Computer Security Symposium 2018
22 - 25 October 2018

XSS 攻撃検知のためのテストベースホワイトリスト自動生成手法に関する検討

井上 佳祐[†1]　本多 俊貴[†1]　向山 浩平[†1]
大木 哲史[†1]　西垣 正勝[†1]

概要：XSS 攻撃において，多様な悪意のある入力を完全に無害化するのは容易ではない．このような状況では，ホワイトリストベースの XSS 対策が効果的で堅牢なアプローチであると考えられる．しかし，現在の Web アプリケーションは動作が複雑なため，理論的に必要十分なホワイトリストを生成することは大変困難である．この問題に取り組むため，我々は，理論ベースのアプローチではなく，ホワイトリスト作成のためのテストベースのアプローチを提案する．本手法では，開発プロセスの最終段階で行われるソフトウェアテストに焦点を当て，各 Web アプリケーションの仕様に合致するホワイトリストを自動生成する方法を確立する．Web アプリケーションテストツールにホワイトリスト生成のための機能を統合することで，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ホワイトリストの自動生成が可能となる．本提案手法を実装し，有効性を評価する．

キーワード：XSS，ホワイトリスト，自動生成，テストケース，攻撃検知

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のみの 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移行している．現在，Web サイトの大部分は，jQuery [1] をはじめとした JavaScript ライブラリや WordPress [2] などのコンテンツ管理システム（CMS）を使用し，クライアント側とサーバ側の両方のプログラムを利用して動的に生成されている．その結果，従来のアプリケーション（Web アプリケーションではない）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を招いた．

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．攻撃者は，この脆弱性を悪用することで，ユーザのブラウザ上で任意のスクリプトを実行するこが可能となる．このような攻撃の結果，攻撃者にセッション 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すことや，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うきっかけとして利用される可能性がある．SQL インジェクションなどの他の Web アプリケーションの脆弱性は，サービス提供者側の Web サーバに影響を与えるのに対し，XSS 攻撃はユーザの PC に直接影響を与える．したがって，XSS 攻撃への効果的な追加対策を行うことが急務となっている．

この脆弱性が発生する主な原因は，Web アプリケーションの開発時に実装すべき XSS 対策が不完全なことにある．現在，サニタイジングによって入力値の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コストを費やしている大手 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，困難な作業で有ることが推測される．また，今まで知られていなかった（ゼロデイ）脆弱性や攻撃手法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という既存の対策は，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．

このような種類の攻撃に対しては，ホワイトリストに基づく対策を採用し，事前に定義された機能のみを使用可能とする対策が有効である．しかし，機能制限に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いないことが，ホワイトリストベースの XSS 攻撃対策の効果を限定的なものにしてしまっている．ポリシーベースの対策が典型的な例である．文献，[5][6][7]では，スクリプト操作を制限して XSS 攻撃を無効化するポリシーベースの方法が提案されている．しかし，開発者はスクリプトの動作を定義するためのポリシーを決定する必要があるが，必要かつ十分なポリシーを決定する方法論が確立していない．

この問題を解決するために，本論文では，ホワイトリストの作成戦略を理論ベースのアプローチからテストベースのアプローチへと変更することを提案する．また本稿では，テストベースのホワイトリストを作成するために，Web アプリケーション開発の最終段階として実施されるソフトウェアの結合テストに焦点を当てる．具体的にはテスト工程で確認された動作をホワイトリストとして定義し，Web アプリケーションの使用時にホワイトリストに含まれるスク

†1 静岡大学
Shizuoka University

リプトの実行のみを許可することにより，XSS 攻撃を検知し防御する．この方法は，(1) スクリプトの実行がテスト工程で事前に確認されたパターンに対してのみ許可されている，(2) テスト工程によって作成されたホワイトリストは，各 Web アプリケーションの仕様に基づいてソフトウェアテストが行われているため，仕様から構成される（仕様と一致する），(3) テスト工程を通して，ホワイトリストを自動的に生成することが可能である．我々は，Web アプリケーションの自動化テストツールとして広く採用されている Selenium [8] を用いてホワイトリストを自動生成する機能を実装し，提案の有効性を評価する．

2. XSS 攻撃とそのメカニズム

クロスサイトスクリプティング（Cross Site Scripting : XSS）の脆弱性には，3 つの種類が存在する．XSS 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．本章では，Reflected XSS と Stored XSS を例に挙げ，脆弱性と攻撃メカニズムについて説明する．

2.1 XSS 脆弱性の種類

2.1.1 Reflected XSS

Reflected XSS は，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でパラメータによって生成される Web ページを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のプログラムがパラメータとして無害化せずに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．

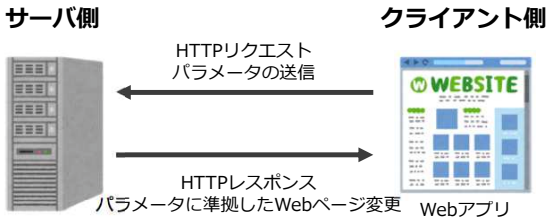

図 1 Reflected XSS の概要

2.1.2 Stored XSS

Stored XSS は，ユーザからの入力を受け取り，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，サーバ側のプログラムで DB に保存された入力値をパラメータとして生成される Web ページをクライアントに返すタイプの 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悪意ある入力値はサーバの DB に保存されており，Web アプリケーションは，その保存された値をアプリケーションが実行される毎に読み込み続けるため，攻撃の持続性がある．

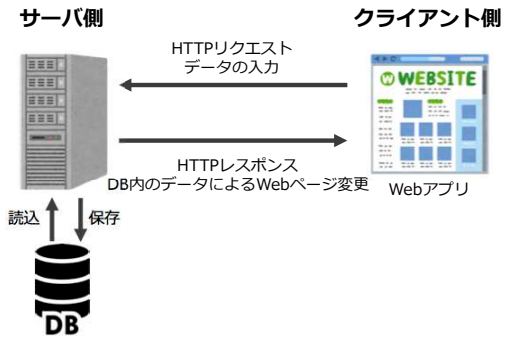

図 2 Stored XSS の概要

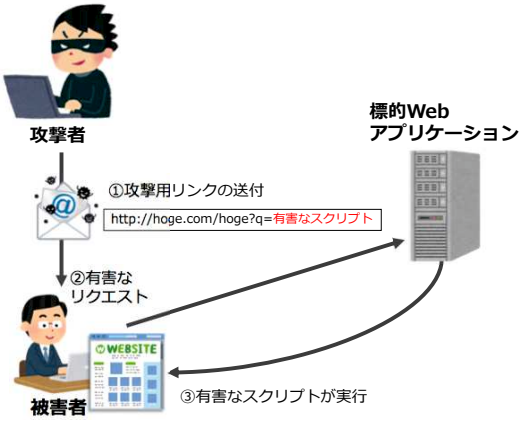

図 3 Reflected XSS の仕組み

2.2 XSS 攻撃のメカニズム

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．

2.2.1 Reflected XSS のメカニズム

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れるためのリンク（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 3 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 3 ②）．標的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してレスポンスする．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を実行してしまう（図 3 ③）．

2.2.2 Stored XSS のメカニズム

攻撃者は，標的 Web アプリケーションに対して有害なスクリプトを埋め込む，例えば掲示板サイトなどであった場合は，有害なスクリプトを書き込む（図 4 ①）．書き込まれた内容は DB に保存される（図 4 ②）．被害者は標的 Web アプリケーションをよく利用するユーザであるため，標的 Web アプリケーションにアクセスしてしまう（図 4 ③）．標的 Web アプリケーションは，書き込まれている内容を DB から読み込む（図 4 ④）．標的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

ウザに対してレスポンスする．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を実行してしまう（図 4 ⑤）．

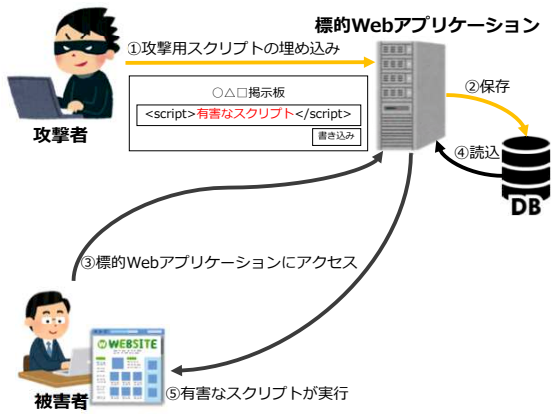

図 4 Stored XSS の仕組み

2.2.3 攻撃成功後の被害

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認証情報を盗む恐れ，また攻撃者が用意した有害な 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はバックグラウンドで行われ，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．

3. XSS 攻撃への対策

3.1 サニタイジング

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や，DB に格納された悪性な入力値を読み込むことにより 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値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実装される．

3.1.1 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．

3.1.2 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．

3.1.3 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バ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．また，今まで知られていなかった（ゼロデイ）脆弱性や攻撃手法が新たに発見される場合もある．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．

3.2 ホワイトリスト

ホワイトリストを採用した対策では，ホワイトリストになにがホワイトとなるのかを定義することができれば，定義したもの以外を否定するだけなので，安全性の面からみるととても頑丈な対策と言える．Web アプリケーションの動作は普遍的なものではないため，このようなアプリケーションへの攻撃に対しては，ホワイトリストに基づく対策を採用し，事前定義された機能のみを有効にすることが効果的である．実際，BEEP [5], ConScript [6],CSP [7]など，セキュリティポリシーで設定された以外のスクリプトの動作を禁止し XSS を防止する手法がある．これはホワイトリストを利用した対策に手法に限りなく類似した手法であり実質ホワイトリストベースの XSS 対策と言える．ホワイトリストベースの対策では，ホワイトリストの品質を向上させることが非常に重要である．この点に関して，ホワイトリストをより拡充させるための経験的アプローチ [9] が存在する．

3.2.1 ポリシーベースホワイトリスト

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）[5] を提案している．

Meyerovich らは，BEEP [5] 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 [6] を提案している．

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）[7] を提案している．CSP は，実際に商用的に実装されており，すでにいくつかのブラウザとサーバで利用可能である．しかし，広く普及しているわけではない．その理由として 1 つは，ブラウザやサーバ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ためである．

3.2.2 経験ベースホワイトリスト

厳密には XSS 攻撃対策ではないが，角田らは，マルウェア感染検知のためのホワイトリストおよびブラックリストを自動的に拡充する手法 [9] を提案している．これは，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，Web サイトをホワイトリスト，ブラックリスト，グレーリストに分類する手法である．グレーリストに分類された Web サイトにおいては，それが良性なのか悪性なのかを再判別するため，マルウェア（自動化プログラム）では突破が困難となるような形式で追加の認証テストが行われる．

3.2.3 問題点

ポリシーベースの手法では，ホワイトリストとなるセキュリティポリシーは開発者自身が決定する必要がある．つまり，開発者は Web アプリケーションのセキュリティについての妥当な知識が必要である．

また，それだけでなく Web アプリケーションの肥大化も問題になる．現在の Web アプリケーションの構造と動作は非常に複雑であるため，開発者でさえアプリケーションの動作を正確に把握することは容易ではない．

したがって，ポリシーを設定することは困難で，不十分な設定が原因でエラーが発生しやすくなる．このようなエラーは，他の脆弱性を引き起こし，攻撃者がセキュリティポリシーをバイパスして不正な動作を行わせる可能性がある．言い換えれば，必要十分なセキュリティポリシーを作成する方法論はまだ確立されておらず，このようなポリシーベースの XSS 対策の効果は非常に限定されていると言える．

経験ベースの方法は，ホワイトリストをある程度拡充するのに有用ではあるが，このような経験的アプローチでは理論的根拠がまだ不足している．この種の不完全さは，既存のすべてのホワイトリストベースのアプローチが直面している大きな問題である．

4. テストベースホワイトリスト

4.1 提案概要

本論文では，前章で述べたホワイトリストベース XSS 攻撃対策に存在する問題を解決するため，ホワイトリストの作成戦略を理論ベースのアプローチからテストベースのアプローチへと変更することを提案する．本稿では，テストベースのホワイトリストを作成するため，Web アプリケーションの開発工程の最終段階として実施されるソフトウェアの結合テストに焦点を当てる．テストベースのホワイトリストは，テスト工程で事前に確認されたスクリプトのみを実行することを許可する．具体的には，テスト工程で検証された動作をホワイトリストとして定義し，Web アプリケーションの使用時にホワイトリストに含まれるスクリプトの実行のみを許可することで XSS 攻撃を検知して防止する．ソフトウェアテストは，各 Web アプリケーションの仕様書に基づいて行われる．したがって，テスト工程にホワイトリスト生成プロセスを統合することで，Web アプリケーションの開発プロセスを変更することなく，各 Web アプリケーションのスクリプトごとに，仕様を逸脱することのないテストベースのホワイトリストを自動生成することが可能となる．

まとめると，この手法には以下のような利点がある．

(1) スクリプトの実行がテスト工程で事前に確認されたパターンに対してのみ許可されている．
(2) テスト工程によって作成されたホワイトリストは，各 Web アプリケーションの仕様に基づいてソフトウェアテストが行われているため，仕様から構成される（仕様と一致する）．
(3) テスト工程を通して，ホワイトリストを自動的に生成することが可能である．

4.2 テストと仕様書の連携

テストベースホワイトリストの重要なコンセプトは，事前にホワイトリストとして確認された動作のみを定義することである．これは理論的に必要かつ十分なホワイトリストではないが，テストと仕様の連携によってヒューリスティックに十分なホワイトリストを作成することができる．

XSS 攻撃の原因は，Web アプリケーション開発者が意図していなかったスクリプトが注入されることである．つまり，問題の本質は，意図しないスクリプトが実行されてしまうことにある．このような意図しないスクリプトの実行を防ぐためには，ソフトウェア開発の初期段階で作成され

る Web アプリケーションの仕様書を利用することが可能であると考えられる．仕様書には，実装されるべき全ての機能と，各機能の実行時にアプリケーションがどのように動作するかが示されてある．すなわち，仕様書というのは，「意図された動作の集合」であると言える．この仕様書は，Web アプリケーション開発の最終段階でも利用されている．開発者は，Web アプリケーションをリリースする前に，Web アプリケーションが要求された仕様を満たしているかどうか，または仕様書で示されている通りの動作をするかどうかをテストする．このように，仕様書に示されているすべての動作がテスト工程で確認されることが期待されるため，テスト工程を通じて，意図されたすべての動作を含むテストベースのホワイトリストを生成することが可能である．

ただし，このようなテストにおいて 100%のカバレッジを達成することは簡単な作業ではない．これを軽減するために，HTML ページソースコードのスクリプト構造に焦点を当てる．前述のように，XSS 攻撃の原因は，HTML ページソースコードへの悪意あるスクリプトの注入である．したがって，XSS 攻撃は，HTML ページソースコードのスクリプト構造のみを比較することによって検知することが可能である．HTML のスクリプト構造は，典型的な入力値をテストするだけで収集できるため，ホワイトリストを作成するために網羅的なテストは必要でないと考えられる．さらに，これによりホワイトリストのデータサイズが縮小され，比較が容易になる．そのため本稿においては， HTML ページソースコードからスクリプト部分（Script タグ内の JavaScript および W3C で定義されているマウスイベント内の JavaScript）のみを抽出し，テスト工程を通して，ホワイトリストに登録している．特定の HTML ページソースコードに対して生成されたホワイトリストの例を図 5 に示す．

4.3 検知手法

動的な Web アプリケーションでは，表示されるコンテンツはユーザからの入力や，データベース中の登録内容に応じて変化する．すなわち，これらのパラメータに従って，ページの HTML ソースコードの構造が変化する．提案手法では，テスト工程において表示・実行された HTML ページソースコードのスクリプト構造がすべてホワイトリストとして登録されている．したがって，ユーザによって入力されたパラメータが Web アプリケーションの仕様内にある限り（通常の意図されたパラメータを入力する限り），Web アプリケーションによって生成される HTML ページソースコードは，ホワイトリストに登録されているスクリプト構造から逸脱することはない．

一方，攻撃者によって Web アプリケーション開発者が想定していないパラメータが入力されると，ホワイトリストに登録されていない HRML ページソースコードが生成される．XSS 脆弱性のある Web アプリケーションにおいて，スクリプトを含むパラメータが入力されると，Web アプリケーションは，意図しないスクリプトが注入された HTML ページソースコードを生成する．その結果，Web アプリケーションによって生成された HTML ページソースコードのスクリプト構造が，ホワイトリストに登録されていない構造に変更される．提案方式では，この特徴を用いて XSS 攻撃を検知する．この特徴は，Reflect XSS だけでなく，Stored XSS や DOM-based XSS などすべての XSS 攻撃で見られるため，同様の手法で攻撃の検知が可能である．

提案手法における XSS 攻撃検知は，クライアントブラウザ上に実装される．ユーザが Web サービスにアクセスしてその Web アプリケーションを利用するたびに，Web アプリケーションによって生成された HTML ページソースコードのスクリプト構造と，ホワイトリストに登録されている HTML ページソースコードのスクリプト構造が常に比較される．両方の構造が一致しない場合，XSS 攻撃が発生していると判断できる．図に XSS 攻撃の例を示す．図 5 の Web ページにおいて「`<script>alert(“Attack!”)</script>`」などの JavaScript を入力することにより，図 6 で示す意図しない HTML ページソースコードが生成される．図 5 および図 6 で示すように，スクリプト構造が異なるため，攻撃を検知することができる．

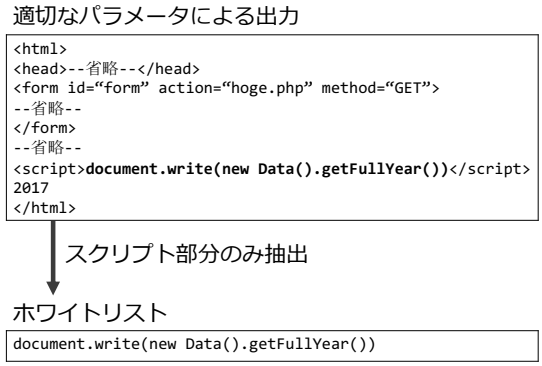

図 5 ホワイトリストの作成例

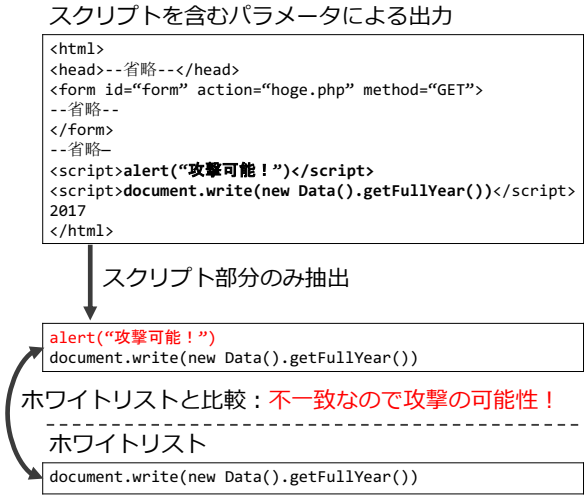

図 6 XSS 攻撃の可能性がある場合の例

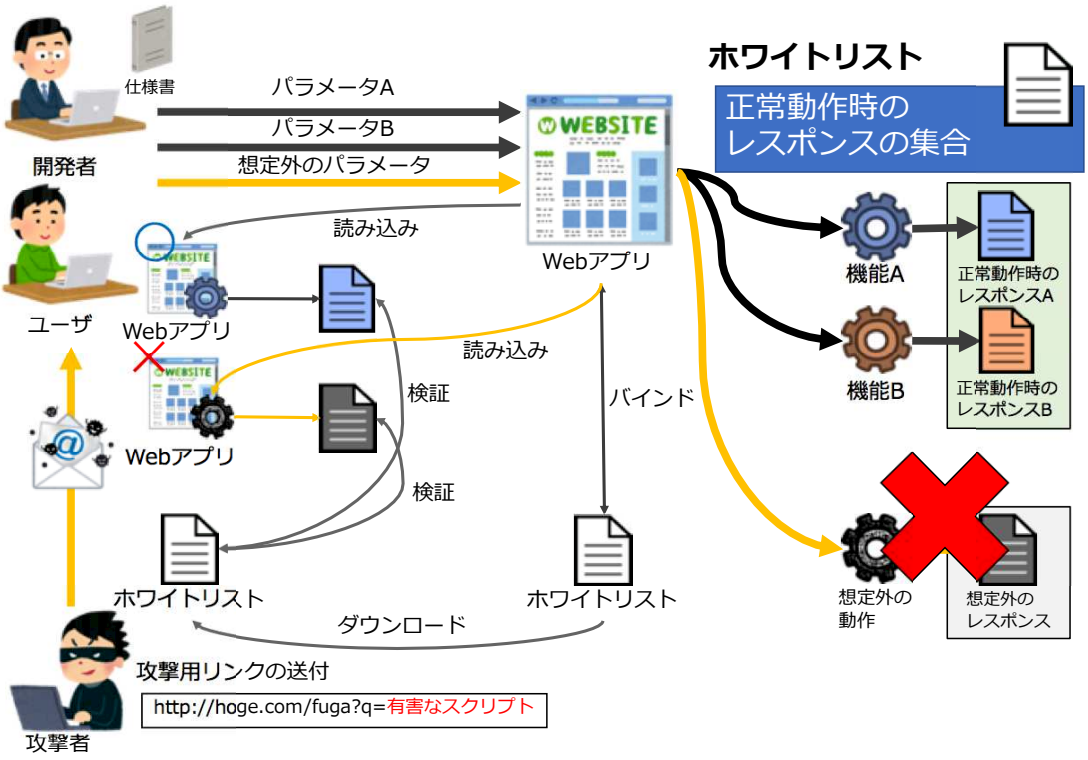

図 7 ホワイトリスト型攻撃検知の展開

4.4 ホワイトリスト自動生成

4.2 節で述べたように，本稿では，前もってホワイトリストとして確認された動作のみを定義し，ヒューリスティックに適切なテストベースのホワイトリストを作成するために仕様書を利用することを提案している．テストベースのホワイトリストは，仕様書に示された意図されたパラメータが入力された時に，Web アプリケーションが生成する HTML ページソースコード内のスクリプト部分で構成されている．これは，提案方式のホワイトリスト生成プロセスが，Web アプリケーションのテスト工程と非常に類似していることを意味する．

テスト工程の目的は，Web アプリケーションが仕様に従って実装されているかどうかを確認することである．Web アプリケーションをリリースする時，開発者は，Web アプリケーションがパラメータテストを含めて，仕様に従って動作しているかどうかを検証する動作テストを行う．このパラメータテストでは，仕様書に示されているいくつかのパラメータパターンを入力し，Web アプリケーションが正しく応答するかどうかを検証する．つまり，ホワイトリスト生成に必要なすべての HTML ページソースコードは，Web アプリケーションのテスト工程において取得することができる．

我々が着目したように，提案方式は，Web アプリケーションのテスト工程においてホワイトリスト生成プロセスを統合することが可能である．より具体的には，テスト工程において生成された各 HTML ページソースコードからすべてのスクリプト部分を抽出することにより，提案方式におけるテストベースホワイトリストを生成することができる．この点に着目し，Web アプリケーションのテストツールを改良して自動ホワイトリスト生成機能を実装することを提案する．この機能を採用することで，通常の Web アプリケーション開発工程で Web アプリケーションごとにテストベースのホワイトリストを生成することが可能となる．これにより，ホワイトリストの XSS 攻撃検知をより効果的かつ合理的に展開することができる．

4.5 デプロイ

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や，DB に格納された悪性な入力値を読み込むことにより 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値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実装される．

4.3 節で述べたように，本検知方法では，実行されるスクリプトの構造はユーザ側のクライアントブラウザで検証される．したがって，提案手法を実用化するためには，生成されたホワイトリストをユーザ側に送信し，クライアントブラウザから利用できるようにする必要がある．これは，対応する Web コンテンツの HTML ページソースコード中にホワイトリストへのハイパーリンクを埋め込むことであ

る．これにより，Web アプリケーションをサーバから受信すると同時にホワイトリストがダウンロードされる．ホワイトリストのダウンロードと検証は，ブラウザの拡張機能によって実装することが可能である．本提案手法の実現全体像を図 7 に示す．

5. 提案手法の実装と評価

実際に，Web アプリケーションのテストツールを用いて，ホワイトリストを自動生成するツールを実装し，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生成することで，本運用手法の有効性を評価する．また，本章の実験結果からホワイトリストのスクリプト構造に対しての考察を行う．

5.1 対象 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，実際の開発で使われた仕様書から，開発対象の Web アプリケーションの特徴的な機能のみを選択し，データベースシステムを模したページ（図 8）を作成した．この Web アプリケーションは，データベースの登録，閲覧を Web ブラウザから行えるようにしたインターフェースである．ページ構成として，「登録フォームページ（図 8 ページ①）」，「登録完了ページ（図 8 ページ②）」，「登録リストページ（図 8 ページ③）」の 3 ページから成っている．また，各ページには図 8 の赤枠および赤文字の「(JS)」で示す場所に，JavaScript を利用している．具体的には，ページ①では，登録ボタンのクリックでフォーム内容を送信し，ページ②へ遷移させるために，ページ②では，登録日に登録した日付を表示させるために，ページ③では，登録されている内容に NEW マークを表示するかどうかを判定し表示するために利用している．また，この Web アプリケーションでは，ページ②で Reflected XSS が，ページ③で Stored XSS の脆弱性が存在し得る．仕様書は，以下の通りである．

（仕様1） ページ①にいてテキストボックスに文字列を入力し，「登録」ボタンを押すと，ページ②へのページ遷移が発生する．

（仕様2） ページ②において，ページの見出しに「登録完了」と表示され，入力して登録された内容が表示される．

（仕様3） ページ②において，「登録リスト一覧」リンクをクリックすると，ページ③へのページ遷移が発生する．

（仕様4） ページ③において，（仕様 2）で登録した内容を含め，データベースに登録されている内容がすべて表示される．

5.2 Selenium を用いたテスト

本研究では，Selenium [8] と呼ばれる Web アプリケーションテストツールを用いて提案方式を実装した．このテストツールは，テストプロセスの自動化に広く利用されている．Selenium では，テストケースとして「キーボードの入力を受け取り→Web ページを生成→生成されたページを教師データと比較」というテスト手順が自動実行プログラムコードで記述されている．また，テスト中に得られるデータを他の処理へ利用することも容易である．

5.3 対象 Web アプリケーションのテスト工程

テスト工程の目的は，Web アプリケーションが仕様に従って実装されているかを確認することである．したがって，この対象 Web アプリケーションのテストプロセスは，5.1 節で示した仕様 1〜4 の機能をテストし，ページごとのテストに合格すると，そのページでのテスト結果は「成功」，それ以外は「失敗」となるように構成される．テスト中に画面の状態を図 9 に示す．

Selenium では，一連のテスト手順としてテストケースを記述する必要がある．この対象 Web アプリケーションの場合のテスト手順は，以下の通りである．

(1) ブラウザを起動
(2) ページ①を表示
(3) 図 9 (a)でページ①が表示されたかを検証
(4) 図 9 (b)，(c)のテキストボックスに文字列を入力
(5) 図 9 (d)のボタンを押下（ページ②に遷移）
(6) 図 9 (e)でページ②が表示され，図 9 (f)，(g)で入力した文字列が表示され，登録が完了しているか検証
(7) 図 9 (h)のリンクを押下（ページ③に遷移）
(8) 図 9 (i)でページ③が表示され，図 9 (j)で登録したデータが表示されているか検証
(9) テストケース終了

図 8 対象とする Web アプリケーション

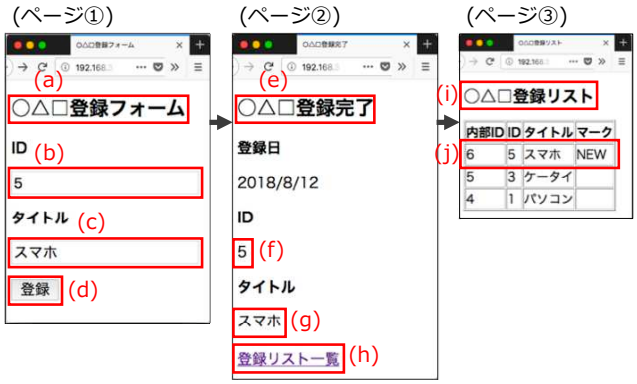

図 9 画面の状態

5.4 ホワイトリストの自動生成

4.4 節で説明したように,テストベースのホワイトリストは，Web アプリケーション開発の最終段階として行われるテスト工程を通して生成可能である．Selenium のプログラムをわずかに変更することで，テスト工程中に生成された各 HTML ページソースコードからすべての JavaScript 部分を抽出し，テストベースのホワイトリストの自動生成機能を実装することができる．Selenium は，テストプロセスのどの段階でも，その時点の JavaScript を含む HTML ページソースコードを取得できる。 したがって，テスト手順のステップ(3)，(6)，(8)を実行した後の HTML ページソースコードから自動的に JavaScript 部品を抽出してホワイトリストを生成することができる。5.1 節で説明したとおり，対象とする Web アプリケーションには 3 つのページがあることから，提案手法の適用により 3 ページ分のホワイトリストが生成された．これらのホワイトリストには，同節で説明した JavaScript の機能が含まれている．

5.5 考察

対象の Web アプリケーションのページ①，ページ②については，データベースの内容にかかわらず，入力された値に対してのみページに反映される．しかし，ページ③では，データベースに保存されているデータも表示に反映するため，データを登録するにしたがって，表の行が増えていく．図 8 で登録しているものに加えて，もう 1 レコード追加したときのページ③の画面状況を図 10 に示す．ここで示す表のマーク列では，各行に対して，JavaScript が生成されるため，登録されているレコード数分の JavaScript が生成されている．また，「NEW」を表示するかどうかで JavaScript の一部の表記が異なる．そのため，Web アプリケーションの実装方法によっては，最終段階のテスト工程で JavaScript だけを抽出するだけでは，不十分な可能性も考えられる．これらの課題を解決するために，条件分岐や繰り返しなどを表現できるようなホワイトリストのスクリプト構造を再考する必要がある．

○△□登録リスト

内部ID	ID	タイトル	マーク
7	7	タブレット	NEW
6	5	スマホ	
5	3	ケータイ	
4	1	パソコン	

図 10 レコード追加後のページ③の画面状況

5.6 特徴

少なくとも Selenium をテスト環境として利用している場合，ホワイトリスト自動生成は容易であるといえる．ホワイトリスト生成に必要なすべての情報/データは，Web アプリケーション開発のテスト工程を通じて取得できる．また，Web アプリケーションの開発時に従来のテスト工程を変更する必要はない．以上から，このホワイトペーパーで提案されている Web アプリケーションのテストプロセスによる自動ホワイトリスト生成は，開発者にとって負担の少なく，可用性と有用性の面から効果的で合理的な方法であるといえる．

6. おわりに

本稿では，XSS 攻撃検知のためにホワイトリストを自動的に生成するための,「テストベース」のアプローチによる手法の提案を行った．提案手法では，テスト工程で検証されるスクリプト構造に焦点を当て，ホワイトリストを定義している．完全性の観点から，テストベースのホワイトリストは，テスト工程で事前に確認されたスクリプトのみが実行される．健全性の観点から，各 Web アプリケーションの仕様書に基づいてソフトウェアテストを行うため，仕様書に一致するホワイトリストを自動的に生成することが可能である．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．

今後は，より高効率で検知することができるスクリプト構造やクライアント側でのホワイトリストの具体的な検証機能の検討・有効性評価を含め，本手法を検証および改良していく．

参考文献

[1] “jQuery”. https://jquery.com/, (参照 2017-12-05).

[2] “日本語 - WordPress”. https://ja.wordpress.org/, (参照 2017-12-05).

[3] “安全なウェブサイトの作り方 改訂第 7 版”. https://www.ipa.go.jp/files/000017316.pdf, (参照 2017-12-18).

[4] “「マウスオーバーの」問題についての全容”. https://blog.twitter.com/official/ja_jp/a/ja/2010-26.html, (参照 2017-12-09).

[5] Jim, T.. Defeating script injection attacks with browser-enforced embedded policies. Proceedings of the 16th international conference on World Wide Web. ACM. 2007, p. 601-610.

[6] Meyerovich, M. and Livshits, B.. Conscript: Specifying and enforcing fine-grained security policies for javascript in the browser. Security and Privacy IEEE Symposium on. IEEE. 2010, p. 481-496.

[7] Stamn, S., Sterne, B. and Markham, G.. Reining in the web with content security policy. Proceedings of the 19th international conference on World Wide Web. ACM. 2010, p. 921-930.

[8] “Selenium - Web Browser Automation”. https://www.seleniumhq.org/, (参照 2017-12-05).

[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. IPSJ SIG Technical Reports, 2014-CSEC-66-16. 2014, p. 1-7.