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

Effectiveness of Examination-based Whitelist for DOM Based XSS Attacks in Microservice Architectures

メタデータ	言語: jpn 出版者: 公開日: 2022-04-01 キーワード (Ja): キーワード (En): 作成者: 井坂, 佑介, 天笠, 智哉, 奥村, 紗名, 佐々木, 葵, 野崎, 真之介, 大木, 哲史, 西垣, 正勝 メールアドレス: 所属:
URL	http://hdl.handle.net/10297/00028835

マイクロサービスアーキテクチャにおける DOM Based XSS 攻撃に対するテストベースホワイトリストの有用性

井坂 佑介[1]　天笠 智哉[1]　奥村 紗名[1]　佐々木 葵[1]　野崎 真之介[1]
大木 哲史[1]　西垣 正勝[1]

概要：Web サービスの複雑化に伴い，マイクロサービスアーキテクチャ（MSA）型の Web アプリケーション開発へと移行している．しかし，MSA においては，マイクロサービス同士を組み上げる際に，各マイクロサービス間でセキュリティポリシのコンフリクトが発生し得るため，セキュリティバイデザインの実現に課題を抱えている．この課題に対処する方法としては，セキュアプログラミングガイドラインの運用によってマイクロサービス間のセキュリティポリシを共通化する方法や，API Gateway によってセキュリティポリシの調停を行う方法が挙げられる．しかし，セキュアプログラミングガイドラインの導入は，サービスの開発に注力したい開発者にとって技術的及び作業的負担が大きいという問題がある．また，DOM Based XSS 攻撃のようにフロントエンドで攻撃が完結するものに対しては，API Gateway での調停が機能しないという問題がある．そこで，本稿では，MSA 型の Web アプリケーションの開発において，テストベースホワイトリストを採用することで，開発テストを通じてマイクロサービス間のセキュリティポリシの調停を開発者が意識せずに達成できる仕組みを提案する．

キーワード：マイクロサービスアーキテクチャ，マイクロフロントエンド，DOM Based XSS，テストベースホワイトリスト，セキュリティポリシ

Effectiveness of Examination-based Whitelist for DOM Based XSS Attacks in Microservice Architectures

Yusuke Isaka[1]　Tomoya Amagasa[1]　Sana Okumura[1]　Aoi Sasaki[1]
Shinnosuke Nozaki[1]　Tetsushi Ohki[1]　Masakatsu Nishigaki[1]

Abstract: As the complexity of Web services increases, Web application developments are migrating to microservice architecture (MSA). However, in MSA, the security policies of each microservice may conflict each other when assembling microservices, which poses a challenge in achieving security-by-design. There are two ways to deal with this challenge: one is to standardize security policies among microservices by implementing secure programing guidelines, and the other is to mediate security policies among microservices by using API Gateway. However, the introduction of secure programing guidelines is technically and operationally burdensome for developers who want to focus on service design. In addition, mediation using the API Gateway does not work for attacks that are completed in the front-end, such as DOM Based XSS attacks. Therefore, in this paper, we propose to apply Examination-based Whitelist in the development of MSA-based Web applications, where mediation of conflict in security policies between microservices is achieved through development tests without the developer taking care of it.

Keywords: Microservices Architecture, Micro Front-end, DOM Based XSS, Examination-based Whitelist, Security Policy

1. はじめに

近年，Web アプリケーションの複雑化に伴い，俊敏な設計変更への対応や短期間での実装が Web アプリケーション開発の場で要求されるようになった．これらに対応するため，従来のモノリシックアーキテクチャから，マイクロサービスアーキテクチャ（MSA）型の Web アプリケーション開発への移行が検討されている[1]．将来的には，サードパーティ製のマイクロサービスが流通することが予想され，これらのマイクロサービスを組み上げてサービスを構築するような開発が行われると考えられる[2][3].

しかし，MSA 型の Web アプリケーション開発の場においては，複数のマイクロサービスを 1 つのサービスに統合する際に，マイクロサービス間のセキュリティポリシにコンフリクトが発生し得るため，セキュリティバイデザインの実現に根本的な課題を抱えている．この課題に対処する方法としては，セキュアプログラミングガイドラインの運用によってマイクロサービス間のセキュリティポリシを共通化する方法[4]や，Web アプリケーションのフロントエンドとバックエンドの中間に位置する API Gateway によってセキュリティポリシの調停を行う方法が挙げられる[5]．しかし，セキュリティ要件が複雑で方法論が不足している MSA においては，セキュアプログラミングガイドラインの導入は困難[6]であり，仮に導入が出来たとしても，サービ

1 静岡大学
Shizuoka University

© 2021 Information Processing Society of Japan

スの開発に注力したい開発者にとって技術的及び作業的負担が大きいという問題がある．また，DOM Based XSS 攻撃のようにフロントエンドで攻撃が完結するものに対しては，フロントエンドの後段に位置する API Gateway での調停が機能しないという問題がある[7].

この問題に対し，本稿では，MSA 型の Web アプリケーションの開発において，テストベースホワイトリスト[8]のアプローチを採用することで，マイクロサービス間のセキュリティポリシの調停を開発者が意識せずに達成できる仕組みを提案する．また，提案の典型例として，MSA において未だ有効的な対策が成されていない DOM Based XSS 攻撃に対する対策を行い，提案の有用性の検討をする．

2. 課題設定

2.1 マイクロサービスアーキテクチャ（MSA）

これまでの Web アプリケーションの開発では，単一のアプリケーションとして構築されるモノリシックアーキテクチャが主流であった．しかし，近年，Web アプリケーションの複雑化に伴い，ビジネスや環境の変化に迅速かつ柔軟に対応することが困難になってきている．モノリシックアーキテクチャでは，1 つのモジュールの修正が他のモジュールに大きく影響し得るため，アジャイルな開発保守の達成が阻害される．このようなモノリシックアーキテクチャの問題を解消するため，2014 年に Martin らによって，マイクロサービスアーキテクチャ（MSA）が提唱された[9]. MSA では，1 つのアプリケーションを機能ごとに分割したものをマイクロサービスとして扱い，それぞれのマイクロサービスが独立して開発及びデプロイを行えるようにする．これにより，工数削減，開発サイクルの短縮，スケーラビリティの向上など様々な恩恵を受けることができる．また，将来的には，サードパーティ製のマイクロサービスが流通することが予想され，これらのマイクロサービスを組み上げてサービスを構築するような開発も行われると考えられる[2][3].

しかし，マイクロサービスごとに独立して開発が行われることで，モノリシックアーキテクチャの際には顕在しなかった問題が発生する．その一つが，セキュリティバイデザインの達成の困難性である．モノリシックアーキテクチャであれば，開発チームとフロントエンドサービス，バックエンドサービスが一塊であるため，設計段階からセキュリティを確保することも比較的容易である．一方，MSA においては，機能ごとに分割されたマイクロサービス同士が独立して開発が行われるため，個々のマイクロサービスの改修・更新が進むにつれて，サービス全体のセキュリティの一貫性が崩れる可能性がある．また，サードパーティ製のマイクロサービスを統合してサービスを開発する際には，マイクロサービスごとのセキュリティポリシによってはコンフリクトが発生し得る．

2.2 MSA におけるセキュリティポリシのコンフリクト

文字列入力を受け付ける変数に関して，異なるセキュリティポリシの下に開発された 2 つのマイクロサービス A，B を統合して，新たなサービス α を開発する場合を考える（図 1）．ここで，A，B，α の想定は以下の通りとする．マイクロサービス A は，ユーザからの文字列入力を司るサービスである．A におけるセキュリティ対策では，入力値に対してエスケープ処理を行うポリシを設けることで文字列を無害化する．マイクロサービス B は，文字列型の内部変数に従って，ページの表示内容を変化させるサービスである．B におけるセキュリティ対策では，想定される入力値を定義したホワイトリストを用意し，ホワイトリストに基づいた入力値の検証を行うことで，文字列を無害化する．サービス α では，マイクロサービス A に入力された文字列をマイクロサービス B の入力として受け渡すことによって，ユーザから受け付けた文字列に応じてページの表示内容を変えるサービスを実現する．

MSA では，マイクロサービス間の値の受け渡しを行うことは推奨されていないが，実際の開発においては，アプリケーションの仕様上，値の受け渡しが必要になる場合が往々にして存在する．あるマイクロサービスから他のマイクロサービスへの値の受け渡しには，例えば，Custom Element を用いたカスタムイベント[10]を定義することで実装される．サービス α において，カスタムイベントを用いてマイクロサービス A に入力された文字列をマイクロサービス B へ受け渡しを行うにあたり，マイクロサービス A は自身のセキュリティポリシに従い，エスケープ処理によって入力値の文字列を変更している．この結果，マイクロサービス B への入力値が，マイクロサービス B にとっては想定外の文字列になり，A から B への値の受け渡しにあたってコンフリクトが発生する．

これに対応するためには，マイクロサービス A と B の間でセキュリティポリシの調停を行う必要がある．しかし，マイクロサービスを越境した設計変更を生じさせてしまうことは，MSA のコンセプトから外れることになる．また，A や B がサードパーティ製のマイクロサービスであった場合には，A あるいは B の開発元に改修を依頼することは実質的に不可能であろう．

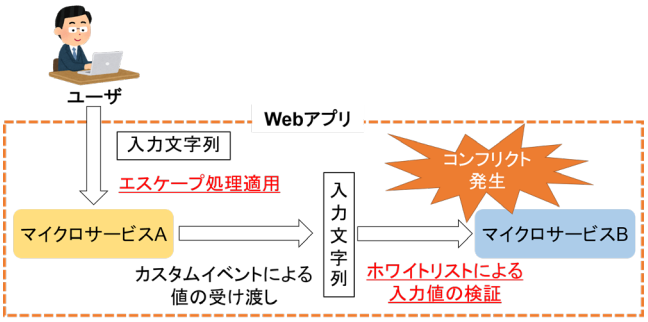

図 1　ポリシのコンフリクトの例

Figure 1　Example of Policy Conflict.

2.3 セキュアプログラミングガイドライン

マイクロサービスを統合して新規サービスを開発する際にマイクロサービス間でセキュリティポリシのコンフリクトが発生しないように，マイクロサービス開発において留意すべき点を明確化し，これを，個々のマイクロサービスを開発・実装する際に順守すべきセキュアプログラミングガイドラインとしてルール化するという方法が考えられる．しかし，セキュアプログラミングガイドラインの導入は，サービスの開発に注力したい開発者にとって技術的及び作業的負担が大きいという問題がある．これは，ガイドライン順守の不徹底を招く要因となり，Web アプリケーションに脆弱性を潜在させてしまう原因となる恐れがある．また，セキュアプログラミングガイドラインはマイクロサービスの実装方法をある程度共通化させることを意味する．実装方法の画一化（共通化）は，頻繁にデプロイが行われるマイクロサービスにおいて，その都度ポリシを更新する必要があり，マイクロサービスの自由度を狭めてしまう可能性を孕み得る[11].

2.4 Content Security Policy

Content Security Policy（CSP）とは，クロスサイトスクリプティング（XSS）やデータインジェクション攻撃などの特定の種類の攻撃を検知し，影響を軽減するためのセキュリティ機構である[12]． CSP を導入し，設計段階において決定したセキュリティポリシを Web アプリケーションに適用することにより，セキュリティバイデザインの達成が期待できる．

しかし，サードパーティ製のマイクロサービスを統合して新たなサービスを開発するケースにおいては，個々のセキュリティポリシの下で設計・実装されている複数のマイクロサービスを結合する形になる．すなわち，新規サービスを企画･設計する段階で決定したセキュリティポリシを，既に実装が完了しているマイクロサービスに遡及させることはできない．このため，新規サービスのセキュリティポリシとマイクロサービスのセキュリティポリシがコンフリクトする場合には，その不整合が，2.2 節で例示したようなマイクロサービス間のセキュリティポリシのコンフリクトとして顕在することになる．

2.5 API Gateway

API Gateway [5]とは，フロントエンドとバックエンドの中間に位置する MSA のセキュリティ機構であり，API の管理を始めとして，セキュリティポリシの作成及び実装を行うことで様々なセキュリティ保護を施すことができる．2.2 節に示した新規サービスとマイクロサービスのセキュリティポリシのコンフリクトの問題に対しても，API Gateway にポリシ間の調停を依頼することが可能である．しかし，API Gateway はフロントエンドとバックエンドの中間に配置されるが故に，フロントエンドで攻撃が完結してしまうようなサイバー攻撃に対しては，セキュリティ保護を行うことができない．

2.6 DOM Based XSS 攻撃

DOM Based XSS 攻撃は，XSS の一種であり，Web ブラウザ上で動的に HTML 生成を行う際に，悪意のあるスクリプトが埋め込まれてしまうことで成立する攻撃である[13]．攻撃者は，攻撃用リンク（正規の URL のアンカ（#）やクエリパラメータ（?）などの後に有害なスクリプトを記述したもの）を何らかの方法で被害者に送信する（図 2①）．Web ブラウザ側で動的生成される HTML コンテンツを提供する Web サーバの場合，被害者がそのリンクにアクセスすることにより，Web サーバに正規の URL のみがリクエストとして送信される（図 2②）．リクエストを受けた Web サーバは，Web ページを被害者に送信する（図 2③）．被害者の Web ブラウザが Web ページの HTML を動的に生成する際に，Web ブラウザ内でアンカやクエリパラメータ以降の有害なスクリプトが適用される（図 2④）．この結果，HTML コンテンツが DOM Based XSS の脆弱性を含んでしまっていた場合には，被害者の Web ブラウザ上で有害なスクリプトが実行されてしまう．

Reflected XSS 攻撃や Stored XSS 攻撃では，バックエンド（Web サーバ上）で HTML が動的生成される際に成立する攻撃であるため，バックエンドを構成するマイクロサービスの一部に XSS 脆弱性が存在していたとしても，API Gateway）におけるセキュリティ機構でその検知・対策が可能である．しかし，DOM Based XSS 攻撃はフロントエンド（Web ブラウザ内）で攻撃が完結してしまうため，API Gateway による救済が不可能である．

2.7 MSA におけるセキュリティ課題

2.1～2.2 節で述べたように，MSA の開発アプローチは，セキュリティバイデザインとの親和性が低く，マイクロサービス間のセキュリティポリシのコンフリクトの調停が難

しい．このため，2.4 節で述べたように，CSP の機構を導入していても，バックエンド側に XSS 脆弱性が潜在したまま残ってしまう可能性が否定できない．そして，2.5～2.6 節で述べたように，DOM Based XSS 攻撃は API Gateway による防御が不可能である．よって，現状，MSA において，DOM Based XSS 攻撃に対する有効的な対策が存在していないと考えられる．

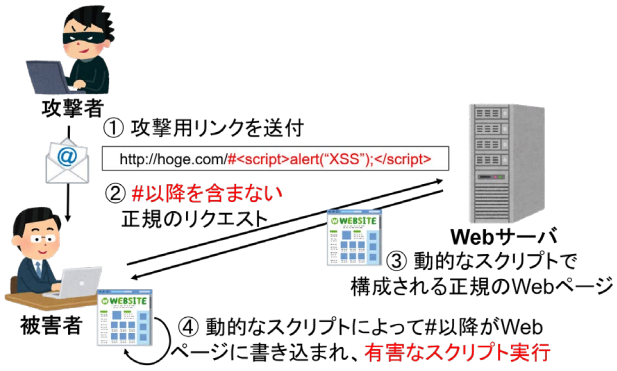

図 2　DOM Based XSS の仕組み

Figure 2　How DOM Based XSS works.

2.8 課題解決に向けてのアプローチ

セキュリティバイデザインに基づいた開発においては，Web アプリケーションの設計段階でセキュリティ要件が精査されて，セキュリティポリシとして仕様書に組み込まれる．よって，個々のマイクロサービスを統合して Web アプリケーションを構築した場合に，その Web アプリケーションが仕様書通りに動作することを保証できれば，MSA 型の Web アプリケーションの開発におけるセキュリティバイデザインが達成されたと考えて良いであろう．

そこで本稿では，2.7 節で示した MSA のセキュリティ課題を解決するためのアプローチとして，テストベースホワイトリスト[8]の適用を検討する．テストベースホワイトリストとは，ホワイトリストの作成を Web アプリケーション開発の最終段階に実施される開発テストと結合し，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する手法である．Web アプリケーション利用時には，開発者の想定外の入力（未テストの入力）による悪意のある動作が実行されることを禁止することができる．

開発テストは，Web アプリケーションのリリース前に，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，テストベースホワイトリストを MSA のアプリケーション開発にて採用することにより，MSA のセキュリティ課題が Web アプリケーションの開発工程の中で自動的に解決される．本稿では，DOM Based XSS の対策を具体例として採り上げ，MSA におけるテストベースホワイトリストの有用性の検討を行っていく．

3. 既存研究

3.1 XSS 対策

3.1.1 Content Security Policy

Web アプリケーション開発者の意図しないスクリプトが実行されてしまうことが XSS の実態である．したがって，実行されようとしているスクリプトが開発者の意図したものであるか否かを判断し，開発者の意図したスクリプトに限定して Web アプリケーションならびに Web ブラウザ上での実行を許可することによって，XSS を防止できる．具体的には，ホワイトリストによるスクリプトの実行制御がこれに該当する．ホワイトリスト型対策は，ホワイトリストに定義されているスクリプト以外の実行が全て禁止されるため，多様化した有害なスクリプトやゼロデイ脆弱性に対しても，頑健な XSS 対策といえる．

Content Security Policy（CSP）[12]は，モノリシックアーキテクチャで開発された Web アプリケーションにおける一般的な XSS 対策の 1 つであり，ポリシベースのホワイトリスト対策である．CSP における DOM Based XSS 対策としては，Trusted Types [14]というディレクティブが挙げられる．Trusted Types を用いて，DOM 操作を行う関数が満たすべきセキュリティポリシを定義することにより，Trusted Types オブジェクト（当該ポリシを満たしたオブジェクト）のみに DOM 操作が許可される．これにより，セキュリティポリシから逸脱する DOM 操作を禁止することができる．

しかし，MSA において Trusted Types を適用することを試みた場合，独立して開発されるマイクロサービスごとに異なるセキュリティポリシが実装され得る．この結果，種々のマイクロサービスを統合して新規サービスを開発する際に，セキュリティポリシのコンフリクトが起こり得る．CSP はモノリシックな Web アプリケーションを想定して開発されているため，現在のところ，1 つの Web アプリケーションの中に Trusted Types に関する複数のポリシを併存させることができない．このため，マイクロサービス間のセキュリティポリシの調停を行うためのセキュリティポリシを新たに定義し，これを新規サービスのセキュリティポリシの中に記載するという方法も採れない．よって，従来のモノリシックアーキテクチャにおいては DOM Based XSS 攻撃に対する一般的な対策であった CSP（Trusted Types ディレクティブ）は，MSA においては有効的な対策とはいえない．

3.1.2 API Gateway

MSA における XSS 対策は，API Gateway による対策が一般的である[5]. Amazon Web Service が提供している Amazon API Gateway [15]は，API Gateway の一種であり，基本的な

機能は，クライアントからのリクエストを受取り，マイクロサービスにルーティングを行うことである．この際，AWS WAF [16]や AWS Lambda [17]と連携することで，SQL インジェクションや XSS などの様々なインジェクション攻撃から，Web アプリケーションを保護することができる．

しかし，検知できるインジェクション攻撃は，API Gateway を経由する攻撃に限るため，2.6 節で述べたように，フロントエンドで攻撃が完結してしまうような DOM Based XSS 攻撃に対しては無力である．

3.2 テストベースホワイトリスト

3.2.1 概要

ポリシベースホワイトリストによる XSS 対策においては，必要十分なセキュリティポリシを作成するための方法論が体系化されておらず，開発者が経験的あるいは暗黙知的にセキュリティポリシが作成されていた．つまり，開発者に Web アプリケーションセキュリティに関する相応の知識が要求されることとなる．また，Web アプリケーションの複雑化に伴い，仮に十分なセキュリティ知識を有した開発者であっても，セキュリティポリシの設定にエラーが混入する可能性は大いにある[8].

このような従来のポリシベースホワイトリストにおける問題に対して，井上らは，Web アプリケーションの開発工程の最終段階で実施される開発テストの結果に基づいてホワイトリストを作成するテストベースホワイトリスト方式を提案した[8]．具体的には，Web アプリケーション開発の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することにより，Web アプリケーション利用時に想定外の入力（未テストの入力）によってスクリプトが実行されることを禁止した．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行われる開発テストを通じて自動的に生成される．

3.2.2 検知手法

テストベースホワイトリストでは，テスト工程において生成された HTML ソースコードの「スクリプト構造（HTML ソースコードからインラインスクリプト部分のみを抽出したもの）」がすべてホワイトリストとして登録されている（図 3）．本番環境に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（意図されたパラメータが入力される限り），Web アプリケーションによって生成される HTML 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．攻撃者によって，開発者が想定していないパラメータが入力されると，Web アプリケーションは，開発者の意図しないスクリプトが注入された HTML 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 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わる（図 4）．テストベースホワイトリストによる XSS 対策では，このような原理で XSS を検知する．

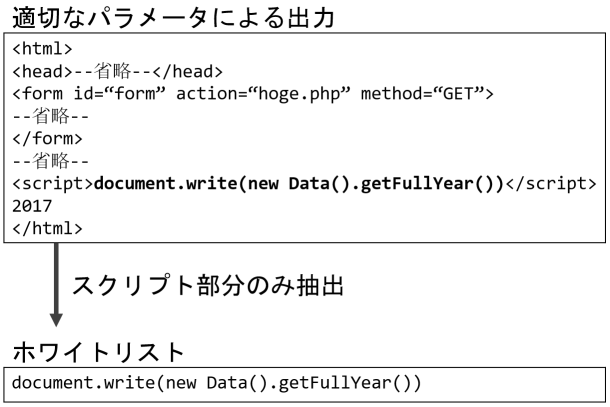

図 3　ホワイトリストの作成例（文献[8]の図 2）

Figure 3　Example of whitelist.

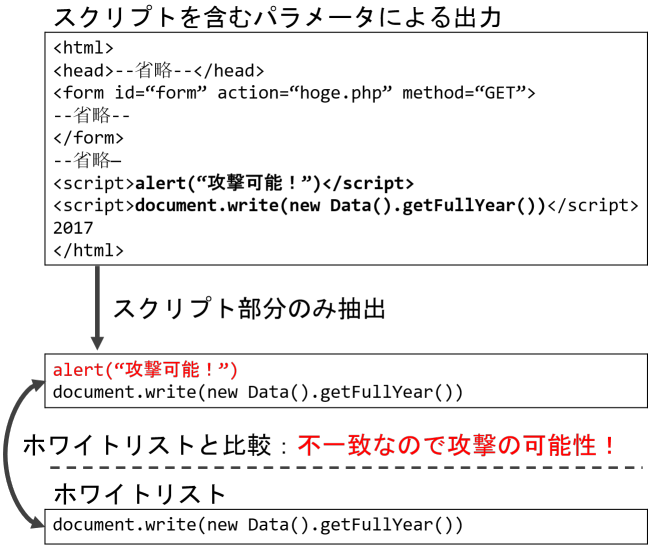

図 4　XSS の可能性がある場合の例（文献[8]の図 3）

Figure 4　Example of XSS attack.

3.2.3 MSA におけるホワイトリストの形式

MSA においては，Web アプリケーションを構成するすべてのモジュール（マイクロサービス）が，JavaScript 処理や Web コンポーネントを介してランタイム時にフロントエンド側（Web ブラウザ内）で統合される．ここで，各マイクロサービスは非同期処理で読み込まれて統合される．この場合，動的生成された HTML ソースコードのスクリプト構造は，常に一定の構造になるとは限らない．このため，正常な入力パラメータであったとしても，開発テスト時にホワイトリストとして登録されたスクリプト構造と，本番環境のスクリプト構造とが異なり得る．よって，テストベースホワイトリストを MSA の DOM Based XSS 対策として

活用するためには，非同期処理による HTML ソースコードの変化に対応する必要がある．

4. 提案手法

4.1 概要

本稿では，MSA において，テストベースホワイトリストのアプローチのセキュリティ対策を導入することで，個々のマイクロサービスを開発するにあたっての独立性を維持しながら，Web アプリケーション全体のセキュリティバイデザインが達成できる仕組みを提案する．提案手法の有用性を示すために，MSA において未だ有効的な対策が成されていない DOM Based XSS 攻撃を対象として，提案手法の具体的な設計及び実装を行っていく．

4.2 ホワイトリストの形式と攻撃検知の仕組み

文献[8]で提案されたテストベースホワイトリストは，モノリシック型の Web アプリケーションの XSS 対策を対象としていた．このため，ホワイトリストの形式として HTML ソースコードのスクリプト構造を採用することによって，効果的な XSS 攻撃の検知を実現していた．一方，MSA 型の Web アプリケーションにおいては，3.2.3 項で述べたように，非同期処理によって動的生成される HTML ソースコードが毎回変化し得るため，スクリプト構造をホワイトリストとして用いることは不適である．

そこで，本稿では，DOM Based XSS 攻撃を行う上で，攻撃者によって有害なスクリプトが埋め込まれる可能性がある「ソース（例：location.hash）」と，ソースに含まれる文字列を受け取り，文字列から JavaScript を生成及び実行する「シンク（例：document.write）」に注目する．DOM Based XSS 攻撃においては，有害なスクリプトの注入はソースとシンクによって発生する[18]ため，HTML ソースコードから構築された DOM ツリーの中で，ソースとシンクのペアによって構成される親子関係（本稿では「ソース・シンク親子関係」と呼ぶ）を抽出してホワイトリストとして定義することで，DOM Based XSS 攻撃の発生を検知することができる．

提案手法における DOM Based XSS 攻撃検知機構は，クライアントの Web 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，その Web アプリケーションを利用するたびに，Web ブラウザにおいて HTML ソースコードが動的生成される．その際，Web ブラウザ内では，HTML ソースコードのレンダリングのために DOM ツリーが構築されている．この DOM ツリーに含まれるすべてのソース・シンク親子関係を抽出し，ホワイトリストに登録されているソース・シンク親子関係と比較する．Web ブラウザの内部処理の中で構築されたソース・シンク親子関係が，ホワイトリストに登録されているソース・シンク親子関係のいずれとも一致しない場合，DOM Based XSS 攻撃が発生していると判断される（図 5）．

各マイクロサービスが非同期処理によって読み込まれ，Web ブラウザ内部で構築される HTML ソースコード全体の DOM ツリーが変わったとしても，個々のソースとシンクのペアは変化しない．従って，個々のソース・シンク親子関係を検査してやれば，非同期処理で実装されている MSA の Web アプリケーションに対しても，安定した DOM Based XSS 攻撃検知が達成される．

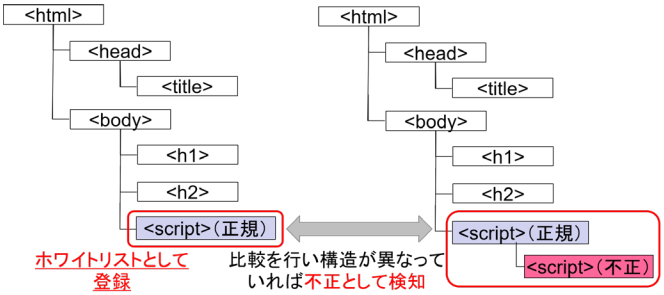

図 5　提案手法による DOM Based XSS 攻撃検知

Figure 5　Detection of DOM Based XSS Attacks in the Proposed Method.

4.3 ホワイトリストの自動作成

4.2 節で述べたように，提案手法による DOM Based XSS 検知はユーザの Web ブラウザ上で実行される．これを実現するためには，次の 3 つの機能が必要となる．1 つ目に，Web アプリケーションの開発テストを通じてホワイトリストを自動生成する機能，2 つ目に，Web アプリケーションとホワイトリストの関連付けを行う機能，3 つ目に，Web ブラウザ内で動的生成される HTML ソースコードをホワイトリストによって検査する機能である．

このうち，2 つ目の機能については，文献[8]と同様の手段で実現可能である．3 つ目の機能については，4.2 節で説明した．そこで，本節では，1 つ目の機能について詳述する．

既存研究文献[8]では，Selenium [19]と呼ばれる Web アプリケーション用テストツールを用いて，ホワイトリスト自動生成機能が実装されている．Selenium は，Web ブラウザのオートメーションツールであり，自動で Web ブラウザを操作することで Web サイトの動作のテストを行うことができる．Selenium では，テストを実施する開発者が「ユーザからの入力（入力データ）」と「その入力に対して生成されるべき Web ページ（教師データ）」のペアから成る検査項目を，テストケースとして用意する．ここで用意したテストケース中のすべての検査項目に対し，「Web アプリケーションに入力データを入力→その結果生成された Web ページを教師データと比較」というテスト手順を繰り返す

ことによって，ソフトウェアの動作を自動検査する．

提案手法においても，Selenium に DOM 構造抽出機能をアドオンし，テスト中に得られる Web ページのソース・シンク親子関係を収集することによって，テスト工程を通じたホワイトリストの自動生成を実現する．DOM 構造抽出アドオンが実行する具体的な処理は，以下の２つである．

（1） Selenium によるテスト結果が，開発者が想定した通りの結果であった場合には，Web ブラウザ内に構築されている DOM ツリーからすべてのソース・シンク親子関係を抽出する．

（2） 抽出したソース・シンク親子関係をホワイトリストに追加する．ホワイトリストは，Web アプリケーションによって生成される Web ページ単位で保存される．

5. まとめ

本稿では，MSA 型の Web アプリケーションの開発において，テストベースホワイトリストのアプローチを採用することで，開発テストを通じてマイクロサービス間のセキュリティポリシの調停を開発者が意識せずに達成できる仕組みを提案した．MSA において未だ有効的な対策が成されていない DOM Based XSS 攻撃の対策方法を取り上げ，提案手法の有用性について検討をした．

今後は，マイクロサービス間でセキュリティポリシのコンフリクトが発生する状況を整理するとともに，具体的な Web アプリケーションを用いての概念実証を通じて，提案手法の実装および有効性検証を行っていく．

参考文献

[1] D. S. Linthicum: Practical Use of Microservices in Moving Workloads to the Cloud, IEEE Cloud Computing, vol. 3, no. 5, 2016, pp. 6-9.

[2] B. Butzin, F. Golatowski and D. Timmermann: Microservices approach for the internet of things, 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation, 2016, pp.1-6.

[3] T. Bui, S. Rao, M. Antikainen, T. Aura: XSS Vulnerabilities in Cloud-Application Add-Ons, Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, 2020, pp.610-621.

[4] T. Yarygina and A. H. Bagge: Overcoming Security Challenges in Microservice Architectures, 2018 IEEE Symposium on Service-Oriented System Engineering, 2018, pp.11-20.

[5] Dawei Yang, Yang Gao, Wei He, Kai Li: Design and Achievement of Security Mechanism of API Gateway Platform Based on Microservice Architecture, Journal of Physics Conference Series, vol. 1738, no. 1, 2021.

[6] Nuno Mateus-Coelho, Manuela Cruz-Cunha, Luis Gonzaga Ferreira: Security in Microservices Architectures, Procedia Computer Science, Volume 181, 2021, pp. 1225-1236.

[7] P. Kothawade and P. S. Bhowmick: Cloud Security: Penetration Testing of Application in Micro-service architecture and Vulnerability Assessment., Dissertation, 2019.

[8] 井上佳祐，本多俊貴，向山浩平，大木哲史，堀川博史，西垣正勝,テストベースホワイトリストと CSP の組合せによる効果的な XSS 対策の実現,情報処理学会論文誌, vol. 61, no. 9, pp.1374‐1387, Sep. 2020.

[9] “Microservices”, https://martinfowler.com/articles/microservices.html. (参照 2021-08-16).

[10] “CustomEvent | MDN Web Docs”, https://developer.mozilla.org/ja/docs/Web/API/CustomEvent. (参照 2021-08-16).

[11] Li X, Chen Y, Lin Z, Wang X, Chen JHao: Automatic Policy, Generation for Inter-Service Access Control of Microservices, 30th USENIX Security Symposium, 2021, pp.3971-3988.

[12] “コンテンツセキュリティポリシー (CSP) - HTTP | MDN”, https://developer.mozilla.org/ja/docs/Web/HTTP/CSP. (参照 2021-08-16).

[13] “「DOMBasedXSS」に関するレポート｜IPA テクニカルウォッチ”, https://www.ipa.go.jp/files/000024729.pdf. (参照 2021-08-16).

[14] “CSP: trusted-types | MDN Web Docs)”, https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Content-Security-Policy/trusted-types. (参照 2021-08-16).

[15] “Amazon API Gateway（規模に応じた API の作成，維持，保護）| AWS”, https://aws.amazon.com/jp/api-gateway/. (参照 2021-08-16).

[16] “AWS WAF（ウェブアプリケーションファイアウォール） | AWS”, https://aws.amazon.com/jp/waf/. (参照 2021-08-16).

[17] “AWS Lambda（イベント発生時にコードを実行）| AWS”, https://aws.amazon.com/jp/lambda/. (参照 2021-08-16).

[18] Sebastian Lekies, Ben Stock, and Martin Johns: 2013. 25 million flows later: large-scale detection of DOM-based XSS, CCS 2013, pp.1193-1204.

[19] “Selenium”, https://www.selenium.dev/. (参照 2021-08-16).