Vol.2016-DPS-166 No.2
Vol.2016-CSEC-72 No.2
2016/3/3

オブジェクト指向のWebアプリケーションに対するXSS攻撃脆弱性の静的解析

林 昌吾[1,a)]　松浦 幹太[1,b)]

概要：Web アプリケーションは日常の様々な場面で利用されるが, その中には脆弱性が存在するものも多い. 中でもクロスサイトスクリプティング (XSS) 攻撃は数多くの研究がなされてきているにも関わらず Web アプリケーションの脆弱性の中で常に首位を争う脅威となっている攻撃であり, 攻撃手法は多様化している. 防御策について, Open Web Application Project (OWASP) のようにチェックシートを公開している組織も存在するが, 開発者全員がこれらを基に安全な実装するのは容易ではない. それ故, 脆弱性を自動で検知するツールの有用性は高く, そのような目標を達成しようとしている既存研究も存在するが, 検知精度が十分ではない. そのような研究も存在する. しかし、既存研究では精度が低い. 特に, スクリプト言語に適用可能なオブジェクト指向に対する研究はあまりなされていない. そこで本研究では 2 方向後方脆弱性分析とクラスキャッシュという概念を導入して, オブジェクト指向で実装されたソースコードの XSS 攻撃脆弱性を静的解析するための手法の提案と実装評価を行う.

Static analysis of XSS attacks vulnerabilities in Web applications implemented with object oriented programming

Shogo Hayashi[1,a)]　Kanta Matsuura[1,b)]

1. はじめに

Web アプリケーションは日常生活の様々な場面で利用される. しかし, インターネットに出回っている Web アプリケーションには多くの脆弱性が存在する. 中でもクロスサイトスクリプティング (Cross site scripting: XSS) 攻撃は特に深刻なセキュリティ攻撃の内の 1 つである. XSS は 1990 年代から存在が確認されており [1], これまでに 10 年以上に渡って数多くの研究がなされてきている. Open Web Application Security Project(OWASP) [2] という,Web アプリケーションで最も深刻な脆弱性の上位 10 位を発表を行っている団体がある. 世の中に XSS の被害が浸透していっているにも関わらず,OWASP によると,2001 年の設立以来常に対象となっているものであり, インターネットの利用者や Web サイトの運営者が被害を受けている. このような現状に対して Hydara 氏らは 2015 年に Web アプリ開発者は各フェーズでもっとセキュリティを意識し, 研究者は XSS を排除することに意識をより傾けるべきだと主張している [3].

OWASP のように XSS 攻撃の防御のためにチェックシートを公開している組織もある. しかし, これらは複雑で量も多いため Web アプリケーション開発者全員が表を基に適切に実装することは容易ではなく, 対策に対して漏れが生じる可能性が極めて高い. さらには, チェックシートに記述されている対策もまた, 完全なものではない. そのため, 開発者が実装したコードに存在する脆弱性を自動検知できるツールがあれば大いに役に立つ. 実際, そのような研究もなされてきている.

Web アプリケーションの中でも, PHP のプログラミング言語によって実装されたサーバサイドの言語は全体の内, 80 %以上を占める [4]. PHP で書かれたコードに対する脆弱性の自動検知に対する研究はされてきている [5], [6], [7],

[1] 東京大学生産技術研究所
Institute of Industrial Science, the University of Tokyo
[a)] h-shogo@iis.u-tokyo.ac.jp
[b)] kanta@iis.u-tokyo.ac.jp

[8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. しかしながら, 既存研究では精度が高いとは言い難い.

また, オブジェクト指向言語によって書かれたコードの解析に対する研究もなされてきている [18]. しかし, これらは Java 言語などの型付き言語を対象としたものであり, 型の概念の弱い PHP のようなスクリプト言語には直接適用することはできない. そのような現状に対して,Dahse 氏らによって 2014 年に一部の攻撃を対象にオブジェクト指向言語に対する脆弱性の検知に対する研究を発表している [14], [15]. これは PHP Object Injection(POI) 攻撃 [*1] を対象としたものである. しかし,PHP によって実装されたものに限定され, また, Web アプリケーションの脆弱性全体の観点では十分な対策とは言い難いものである.

本稿では, 型の概念の無いスクリプト言語を対象に, 脆弱性の自動検知の静的解析を行う. セキュリティ攻撃の中でも最も問題の大きいとされる XSS 攻撃について, クラスキャッシュと 2 方向後方脆弱性分析という概念を導入して, オブジェクト指向言語の静的解析を行う. 実装は PHP を対象に行った.

2 章では既存研究, 3 章ではオブジェクト指向に対する Web アプリケーションの静的解析の提案手法, 4 章で評価, 5 章で今後の課題, 6 章でまとめとなる.

2. 既存研究

2.1 XSS 攻撃の静的解析

Web アプリケーションにおける XSS 攻撃の静的解析の研究の特徴の比較に関して以下の研究はすべて PHP を対象としたであり, 研究・ツール名と検査対象の性質の対応の可否についてまとめたものが**表 1** となる. この他の既存研究は Pixy [16], [17] に基づいて安全なバリデーションの手法を提案するものである, もしくは,RIPS [14], [15] の中で採用されているものであるも, 表から割愛している. ただし, Pixy と RIPS に関しては論文の内容と合わせて実際に検証した結果であるが, XSSDM [19] に関しては論文の内容を基に作成したものである.

Pixy は,XSS 攻撃については Reflected XSS のみ, オブジェクト指向については静的な関数の呼び出しの時, かつ静的な呼び出しの階層が浅い時のみ検出可能であった. また,Pixy は PHP4.x までしか対応しておらず, PHP5.x 以降の記述では一部 syntax error と判定されるものがあり, 特に PHP5.x 以降でオブジェクト指向に対して大幅なアップグレードを行っているため. また,Pixy では解析するファイル毎にファイル名を個々に指定しなければならず, 複数の PHP ファイルにまたがる大規模な Web アプリケーションの静的解析には手間のかかる.

RIPS は, XSS 攻撃については実際に検証したところ Reflected XSS のみの検出であったが, RIPS が参照している Xie らの手法の論文中では Stored XSS も検出したという結果を報告している [13]. オブジェクト指向については PHP Object Injection(POI) 攻撃と呼ばれる攻撃に対する検出は可能であった. ただし, これは静的な呼び出しによるものだけであり, 動的な呼び出しについては検出することができない

XSSDM は, 論文の内容を基にまとめたものである [19]. 論文中で Gupta らはオブジェクト指向には非対応であると主張し, 評価ではオブジェクト指向で実装されたものは除いて評価を行っている. また, XSSDM の手法は HTML 文脈に着目したものである. タグにおけるユーザ入力の位置づけを基に脆弱性の判断を行うものであり, 厳密にどの種類の XSS 攻撃に対応できるかに関する言及はないが, 実装次第で Reflected XSS 及び Stored XSS に対応できる手法となっている.

以下, 本稿において主に拡張した 2 つの論文を紹介する. 一つは RIPS 中でも採用されている Xie らによる関数とファイルのサマリーによる手法, もう一つは RIPS 中で POI 攻撃の検知を対象にしたオブジェクト指向に対する手法について取り上げる.

2.2 関数とファイルのサマリー

関数とファイルのサマリーを用いた検知手法 [13] は Xie らによって提唱された手法で大きく以下の 5 つの段階で行われる.

(1) 関数ごとに抽象構文木 (Abstract Syntax Tree, AST) を作成する.
(2) 制御フローグラフ (Control Flow Graph, CFG) を作成する.
(3) 基本ブロックシミュレーションを実行する.
(4) Intraprocedural 分析を行う.
(5) Interprocedural 分析を行う.

3 段階めの基本ブロックシミュレーションにおいては, スクリプト言語における動的な挙動に対して静的解析を可能にするためにシンボリック実行という概念を導入している. このシンボリック実行は大きく 3 つの段階で行われ, PHP 言語のモデル化, モデル化した PHP を基に式の評価や宣言のシミュレーション方法の定義, ブロックサマリーを作成するという流れで行われる.

そして, Intraprocedural 分析, Interprocedural 分析を経てブロックサマリーから関数サマリー, ファイルサマリーを作成するという流れのように, 前の段階で作成した情報を基に規模を拡大しながらサマリーを作成していく.

2.3 POI 攻撃でのオブジェクト指向解析の手法

Dahse ら [14] は PHP Object Injection 攻撃を対象に静的解析に取り組んだ. 論文中ではオブジェクト指向の動

[*1] Code Reuse 攻撃の一つで, PHP におけるオブジェクト指向に対する攻撃

表 1　既存研究の比較
Table 1　Comparison of existing researches

Tool	Not OOP	OOP	Dynamic OOP	Refered XSS	Stored XSS	DOM based XSS
Pixy	Detectable	Rarely	Undetectable	Detectable	Undetectable	Undetectable
RIPS	Detectable	Sometimes	Undetectable	Detectable	Rarely	Undetectable
XSSDM	Detectable	Undetectable	Undetectable	Detectable	Detectable	Undetectable

的にメソッドの中身が決まるという性質に対して大きく 3 つの手法で解決に取り組んだ. まず最初に, レシーバがあるメソッドを呼んだときにそのメソッド名が各クラスの中で一つだけであればそのクラスのメソッドに特定する. もし, これで特定できなければ次にメソッドが受け取る引数の関数に着目する. それでも特定できなければ Internal-procedural 分析を行う. 同じ関数名を持つすべてのクラスを対象に分析を行う. もしこの中に脆弱な可能性を含むメソッドが含まれていれば, 元の呼び出し元に伝える. 呼び出し元は脆弱な性質を持つプロパティとしてさらにこれの呼び出し元に伝えていくという手法で脆弱性を分析する.

3. オブジェクト指向に対する静的解析の提案手法

Web アプリケーション, 特にスクリプト言語におけるオブジェクト指向の解析を困難にする原因として, オブジェクト指向において関数の具体的な処理は実行時に動的に決定することが挙げられる. これは関数と結びつくレシーバが実行前には決定せず, 実行時にレシーバのクラスが決定し, その後関数名に一致するそのクラス中に存在する関数が実際の処理内容となるというものである. 本研究では, 主に Xie らによる関数とファイルのサマリー [13] と Dahse らによる RIPS [14] を拡張し, クラスキャッシュと 2 方向後方脆弱性分析という概念を導入して問題の解決に取り組む. このクラスキャッシュには 2 種類あり, またそれぞれ 2 種類のクラスサマリーを持つ. 最初にオブジェクト指向において XSS 攻撃が発生する状況の分類. また, それらをそれぞれ掘り下げて分類していき, クラスキャッシュと 2 方向後方脆弱性分析による分析方法を説明する. 最後に Dahse らによりオブジェクト指向の分析において課題だとする状況における分析手法ついて説明する.

3.1 オブジェクト指向における XSS 攻撃発生状況の分類

オブジェクト指向において XSS 攻撃が発生する状況として大きく 2 つに分類される.

Case 1　ユーザ入力由来の変数を引数として, echo などの脆弱を引き起こす可能性のある組み込み関数を持つクラス関数が呼び出される場合.

Case 2　ユーザ入力由来のクラスプロパティが echo 関数などの脆弱な可能性を含む組み込み関数によって呼び出される場合.

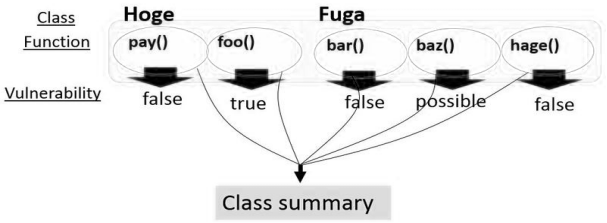

図 1　クラスメソッド用のクラスキャッシュ
Fig. 1　Class cache for class method

3.1.1 ユーザ入力由来の変数を引数とした echo など脆弱を引き起こす可能性のある組み込み関数を持つクラス関数が呼び出される場合

最初にソースコードをスキャンしてクラスメソッドの脆弱な情報をクラスキャッシュとしてキャッシュする. クラス関数の情報を含むクラスキャッシュは**図 1** のようになる. クラスキャッシュは各クラスごとにそれぞれのクラスに含まれる関数の情報, それら関数についての脆弱性について true, possible, false の 3 種類の情報を含む. true はこのクラス関数が呼び出された場合に無条件で脆弱となる場合. false はこのクラス関数が呼び出されても脆弱性に繋がる可能性のない場合. possible はクラス関数が呼び出された場合において, 関数の引数がユーザ入力由来である場合に脆弱性となる場合における情報である. クラスキャッシュのそれぞれの要素はクラスサマリーと呼び, 以下クラスサマリーの詳細を述べる.

関数におけるクラスサマリーについて説明したものについて**図 2** のとおりとなる. この時のクラスサマリーには true, possible, false の 3 種類の情報が含まれる. true の場合について, echo 関数など脆弱性に繋がる組み込み関数が呼び出される際に、呼ばれる対象が変数であり, かつその変数を後方脆弱性分析によりソースコードを遡って行った際に変数がユーザ入力由来であると分析された場合である. possible について, echo 関数など脆弱性に繋がる組み込み関数が呼び出される際に, 呼ばれる対象が変数であり, かつその変数を後方脆弱性分析によりソースコードを遡って行った際に変数が関数の引数由来であると分析された場合である. この場合, 関数が呼び出される時, 関数の引数がユーザ入力由来であれば脆弱性に繋がるが, それ以外の場合であれば脆弱性にはならないということを意味する. false について, 関数が true や possible の条件を満たさない

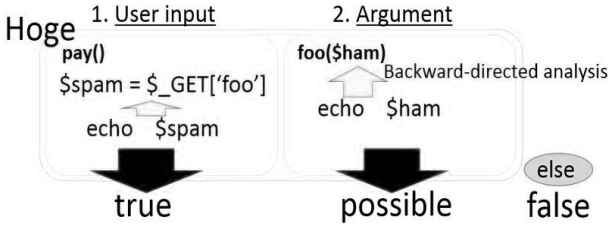

図 2　クラスメソッド用のクラスサマリー
Fig. 2　Class summary for class method

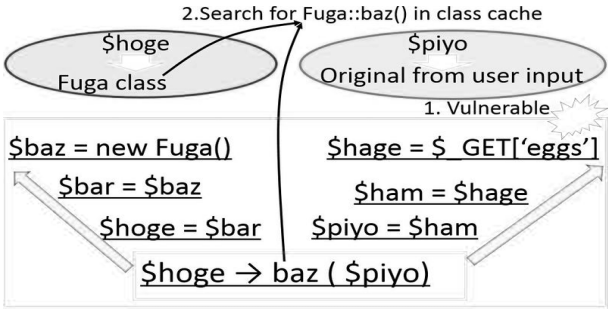

図 3　クラスメソッドにおける 2 方向後方脆弱性分析
Fig. 3　2 way backward-directed analysis for class method

場合すべての場合を意味する.

クラスキャッシュを作成後, このクラスキャッシュを基にして脆弱性分析を行う. 本研究ではオブジェクト指向の分析のために 2 方向後方脆弱性分析という概念を導入して分析を行う. 2 方向後方脆弱性分析を説明した図は**図 3** となる. $hoge→ baz($piyo) について分析を行う際に, 最初に引数に渡された$piyo 変数について一方の後方脆弱性分析を行う. この変数の代入などの繰り返しを遡って行った際に$_GET などによるユーザ入力由来であると分析された場合, 脆弱な可能性があるとして, さらにもう一方の後方脆弱性分析を行う. こちらの後方脆弱性分析は$hoge を対象に後方向脆弱性分析を行い, 変数の代入などの繰り返しを遡って行った際にインスタンスの生成部分で宣言されたクラス名を特定する. 図 3 中においては Fuga クラスの baz 関数が引数としてユーザ入力由来の変数$poyo を渡しているために脆弱であるかの可否をクラスキャッシュに情報を探索しにいく. 探索について説明したものは**図 4** となる. ここでは Fuga クラス中の baz 関数の情報を探索しにいったときに possible という情報を持っている. この possible は Fuga クラスの baz 関数が引数としてユーザ入力由来のものを受け取った場合に脆弱性となるという意味であるため, 探索の結果$hoge→ baz($piyo) は脆弱性であると判断する.

3.1.2　ユーザ入力由来の変数を持つクラスプロパティが原因となる場合

ユーザ入力由来の変数を持つクラスプロパティが原因となる場合について, プロパティに値が代入される場合とプロパティの値が呼び出される場合の 2 つの場合について考える.

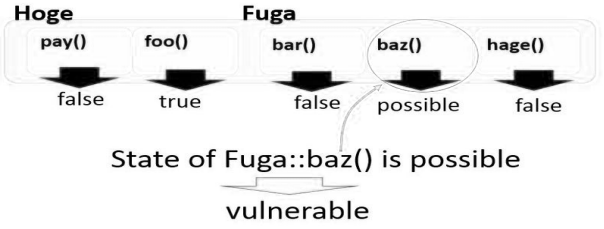

図 4　クラスメソッドの脆弱性情報の探索
Fig. 4　Search for vulnerability information of class method

オブジェクト指向において, プロパティに値が代入される場合について大きく 4 つの状況に分類する.

Case 1　クラスプロパティにユーザ入力由来の値が直接代入される場合. 例)$hoge→ obj=$_GET['input']

Case 2　クラスメソッドでセット関数を通してクラスプロパティに値が代入される場合 例) $hoge→ setParam($fuga)

Case3　コンストラクタ内でユーザー入力由来の変数がクラスプロパティに直接値が代入される場合.

Case4　コンストラクタ内で引数を通してクラスプロパティに値が代入される場合.

一つ目のクラスプロパティにユーザ入力由来の値が直接代入される場合について, 関数における分析と同様に 2 方向後方脆弱性分析を行う. これを説明したものは**図 5** となる. 最初に一方の後方脆弱性分析を行い$piyo の代入の繰り返しを遡って行く. もし, $piyo がユーザー入力由来であれば脆弱な可能性があると判断する. ここでは遡った先が$_GET['eggs'] 由来とユーザからの入力を受け取るものであるため, 脆弱な可能性があると判断する. 脆弱な可能性があると判断された場合, もう一方の後方脆弱性分析を$fuga を対象に行う. $fuga の代入の繰り返しを遡って行った先に Hoge クラスのインスタンスが生成されている箇所がある. そのため, $fuga は Hoge クラスのインスタンスを持つと分析する. 次に, Hoge クラスの hobj1 プロパティは脆弱であるという情報をプロパティ用のクラスキャッシュにキャッシュする. このクラスキャッシュの要素であるプロパティ用のクラスサマリーも関数のクラスサマリーの場合と同様のものであり, **図 6** の通りとなる.　関数用のクラスサマリーとプロパティ用のクラスサマリーの大きな違いについて説明する. 関数用のクラスサマリーは各クラスの情報, その各クラスにはそれぞれの関数の情報, それらの各関数には脆弱性の情報である true, possible, false の 3 種類の値をキャッシュしていた. 一方で, プロパティ用のクラスサマリーは各クラスの情報, その各クラスにはそれぞれ所有するプロパティの情報, それらの各プロパティには脆弱性の情報である true と false の 2 種類の情報を持ち, それぞれ true はユーザ入力由来であるため脆弱性を引き起こすもの, false はそうでない場合を表す.

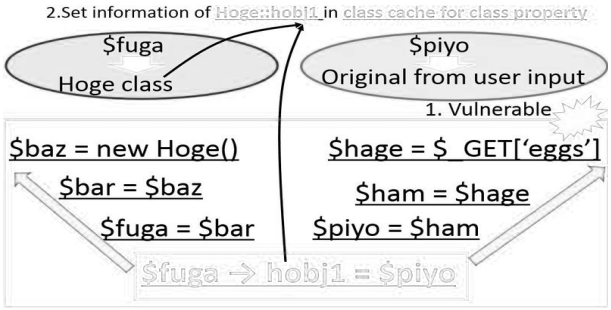

図 5　クラスプロパティにおける 2 方向後方脆弱性分析
Fig. 5　2way backward-directed analysis for class property

図 6　クラスプロパティ用のクラスサマリー
Fig. 6　Class summary for class property

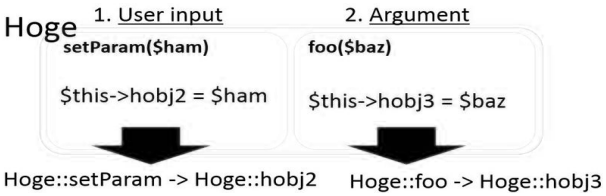

図 7　クラスメソッド用のクラスキャッシュ
Fig. 7　Class cache for class method

二つ目のクラスメソッドでセット関数を通してクラスプロパティに値が代入される場合について，　一つ目のクラスプロパティに値が直接代入される場合と同様だが, その前に関数用のクラスサマリーの拡張を行い, この拡張した情報を基にクラスプロパティに値が直接代入される場合のような処理を行う. このときの状況について説明した図が**図 7** となる. 関数用のサマリーの拡張方法について, 関数用のクラスサマリーを作成する際にクラスメソッド内でクラスプロパティに値が直接代入されている場合, このクラスメソッドから対象のクラスプロパティに対してエイリアス情報を持たせる. この場合, Hoge クラスの setParam 関数中で$this→ hobj2 = $ham という代入がなされている. そのため, Hoge クラスの setParam 関数は Hoge クラスの obj2 プロパティに対するエイリアス情報を持つ. このようなキャッシュを作っておき, もし, Hoge クラスの setParam 関数が呼び出された場合はクラスキャッシュの探索を行い, Hoge クラスの setParam 関数のクラスサマリーは Hoge クラスの hobj2 プロパティへのエイリアス情報を持つため, このクラスサマリーの脆弱性の情報をみて脆弱性の判断を行う.

三つめのコンストラクタ内でユーザ入力由来の変数がクラスプロパティに直接代入される場合について, これは一つ目のクラスプロパティにユーザ入力由来の値が直接代入される場合と同様だが, 解析のタイミングについてインスタンス生成時に行われる. ただし, 解析対象となるソースコードが実装したクラスは必ずインスタンスが生成されるのであればクラスの解析時にも可能である.

四つ目のコンストラクタ内で引数を通してクラスプロパティに値が代入される場合について, これは二つ目のクラスメソッドでセット関数を通してクラスプロパティに値が代入される場合と同様だが, 解析のタイミングについてインスタンス生成時に行われる. ただし, 解析対象となるソースコードが実装したクラスは必ずインスタンスが生成されるのであればクラスの解析時にも可能である.

次に, オブジェクト指向において, プロパティの値が呼び出される場合について大きく 2 つの状況に分類する.

Case 1　クラスプロパティが組み込み関数によって直接呼ばれる場合. 例) echo $hoge→obj

Case 2　クラスプロパティがクラスメソッドを通して呼ばれる場合 例) echo $hoge→getParam($fuga).

一つ目のクラスプロパティが組み込み関数によって直接呼ばれる場合について, これはオブジェクト指向において, オブジェクトを所有するレシーバを後方脆弱性分析を行う. この分析によってインスタンス生成部を特定することによりクラス名を特定し, クラス名とオブジェクトの組み合わせを具体的に特定する. 次に, プロパティが代入される場合の一つ目の場合におけるクラスプロパティにユーザ入力由来の値が直接代入される場合において作成したクラスキャッシュで, 先ほどのクラス名とオブジェクトの組み合わせの情報を探索する. 探索した情報が true であれば脆弱性があると判断し, false であれば脆弱性がないと判断する.

二つ目のクラスプロパティがクラスメソッドを通して呼ばれる場合について, これは関数用のクラスサマリーの拡張を行い, この拡張した情報を基にクラスプロパティが組み込み関数によって直接呼ばれる場合と同様の処理を行う. このときの状況を説明した図が**図 8** となる. 関数用のサマリーの拡張方法について, 関数用のクラスサマリーを拡張する際に, クラスメソッド内でクラスプロパティの値を返り値とする場合はこのクラスメソッドから対象のプロパティに対してエイリアス情報を持たせる. この場合, Hoge クラスの getParam 関数中で return $this→hobj1 のように hobj1 プロパティが返り値として値を返している. そのため, Hoge クラスの getParam 関数は Hoge クラスの hobj1 プロパティに対するエイリアス情報を持つ. 解析時に echo などの脆弱性を引き起こす可能性のある組み込み関数によってクラスメソッドが呼ばれた場合の解析方法を説明したものが**図 9** となる. 最初に getParam 関数における引数の由来を分析するために後方脆弱性分析を行う. ここでは$piyo 変数が代入されるのを遡って行き$_GET['eggs'] 由来であり, これがユーザ入力由来となり脆弱な可能性が

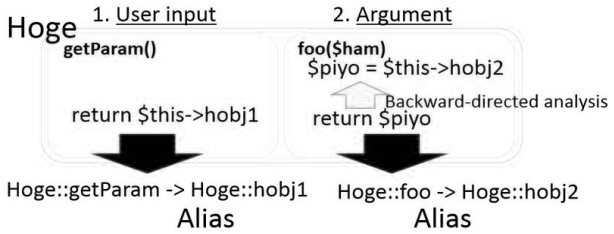

図 8　GET メソッドにより呼ばれる場合のクラスサマリー
Fig. 8　Class summary extended for case called by get method

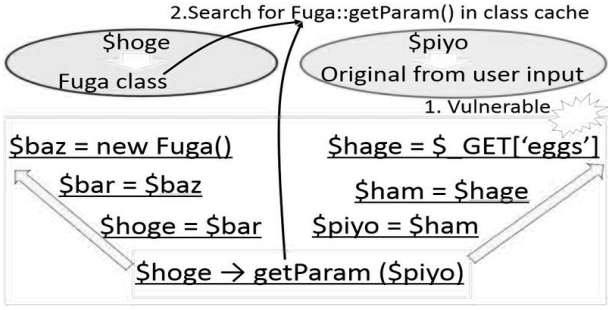

図 9　クラスメソッドによりクラスプロパティが呼ばれる場合における 2 方向後方脆弱性分析
Fig. 9　2way backward-directed analysis for class property called by class method

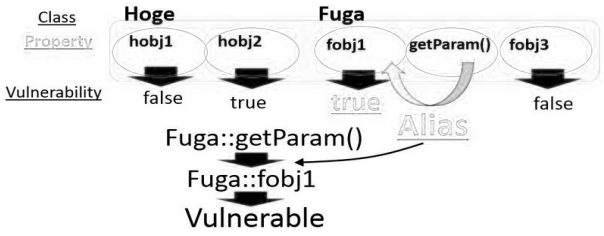

図 10　クラスメソッドによりクラスプロパティが呼ばれる場合における探索
Fig. 10　Search for class property called by class method

あると判断する．次にもう一方の後方脆弱性分析を行う．ここでは$hoge 変数の代入を遡って行きインスタンス生成部で Fuga クラスが生成されているため, Fuga クラスの getParam 関数をクラスキャッシュに探索しにいく．この時のクラスキャッシュの探索の様子が図 **10** となる．クラスキャッシュ中に Fuga クラスの getParam 関数の情報を探索し, 先ほど作成したクラスサマリーの情報を確認する．このとき, Fuga クラスの getParam 関数は Fuga クラスの fobj1 プロパティにエイリアスが張られているため, Fuga クラスの fobj1 のプロパティをクラスキャッシュ中からクラスサマリーを見つける．もしここで true という情報を持っていれば, echo $hoge→getparam($piyo) は脆弱性を持つと判定される．false であれば脆弱性がないと判定される．

3.2　議論

Dahse らは論文中でオブジェクト指向の分析が困難である状況として大きく以下の 3 つを挙げている．

Case 1　クラスメソッド中から別のクラスメソッドが呼ばれている場合

Case 2　引数にインスタンスが格納された変数が渡される場合

Case 3　関数内でインスタンスが格納された変数について global 変数が使用される場合

一つ目のクラスメソッド中から別のクラスメソッドが呼ばれている場合について, Dahse らは動的なクラスメソッドの呼び出しは実行時でないと決まらないためこのような状況に対応するのは困難であると述べている．しかし, 論文中で述べられている例について, 別のクラスメソッドが現れた場合も本稿で提案したクラスキャッシュの作成時のクラスサマリーの情報を参照することで対応できる．

二つ目の場合である引数にインスタンスが格納された変数が渡される場合について, クラスメソッド用のクラスキャッシュを作る際に true, possible, false に加えて object という情報を追加し, object とセットでクラス名を格納することで, 分析時に引数由来のインスタンスからメソッドを呼び出す際にはこの情報を参照することで分析可能になる．

三つ目の関数内でインスタンスが格納された変数について global 変数が使用される場合について, これは変数名とクラス名の対応するキャッシュを新たに用意して作っておく．関数内の分析中に global で宣言された変数があり, かつこれをインスタンスが格納されていると見なし, クラスメソッドが呼ばれている場合はこのキャッシュを参照することで分析可能になる．

4.　評価

評価について, 代表的な脆弱性のパターンをテストコードとして用意したものによる評価と PHP Vulnerability test suite [25] というデータセットを用いた評価という 2 つの方法で行った．

代表的な脆弱性のパターンをテストコードについて, 以下のパターンを用意した．

Case 1　ユーザ入力由来の変数を引数として動的なメソッドが呼び出され, このメソッド中で引数を echo などの組み込み関数によって呼ばれる場合

Case 2　コンストラクタ内でクラスプロパティに直接ユーザ入力由来の値が代入され このクラスプロパティが直接 echo などの組み込み関数によって呼ばれる場合

Case 3　クラスメソッドを通してクラスプロパティを取り出して, echo などの組み込み関数によって直接呼ばれる場合

Case 4　クラスのプロパティに直接ユーザ入力由来の値を代入し, このプロパティを直接 echo などの組み込み関数によって直接呼ばれる場合

Case 5 ユーザ入力由来の変数を引数として静的なメソッドが呼び出され, このメソッド中で引数を echo などの組み込み関数によって呼ばれる場合

Case 6 インスタンスを生成して変数に代入してから, 実際にユーザ入力由来の変数を引数とした動的にメソッドが呼び出されるまでに変数の代入が繰り返される場合

Case 7 ユーザ入力由来の変数を引数として動的なメソッドが呼び出され, このメソッド中で別のクラスメソッドが呼び出され, このメソッド中で引数を echo などの組み込み関数によって呼ばれる場合

上記のパターンはいずれも検出可能と判定された.

データセットによる評価について, PHP Vulnerability test suite における CWE 79: Cross-site Scripting を対象に行った.

PHP Vulnerability test suite 中に XSS 攻撃に脆弱なファイルは 5440 個用意されている. これを RIPS で測定したところ, 276 個検出可能であった. 本提案手法により実装したもので測定したところ, 新たに 2238 個が検出可能となり, 大幅に改良されたことが確認された.

5. 今後の課題

今後の課題として, 評価の充実が挙げられる. False Positive(FP) や False Negative(FN) による評価, オープンソースの評価を行う. 2 つめとして, 今回検知可能でなかった XSS 攻撃の脆弱性の分析と実装, またはその評価を行う. 3 つめとして, XSS 攻撃の脆弱性においてソースコード自体以外の周辺環境から生じる複合的要因の分析とその検知方法の考察を行う. 4 つめとして今回対象としなかった DOM based XSS の検知についての考察が考えられる. この DOM based XSS は他の種類と比較して比較的最近発見された XSS 攻撃で [26], 現状満足な対策方法も考えられていない [20], [21], [22], [23], [24]. その要因として, DOM based XSS はクライアントサイドで実行される脆弱性であり, 攻撃を受ける側のブラウザに依存すること, もしくはクライアント側でサードパーティの JavaScript が実行されこれが原因となって脆弱性に繋がることなどが理由として挙げられる.

6. まとめ

本稿では Web アプリケーションの脆弱性における静的解析について, 特にスクリプト言語を対象にオブジェクト指向における XSS 攻撃の脆弱性の自動検知のための静的解析手法を提案し, サーバサイド言語で 80 %以上の割合で使用される PHP 言語を対象に実装を行った. XSS 攻撃の原因はサーバ側だけでなく, クライアント側のブラウザの種類やバージョン, その他様々な要素が起因してくるため完璧な対策は不可能に近いものであろう. 本研究でオブジェクト指向に対して静的解析を可能にするための手法の提案と実装を行ったことにより検知可能な XSS 攻撃の脆弱性の種類が広がった.

参考文献

[1] Grossman et al. : XSS Attacks: CROSS SITE SCRIPTING EXPLOITS AND DEFENSE, SYNGRESS, pages 480, 2007.
[2] OWASP 入手先 ⟨https://www.owasp.org/index.php/Main_Page⟩ (参照 2016-01-30)
[3] Hydara et al., Current state of research on cross-site scripting(XSS) - A systematic literature review, Information and Software Technology 58(2015), pages 170–186, 2015.
[4] W3techs: Historical trends in the usage of server-side programming languages for websites 入手先 ⟨http://w3techs.com/technologies/history_overview/programming_language⟩. (参照 2016-01-30)
[5] Balzarotti, Davide, et al.,Saner: Composing static and dynamic analysis to validate sanitization in web applications., Security and Privacy, 2008. SP 2008. IEEE Symposium on. IEEE, 2008.
[6] Dahse, Johannes, and Thorsten Holz, Simulation of built-in PHP features for precise static code analysis., Symposium on Network and Distributed System Security (NDSS), 2014.
[7] Dahse, Johannes, and Thorsten Holz, Static detection of second-order vulnerabilities in web applications, USENIX Security Symposium, 2014.
[8] Huang, Yao-Wen, et al., Securing web application code by static analysis and runtime protection, Proceedings of the 13th international conference on World Wide Web. ACM, 2004.
[9] Kneuss, Etienne, Philippe Suter, and Viktor Kuncak, Phantm: PHP analyzer for type mismatch, Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering. ACM, pages 373–374, 2010.
[10] Son, Sooel, and Vitaly Shmatikov, SAFERPHP: Finding semantic vulnerabilities in PHP applications, Proceedings of the ACM SIGPLAN 6th Workshop on Programming Languages and Analysis for Security. ACM, 2011.
[11] Wassermann, Gary, and Zhendong Su, Static detection of cross-site scripting vulnerabilities, Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on. IEEE, pages 171–180, 2008.
[12] Wassermann, Gary, and Zhendong Su, Sound and precise analysis of web applications for injection vulnerabilities, ACM Sigplan Notices. Vol. 42. No. 6. ACM, pages 32–41, 2007.
[13] Xie, Yichen, and Alex Aiken, Static Detection of Security Vulnerabilities in Scripting Languages, USENIX Security. Vol. 6, pages 179–192, 2006.
[14] Dahse, Johannes, Nikolai Krein, and Thorsten Holz, Code Reuse Attacks in PHP: Automated POP Chain Generation, Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security(2014), pages 42–53, 2014.
[15] Dahse, Johannes, and Jorg Schwenk, RIPS-A static source code analyser for vulnerabilities in PHP scripts, Retrieved: February 28 (2010), 2012.
[16] Jovanovic, Nenad, Christopher Kruegel, and Engin Kirda, Pixy: A static analysis tool for detecting web

application vulnerabilities, Security and Privacy, 2006 IEEE Symposium on. IEEE(2006), pages 6–263, 2006.
[17] Jovanovic, Nenad, Christopher Kruegel, and Engin Kirda, Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities (Technical Report), Secure Systems Lab, Vienna University of Technology (2006), 2006.
[18] Smaragdakis, Yannis, Martin Bravenboer, and Ondrej Lhotak, Pick your contexts well: understanding object-sensitivity, ACM SIGPLAN Notices 46.1(2011), pages 17–30, 2011.
[19] Gupta, Mukesh Kumar, et al., XSSDM: Towards detection and mitigation of cross-site scripting vulnerabilities in web applications, Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on. IEEE, 2015.
[20] Parameshwaran, Inian, et al., Auto-patching DOM-Based XSS at Scale, Foundations of Software Engineering (FSE) (2015), 2015.
[21] Weissbacher, Michael, et al., ZigZag: Automatically Hardening Web Applications Against Client-side Validation Vulnerabilities, 24th USENIX Security Symposium (USENIX Security 15). USENIX Association, 2015.
[22] Stock, Ben, et al., Precise Client-side Protection against DOM-based Cross-Site Scripting, Proceedings of the 23rd USENIX security symposium(2014), pages 655–670, 2014.
[23] Lekies, Sebastian et al., 25 Million Flows Later - Large-scale Detection of DOM-based XSS, Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. ACM, 2013.
[24] Stock, Ben, et al., From Facepalm to Brain Bender: Exploring Client-Side Cross-Site Scripting, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015.
[25] PHP Vulnerability test suite 入手先 ⟨https://github.com/stivalet/PHP-Vulnerability-test-suite⟩ （参照 2016-01-30）
[26] Klein, Amit, DOM based cross site scripting or XSS of the third kind, Web Application Security Consortium, Articles 4 (2005), 2005.

付　　録

A.1　クロスサイトスクリプティング攻撃

A.1.1　概要

XSS 攻撃とは, Web サイトの実装の不備を利用して攻撃者が Web サイトの利用者に対して攻撃する手法である. これは, 主にユーザの入力に応じて Web サイトのコンテンツを生成するような場面において, サーバサイド側における利用者の入力を適切に処理しないことで生じるものである. この攻撃を受けた被害者は, ブラウザ上に保存された Cookie の情報が盗み出される, ブラウザ上で悪意のある Java Script のコードを実行させられるなど様々被害が考えられる [26].

A.1.2　分類

XSS 攻撃は大きく反射型 XSS(Reflected XSS), 持続型 XSS(Stored XSS),DOM based XSS の 3 種類に分類することができる. これらの分類の比較表は**表 A·1** の通りになる.

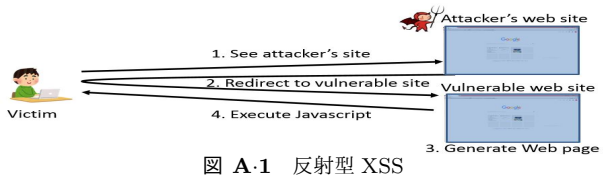

図 A·1　反射型 XSS

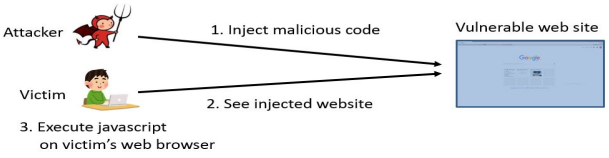

図 A·2　蓄積型 XSS

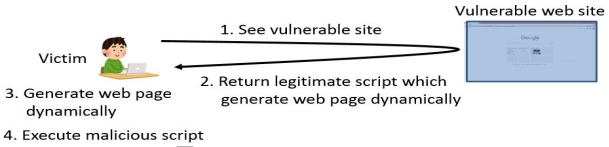

図 A·3　DOM based XSS
Fig. A·3　DOM based XSS

表 A·1　XSS 攻撃の特徴の比較
Table A·1　Comparison of features of XSS attacks

XSS の種類	XSS 攻撃実行時の処理側	攻撃コードの働き方
Reflected XSS	サーバサイド	間接的
Stored XSS	サーバサイド	直接的
DOM based XSS	クライアントサイド	間接的

A.1.3　Reflected XSS

Reflected XSS は, **図 A·1** の通りとなる. これは, 攻撃者がクエリー等に Javascript による攻撃コードを埋め込んでおいた URL を用意しておき, これを用いて利用者にアクセスさせること等で実行される.

A.1.4　Stored XSS

Stored XSS は, **図 A·2** の通りとなる. 攻撃者が予め Web サイトの入力に Javascript による攻撃コードを与えておきそれがデータベース等に保存される. 利用者がこの攻撃コードが与えられた Web サイトを訪れる時に実行される.

A.1.5　DOM based XSS

DOM based XSS は, **図 A·3** の通りとなる. 2005 年に Klein らによって最初に言葉が用いられるようになった他の種類の XSS と比較して新しい概念である [24], [26]. DOM based XSS も Reflected XSS のように攻撃者が用意しておいた URL を利用者にアクセスされて間接的に攻撃されるものである. 顕著な特徴としては, サーバサイドには攻撃コードを必ずしも送る必要がなく, サーバサイドでは検知できない状況も考えられる.