生じてしまっていることにある．現在，サニタイジングによって無害化を図る方法が一般的な XSS 対策として普及している[3]．しかし，セキュリティ対策に十分な費用を投入している大手 IT 企業の Web サービスにおいても同様の脆弱性が発見されている[4]ことを考えると，多様化した悪意ある入力を完全に無害化することは，開発者の能力にかかわらず，困難な作業であることが推測される．また，今まで知られていなかった脆弱性が新たに発見される場合もある．したがって，サニタイジングによって無害化を図ることは，十分かつ容易な対策となり得ていないという現状にある．

SQL インジェクションなど他の Web アプリケーションの脆弱性は開発者側に影響があるのに対し，XSS 攻撃はユーザに直接影響を及ぼす脆弱性であるため，効果的な追加対策を行うことが急務である．関連研究[6][7][8]では，スクリプトの動作を制限することによって XSS 攻撃を無効化する方法が提案されている．しかし，スクリプトの動作を規定するためのポリシーは開発者が記述する必要があり，必要十分なポリシーを作成するための方法論が確立していない．

そこで筆者らは，Web アプリケーションのスクリプト動作に対して，Web アプリケーション開発の初期工程で作成される仕様書で示されたとおりの動作をホワイトリストとして定義し，実行時のスクリプトを検証することで，XSS 攻撃を検知し防ぐ手法を考案する．さらに，Web アプリケーション開発の最終工程として行われるテスト工程を利用して，検知に必要となるホワイトリストを半自動的に作成する手法を提案する．そして Web アプリケーションの結合テストツールとして利用されることが多い Selenium[5]を用いて提案手法を実装し，本手法の有効性を評価する．

2 XSS 攻撃とその一般的な対策

2.1 XSS 脆弱性の種類

クロスサイトスクリプティング（Cross Site Scripting : XSS）の脆弱性には，大きく分けて次に示す 3 つの種類が存在する．この脆弱性は，非持続性か持続性か，サーバ側の処理で発生するのか，クライアント側の処理で発生するのかによって分類される．

2.1.1. Reflected XSS

ユーザからの入力をパラメータとして受け取り，サーバ側のプログラムがそのパラメータをそのままクライアント側に返すタイプの Web アプリケーション（図 1）において発生する可能性のある XSS 脆弱性である．有害な入力値をサーバ側でパラメータとして入力値をそのまま利用することによって発生する．サーバではその値を保持しないため攻撃の持続性はない．

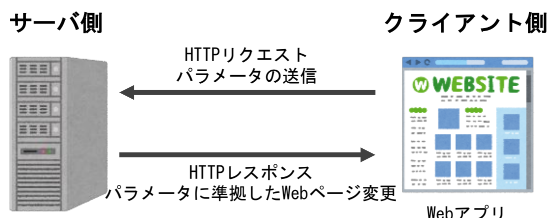

図 1 Reflected XSS の例

2.1.2. Stored XSS

ユーザからの入力をパラメータとして受け取り，サーバ側のプログラムがその入力値をデータベース(DB)などのストレージに保存し，その保存された値を元に，クライアントに返す応答内容を変えるタイプの Web アプリケーション（図 2）において発生する可能性のある XSS 脆弱性である．サーバがパラメータとして受け取った入力値はサーバの DB に保存されており，その保存された値を読み込むことによって発生するため，攻撃の持続性がある．

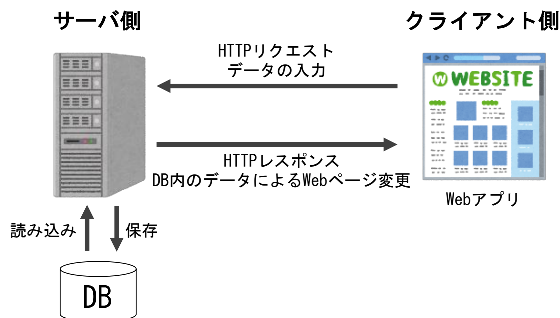

図 2 Stored XSS の例

2.1.3. DOM-Based XSS

ユーザからの入力をパラメータとして受け取り，クライアント側のプログラムがそのパラメータをそのまま DOM に反映するタイプの Web アプリケーション（図 3）において発生する可能性のある XSS 脆弱性である．受け取ったパラメータの値はクライアント側で処理される．サーバではその値を保持しないため攻撃の持続性はない．

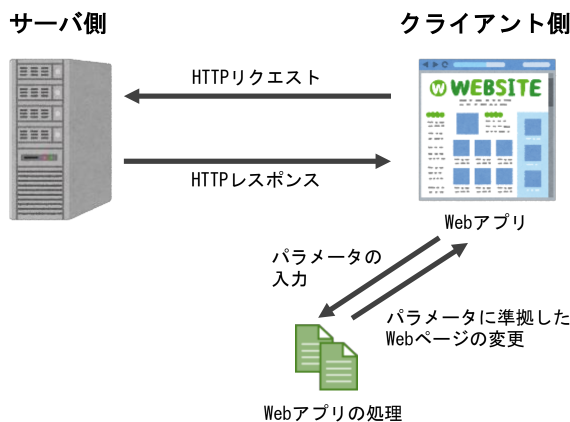

図 3 DOM-Based XSS の例

2.2　XSS 攻撃のメカニズム

本節では代表的な XSS 攻撃として Reflected XSS を例に挙げ，そのメカニズムを示す．XSS 攻撃は，「攻撃者」，「被害者」，XSS 脆弱性のある「標的 Web アプリケーション」の 3 者が関与して成り立つ（図 4）．

攻撃者は，標的 Web アプリケーションに対して有害なリクエストを発生させるためのリンク情報（標的 Web アプリケーションの URL＋XSS 脆弱性を突くパラメータ）を E メールに記載するなど，何らかの方法で被害者に送信する（図 4 ①）．被害者は，そのリンクにアクセスすることにより，攻撃者の指定した有害なリクエストを被害者のブラウザを通して標的 Web アプリケーションに送信する（図 4 ②）．標的 Web アプリケーションは，有害なスクリプトを含んだ Web コンテンツを被害者のブラウザに対するレスポンスとして送信する．被害者のブラウザはレスポンスに含まれた有害なスクリプトも DOM として認識し実行してしまう（図 4 ③）．

この脆弱性が悪用され，ブラウザ上で任意のスクリプトが実行されてしまうと，標的 Web アプリケーションのセッション ID などが奪取され，その Web アプリケーションに対して攻撃者による被害者へのなりすましを許す恐れや，攻撃者が偽のログインページを表示させて被害者の情報を盗む恐れ，また攻撃者が用意した悪意ある Web サイトへと被害者を誘導してドライブバイダウンロード攻撃を行う恐れが生じることとなる．このような有害なスクリプトの動作が被害者に対して明示的に示されることは少なく，XSS 攻撃を受けていても被害者は気づかない場合が多いと想定される．

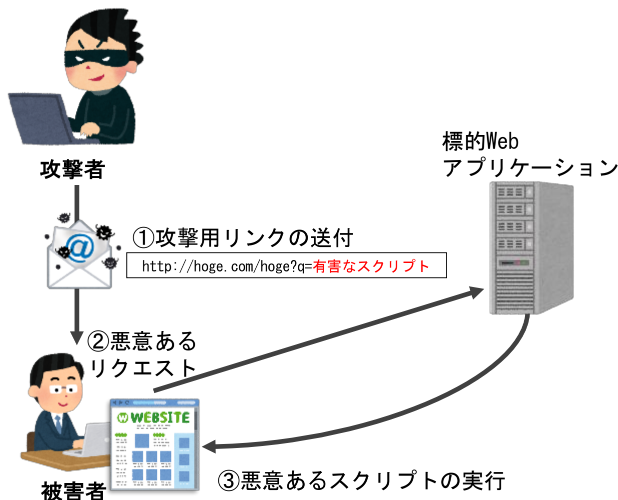

図 4 XSS 攻撃の仕組み

2.3　一般的な対策

XSS 攻撃における有害なスクリプトは，攻撃者がリンク情報の中に記載するパラメータにより Web アプリケーションに注入されてしまう．したがって，パラメータなどのユーザからの入力を適切に無害化することが，現在一般的に利用されている XSS 対策方法である．具体的には，入力値チェックとエスケープ処理によるサニタイジングである．サニタイジング処理は，Web アプリケーション開発時に開発者によってプログラミングされる．

2.3.1. 入力値チェック

XSS 攻撃によってブラウザ側でスクリプトを実行させるには，ブラウザがスクリプトであると認識するコード（例えば，「`<script>有害なスクリプト</script>`」）を含めた形でパラメータに含める必要がある．このようなパラメータに対しては，その中にスクリプトを示す HTML タグ（例: `<script>`）が含まれているかをチェックし，そのパラメータを無効にすることによって有害なパラメータを無毒化するサニタイジング対策が可能である．

2.3.2. エスケープ処理

例えば，ブラウザ上に「`<script>有害なスクリプト</script>`」という文字列を表示させたい場合は，パラメータ中の`<script>`タグを無効化することができない．このような場合には，タグが DOM の要素として認識されないように，エスケープ処理を行うという対策が可能である．Web アプリケーションが「`<script>有害なスクリプト</script>`」というパラメータを受け取った場合は，Web アプリケーション側でエスケープ処理を行い，「`<`」を「`<`」に，「`>`」を「`>`」にそれぞれ変更する．これによって，Web アプリケーションからブラウザへ送信されるレスポンスにおけるパラメータは「<script>有害なスクリプト</script$gt;」となり，ブラウザ側ではこれを（スクリプトではなく）文字列として認識し，ブラウザ上では「<script>有害なスクリプト</script>」と正しく表示される．

2.3.3. 問題点

ブラウザでスクリプトとして認識されるタグやイベントは「`<script>`」以外にも無数にあり，単純なエスケープ処理をしていてもそれを回避できるパターンが複数存在する．よって，現在の XSS 対策は対策漏れが往々にして発生すると考えられる．大手 IT 企業の Web サービスにおいても同様の脆弱性が発見されていることを考えると，多様化した悪意ある入力を完全に無害化することは，開発者の能力にかかわらず，困難な作業であることが予想される．そのため，現在の一般的な対策方法であるサニタイジングは十分かつ容易な対策となっていない可能性がある．

3　ホワイトリスト型攻撃検知の提案

3.1　提案概要

本稿では，サニタイジング等の失敗により XSS 攻撃が発生するという仮定の下で，発生した XSS 攻撃を検知し，被害を防ぐための方法を検討する．ここでは，注入された有害なスクリプトが Web アプリケーション作成者にとって意図せぬ動作を引き起こすことに着目し，

Web アプリケーションの仕様書に書かれた動作をホワイトリストとして定義することで XSS 攻撃の検知と防御を行う手法を提案する．ホワイトリスト型の攻撃検知を実効的なものとするには，ホワイトリストの定義やその作成方法を明確化し，ホワイトリストを自動的に生成可能とすることが肝要である．提案手法においては，Web アプリケーションのテスト工程の中でホワイトリストを自動生成させることが可能である．Web アプリケーションの開発工程を変更することなく，仕様書と逸脱することのないホワイトリストを，それぞれの Web アプリケーションのスクリプトごとに作成できることが提案手法の最大の特長である．

3.2　ホワイトリストの定義

XSS 攻撃の最大の原因は，Web アプリケーション開発時には想定していなかったスクリプトが動作してしまう点にある．すなわち，仕様から外れた動作を実行できてしまう点が問題の本質である．Web アプリケーションに限らず，ソフトウェアが仕様以外の動作をすることは不条理であり，仕様書通りの動作のみを許可するべきである．これは，仕様書に書かれた動作をホワイトリストとして定義することで実現可能である．

例として，ユーザから 3 パターンのパラメータを受け付けて，それに応じた 3 種類の Web サービスをユーザに提供する Web アプリケーションを考えよう．Web アプリケーションが 3 パターンのパラメータに応じて 3 種類の Web サービスを実行することは設計の初期段階で策定され，その内容に基づいて仕様書が作成される．仕様書に記載されているこの 3 パターンのパラメータを Web アプリケーションに入力することによって生成されるレスポンス（クライアントに送信される HTML ファイル）を「正常動作時のレスポンス」とすると，ホワイトリストは正常動作時のレスポンスの集合という形で定義される．

XSS 攻撃が動的なプログラム部分を悪用するものであることに鑑み，提案手法では，正常動作時のレスポンスの中のスクリプト部分のみを抜き出すことでホワイトリストを構成する．具体的なページソースに対して作成されたホワイトリストの例を図 5 に示す．このホワイトリストを用い，Web アプリケーション読み込み時にスクリプトの構造を検証することで，XSS 攻撃を検知する．

適切なパラメータによる出力

```
<html>
<head>--省略--</head>
<form id="form" action="hoge.php" method="GET">
--省略--
</form>
--省略--
<script>document.write(new Data().getFullYear())</script>
2017
</html>
```

↓ スクリプト部分のみ抽出

ホワイトリスト

```
document.write(new Data().getFullYear())
```

図 5 ホワイトリストの作成例

3.3　検知手法

Web アプリケーションは，ユーザから受け取るパラメータにより表示内容が動的に変化する．すなわち，パラメータに応じて，そのレスポンスである HTML ファイルの構造が変化する．しかし，提案手法においては，仕様書に記載されている全種類のパラメータに対して，そのそれぞれのパラメータが入力された際のレスポンス（クライアントに送信される HTML ファイル）が，ホワイトリスト（正常動作時のレスポンス）としてすべて登録されている．このため，レスポンスの HTML ファイルの構造が，ホワイトリストに登録されている状態から外れるようなことは起こり得ない（起こってはいけない）．

これに対し，Web アプリケーション開発者が想定していなかったパラメータが攻撃者によって入力された場合には，その際のレスポンスの HTML ファイルの構造が変化することになる（典型的には，スクリプトを含んだパラメータが入力されることによって，レスポンスの中に新たなスクリプトが挿入される）．この結果，HTML ファイルの構造がホワイトリストに登録されていない構造へと変化する．この特徴に着目し，XSS 攻撃を検知する．

具体的には，3.2 節で定義したホワイトリストと現在ユーザが表示しているページのスクリプト部分をブラウザで比較することによって，XSS 攻撃に対するホワイトリスト型検知を実現する．ブラウザは，ホワイトリストに登録されているレスポンス（HTML ファイル）の構造と実際のレスポンス（HTML ファイル）の構造を常に比較し，両者の構造が合致している場合は XSS 攻撃の可能性はないと判断する．両者の構造が合致しない場合は XSS 攻撃を受けている可能性がある（図 6）と判断する．

スクリプトを含むパラメータによる出力

```
<html>
<head>--省略--</head>
<form id="form" action="hoge.php" method="GET">
--省略--
</form>
--省略–
<script>alert("攻撃可能！")</script>
<script>document.write(new Data().getFullYear())</script>
2017
</html>
```

スクリプト部分のみ抽出

```
alert("攻撃可能！")
document.write(new Data().getFullYear())
```

ホワイトリストと比較：不一致なので攻撃の可能性！

ホワイトリスト

```
document.write(new Data().getFullYear())
```

図 6 XSS 攻撃の可能性がある場合の例

3.4 実現方法

本検知手法では，実行されるスクリプトはユーザのブラウザ上で検証される．つまり，この手法を実現する条件として，作成したホワイトリストをユーザが利用している端末上に存在し，その端末上のブラウザから利用できる状態である必要がある．これは，ホワイトリストをあらかじめWebサーバに保存しておき，ブラウザがWebアプリケーションを要求し，アプリケーションのデータをサーバから読み込む際，同時にホワイトリストもダウンロードすることで実現可能である．ホワイトリストのダウンロードや検証は，ブラウザの拡張機能で実装可能である．その実現方法を図 7 に示す．

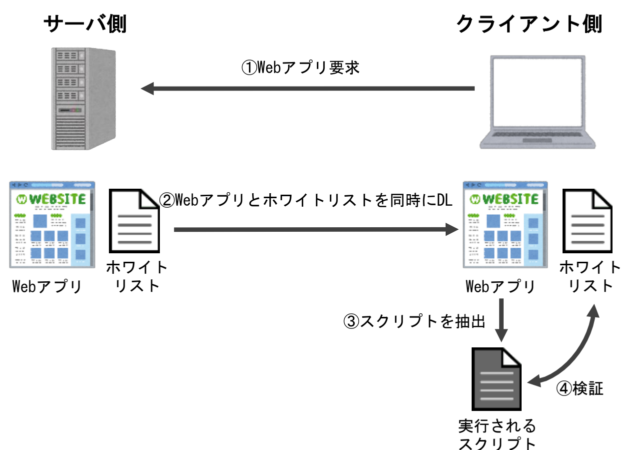

図 7 ホワイトリスト型検知の実現方法

3.5 運用手法の提案

通常，Webアプリケーションには複数のページがあり，そのページごとにユーザ入力などのパラメータを取得している．XSS 攻撃検知のためのスクリプトのホワイトリストは，そのページごとに作成する必要があるため，このようなパラメータを取得して表示するページが多いほど，ホワイトリストを手動で作成するのは困難になる．

この結果，ホワイトリストの作成は必然的に自動化を行うことが求められる．本稿では，アプリケーションをリリースする際，多くの開発企業がパラメータテストを含め，アプリケーションが仕様書通りに動作しているのかの動作テストを行うことに着目し，開発側のテスト環境において自動的に動作テストの工程でスクリプトのホワイトリストを生成し，そのホワイトリストを XSS 攻撃検知に利用するという運用手法を提案する．

この運用手法を採用することで，本来の開発工程を変更することなくスクリプトのホワイトリストを作成することが可能であり，効率的かつ理にかなった運用手法であると考えられる．本提案手法を採用した工程と通常の工程の比較を図 8 に示す．

通常の工程

提案手法による工程

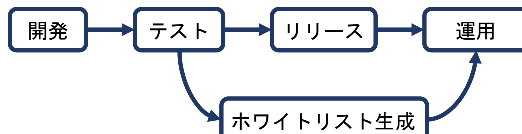

図 8 提案手法の有無による工程の比較

4 運用手法の実装による評価

実際に Web アプリケーションのテストツールを用いて Web アプリケーションが実行可能なスクリプトのホワイトリストを自動的に生成することで，本運用手法の有効性を評価する．

4.1 対象とする Web アプリケーション

テスト対象とする Web アプリケーションとして，申し込みフォームを模したページを作成した（図 9）．このWebアプリケーションはJavaScriptによって出力される申し込み年月日と見出し，3つのテキストボックス，ボタンの5つの要素から成っている．申し込みフォームのテキストボックスに文字列を入力し，確認ボタンを押すと申し込み確認ページに遷移し，入力した値を確認することができる．

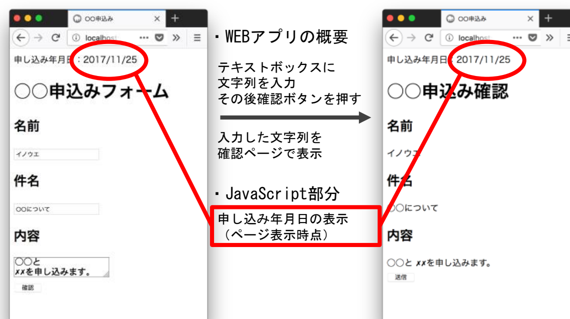

図 9 対象とする Web アプリケーション

4.2 Selenium を用いたテスト

Web アプリケーションのテストに用いられる Selenium[5]というテストツールを用いて評価を行う．このテストツールは，テストの自動化で広く利用されており，キーボードからの入力や，表示の比較など，テストをコードとして記述することが可能である．また，テスト中に得られるデータを別の処理へ利用することも容易である．

4.3 提案手法の有効性評価の手法

この Web アプリケーションでは，申込みフォームのテキストボックスに文字列を入力して確認ボタンを押すと，正常に申し込み確認ページに遷移することをテストする．したがってテストケースは，確認ボタンを押すことによりページが遷移し，その遷移先のページにおいて，見出しに「〇〇申込み確認」と表示されているか文字列の比較を行い，同様の文字列であればテスト結果は「成功」，そうでなければ「失敗」とする．

テストツールを通してテストケースの検証を行っていく．テストの流れは以下の通りである．テスト対象の画面の状態を図 10 に示す．

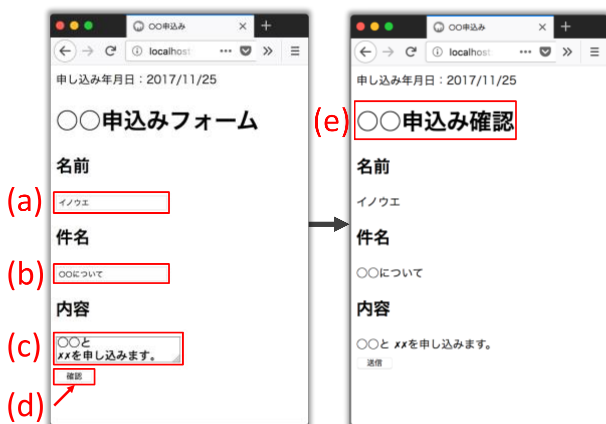

図 10 テスト対象の画面状態

(1) ブラウザを起動
(2) ページを表示
(3) 図 10 (a) のテキストボックスに文字列を入力
(4) 図 10 (b) のテキストボックスに文字列を入力
(5) 図 10 (c) のテキストボックスに文字列を入力
(6) 図 10 (d) のボタンを押す (ページ遷移が発生)
(7) 図 10 (e) でテストケースを検証

プログラミング言語によるテストケースは，この流れに沿って記述していく．

Python 言語による記述例を図 11 に示す．このプログラムでは，テストの結果が成功であった場合は何も返さず，テスト結果が失敗であった場合は「AssertionError」を返す．

```
browser = webdriver.Firefox()
browser.get('http://localhost:8888/XSS/test.php')
browser.find_element_by_name('name').send_keys(”name123")
browser.find_element_by_name('title').send_keys("title123")
browser.find_element_by_name('content').send_keys("con123")
browser.find_element_by_id('submitBtn').click()
assert "〇〇申込み確認".decode('utf-8') in browser.page_source
```

図 11 Python 言語でのテストケース記述例

4.4 ホワイトリストの自動生成と運用

ホワイトリストを生成するには，仕様書に書かれた動作だけが記述された JavaScript 部分が必要である．この JavaScript は，Web アプリケーションの正常動作時に出力されるページソースから JavaScript 部分だけを抽出することで得ることができる．

Selenium では，テストの流れにあるどの状態でも，その時点での JavaScript を含めたページソースを取得することができる．よって，テスト実行時に状態(7)以降，すなわちテストケースが正しく検証された後のページソースから JavaScript 部分を自動的に抽出・保存することでホワイトリストを自動で作成することが可能である．

自動生成したホワイトリストを Web サーバに保存し，提案手法の運用において利用する．Web アプリケーションに変更が生じた場合も生成時と同様にテストツールにてテストを行い，ホワイトリストを再生成することで，容易に最新の状態に保つことが可能である．このような一連した運用が，ユーザを保護することを可能とする．

4.5 結果

本提案手法が Web アプリケーション開発者に要求することは，テスト時に取得できる情報をホワイトリスト作成用のプログラムに渡すことのみである．したがって，少なくとも Selenium を利用するテストにおいて，ホワイトリスト作成を自動化することは容易であるといえる．

また，開発における本来のテスト工程を変更する必要はない．さらに，複数のページをテストする必要のある，より複雑な Web アプリケーションにおいても，テストの大幅な流れは変わらないため，テストケースを検証し「成功」，「失敗」を見てホワイトリスト生成用のプログラムにページソースを入力するようにテストを記述すればよい．

よって，本稿で提案するテスト工程を利用したホワイトリスト作成の自動化は開発者にとって負担の少なく，可用性と有用性の面から優れた手法であるといえる．

5 関連研究

サニタイジング以外において悪性なスクリプトの実行を防止する XSS 攻撃対策について，[6][7][8]の研究が

行われている．

Trevor は，開発者がアプリケーションにセキュリティポリシーを記述することによってクライアントのブラウザ上で JavaScript のコードを書き換え，セキュリティ上重要な API をブラウザからフックすることにより，セキュリティポリシーに記述された API へのアクセスを限定する手法である Browser-Enforced Embedded Policies(BEEP)を提案している[6].

Meyerovich らは，BEEP[4]と非常に類似しているが，セキュリティ上重要な API に対してアスペクト指向プログラミング（Aspect Oriented Programming：AOP）を用いてクライアントのブラウザ上で API をフックすることにより，ホワイトリスト型のセキュリティポリシーをアプリケーションに適用する手法である ConScript を提案している[7].

これらの手法は，開発者によるセキュリティポリシーの設定する必要がある．しかしポリシーを書くことは難しくエラーになりやすい．これらのエラーは，他の脆弱性を引き起こすことや，攻撃者がポリシーを回避する可能性があるため，開発者には相応の知識が要求される．

Sid らは，サーバの HTTP レスポンスヘッダにセキュリティポリシーを付加することでブラウザにおけるアプリケーションの動作を制限する手法である Content Security Policy（CSP）を提案している[8].

この手法は実際に商用的に実装されているが，Web サーバへのセキュリティポリシーの設定が必要であり，またブラウザ毎に実装の対応状況が異なるため想定通りに動作をしない可能性がある．

BEEP[6]・ConScript[7]・CSP[8]は，Web アプリケーションの動作は開発者が熟知していると仮定し，ポリシーの記述は開発者が行う必要がある．そのため，セキュリティに関する責任は開発者の手に委ねられている．

ホワイトリストを自動生成する手法について角田らの研究[9] が行われている．

角田らは，マルウェア感染検知のため，ネットワークのアクセスログを分析し，Web サイトの悪性度を算出することで，ホワイトリスト，ブラックリスト，グレーリストに振り分け，その後マルウェアなどのプログラムでは突破が困難となるような形式で追加認証を行い，グレーリストをホワイトリストかブラックリストに振り分けることで，ホワイトリストとブラックリストの自動的な拡充方法を提案している[9].

6　おわりに

本稿では，Web アプリケーションのスクリプトに対して，「仕様書に示されたとおりの動作」をホワイトリストとして定義することで，XSS 攻撃を検知する手法の提案を行った．

本手法により，Web アプリケーションの開発工程の中で必ず行われるテストで，スクリプトのホワイトリストをテストケースから自動生成することが可能となり，また，従来の Web アプリケーションの開発工程を変更することなく，仕様書と逸脱することのないホワイトリストを，それぞれの Web アプリケーションのスクリプトごとに作成することが可能となる．提案手法をテストツールである「Selenium」を用いて実装・評価し，提案手法の有効性を示した．

今回は，主にテストケースによるホワイトリストの自動生成および運用について取り組んだ．今後は，検知方法の有効性評価も含め，本手法を検証および改良していく．

参考文献

[1] jQuery, < https://jquery.com >, 2017.12.05 閲覧

[2] 日本語 WordPress, < https://ja.wordpress.org >, 2017.12.05 閲覧

[3] 安全な Web サイトの作り方 改訂第 7 版 - IPA 独立行政法人 情報処理推進機構, < https://www.ipa.go.jp/files/000017316.pdf >, 2017.12.18 閲覧

[4] 「マウスオーバーの」問題についての全容 – Twitter 社 日本語公式ブログ, < https://blog.twitter.com/official/ja_jp/a/ja/2010-26.html >, 2017.12.09 閲覧

[5] Selenium - Web Browser Automation, < http://www.seleniumhq.org >, 2017.12.05 閲覧

[6] Trevor Jim, “Defeating script injection attacks with browser-enforced embedded policies”, Proceedings of the 16th international conference on World Wide Web. ACM, pp601-610, 2007.

[7] Leo Meyerovich, Benjamin Livshits, “Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser”, Security and Privacy (SP), 2010 IEEE Symposium on. IEEE, pp481-496, 2010.

[8] Sid Stamm, Brandon Sterne, Gervase Markham, “Reining in the web with content security policy”, Proceedings of the 19th international conference on World Wide Web. ACM, pp921–930, 2010.

[9] 角田朊, 大鳥朊哉, 藤井康広, 谷口信彦, 木城武康, "グレーリストを用いたホワイトリスト/ブラックリストの自動生成によるマルウェア感染検知方法の検討", CSEC66, pp1-7, 2014.