# 強化学習を用いた自律的な反射型クロスサイトスクリプティング検査手法の提案

長谷川 健人[1,a)]　披田野 清良[1]　福島 和英[1]

**概要**：近年，サイバー空間の脅威は多様化，複雑化している．ネットワークに接続された多くの機器が備えるWebインターフェースには，代表的な脆弱性の例として反射型クロスサイトスクリプティング（XSS）がある．既存のXSS検査ツールでは既知の攻撃パターンをもとに総当たり的あるいはヒューリスティックな方法で試行的に検査しており，Webページへの問い合わせ回数が膨大となることから，検査時間の増加やWebサーバの負荷増大が課題となる．そこで本稿では，強化学習を用いた自律的な反射型XSS脆弱性の検査手法を提案する．提案手法では，予め与えた反射型XSS攻撃に使われる既知の文字列の断片の置換操作と，構文解析による状態観測を通じて，従来手法で必要だった人間の介在なしに自律的に検査文字列を構成する方法を強化学習で学習する．その結果，強化学習による最適な方策の獲得から，検査時にWebサイトへの問い合わせ回数を大幅に減らすことが可能となる．評価実験を通じ，既存ツールとの比較において，提案手法は最も少ない問い合わせ回数で脆弱性を発見できることを示す．

**キーワード**：強化学習, セキュリティ, クロスサイトスクリプティング, 自律的, 攻撃

# Toward Improving Testing Cost of Reflected Cross-Site Scripting Using Reinforcement Learning

Kento Hasegawa[1,a)]　Seira Hidano[1]　Kazuhide Fukushima[1]

***Abstract:*** Reflected cross-site scripting (XSS) is a typical vulnerability in a web interface. Existing XSS testing tools use brute force or heuristic methods based on known attack patterns, which increases the testing time and the load on the web server due to a large number of requests. In this paper, we propose a method of autonomous audit testing for reflected XSS vulnerability using reinforcement learning. In the proposed method, we use reinforcement learning to learn how to autonomously construct attack strings of reflected XSS through recombination of known attack string fragments and state observation of parsed web pages. As a result, the proposed method significantly reduces the number of requests to the website during the testing process by acquiring the optimal policy through reinforcement learning. The experimental results demonstrate that the proposed method can find XSS vulnerabilities with the least number of requests compared to the existing tools.

***Keywords:*** Reinforcement learning, Security, Cross-Site Scripting, Autonomous, Attack

## 1. はじめに

近年，AIセキュリティへの関心が高まっている．AIセキュリティは機械学習とセキュリティを融合した分野であり，機械学習を手段として既存システムを防御する方法や，機械学習を攻撃対象として機械学習アルゴリズムの誤動作を誘発する方法，機械学習を攻撃の手段として既存システムを攻撃する方法が扱われる．また，Society 5.0の提唱によりIoTやAI技術を活用してサイバー空間とフィジカル空間を融合させる取り組みが加速していることから，我々

[1] KDDI総合研究所
KDDI Research, Inc.
[a)] kt-hasegawa@kddi-research.jp

の身の回りにおいて実に多くの “もの” が高度に情報化されている．高度に情報化された社会においては，自動化や遠隔操作が可能となるように多くの利便性を享受できる一方で，攻撃者にとっても攻撃の機会や方法が多様化している．特に，攻撃者が専門的知識を駆使するだけでなく，AI 技術を活用することで攻撃方法を高度化，省力化する可能性がある．以上の背景のもとで，強化学習を利用した自律的な攻撃の脅威が指摘されている．強化学習を利用することで一連の攻撃を自動化することが可能となり，攻撃者にとってはより効率的な攻撃や自身の労力の軽減を実現できる．防御者の観点からは，こうした攻撃が脅威となり得る一方で，事前の脆弱性検知にも活用できる．強化学習を利用した自律的な攻撃の可能性や，それに対する防御，脆弱性検知への応用検討は，重要な研究課題の一つである．

ここで，ネットワーク機器における脆弱性検知に着目する．代表的な脆弱性の例として反射型クロスサイトスクリプティング（XSS）がある．既存の XSS 検査ツールでは既知の攻撃パターンをもとに総当たり的あるいはヒューリスティックな方法で試行的に検査しており，Web ページへの問い合わせ（リクエスト）回数が膨大となることから，検査時間の増加や Web サーバの負荷増大が課題となる．そこで本稿では，強化学習を利用した自律的な攻撃の性質の解明を背景に，強化学習を用いた自律的な反射型 XSS 脆弱性の検査手法を提案する．提案手法では，予め与えた反射型 XSS 攻撃に使われる既知の文字列の断片の置換操作と，構文解析による状態観測を通じて，脆弱性の有無を検査するために脆弱性を攻撃する文字列（検査文字列）を構成する．検査文字列の構成にあたり，従来手法で必要だった人間の介在なしに自律的に検査文字列を構成する方法を強化学習で学習する．その結果，強化学習による最適な方策の獲得から，検査時に Web サイトへのリクエスト回数を大幅に減らすことが可能となる．評価実験を通じ，既存ツールとの比較において，提案手法は最も少ないリクエスト回数で脆弱性を発見できることを示す．

本稿の貢献は，以下である．

- 強化学習を用いた反射型 XSS 検査手法を実現するため，構文解析による状態観測と．文字列置換によるエージェントの行動，状態にもとづく報酬を定義する．
- 状態，行動，報酬にもとづき，強化学習を用いた反射型 XSS 検査アルゴリズムを提案する．
- 評価実験を通じ，既存ツールと比較して最も少ないリクエスト回数で反射型 XSS 脆弱性を発見することを示す．

## 2. 背景知識

### 2.1 強化学習

強化学習は，ある環境におかれたエージェントと呼ばれる行動主体が環境の状態を観測し，環境から得られる報酬を最大化するような方策を求めるアルゴリズムの一つである．ここでは，環境がマルコフ決定過程 $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathbf{P}, r)$ に従うと仮定する．ただし，$\mathcal{S}$ は環境から観測される状態の集合，$\mathcal{A}$ は環境が取り得る行動の集合，$\mathbf{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ は状態遷移確率関数，$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ は即時報酬関数である．強化学習では，時刻 $t$ において環境を観測したときに得られた状態 $s_t \in \mathcal{S}$ と，その時の報酬 $r_t$ をもとに，エージェントにおいて次にとるべき行動の意思決定を行い，実際に行動 $a_t \in \mathcal{A}$ を環境で実行する．環境はこの行動の結果によって内部状態が変化するため，次の時刻 $t + 1$ における状態 $s_{t+1}$ と報酬 $r_{t+1}$ を出力する．一連の行動（エピソード）が時刻 $T$ まで続くとき，割引率 $\gamma \in (0, 1]$ を用いて，割引累積報酬は $R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}$ と定義される．方策 $\pi(a \mid s)$ は，ある状態 $s \in \mathcal{S}$ において行動 $a \in \mathcal{A}$ をとる確率を示す．強化学習アルゴリズムを用いることで，環境とエージェントの相互作用を通じ，エージェントは方策 $\pi$ のもとで割引累積報酬 $R_t$ を最大化する最適な行動を選択するよう学習する．

```
<textarea>__USER_INPUT__</textarea>
```

(a) ユーザ入力値「__USER_INPUT__」の出力．

```
<textarea></textarea><script>alert(1);</script></textarea>
```

(b) 反射型XSS脆弱性により攻撃文字列が表示される例．

**図 1** XSS 脆弱性攻撃の例．
**Fig. 1** Example of an XSS attack.

### 2.2 クロスサイトスクリプティング（XSS）

クロスサイトスクリプティング（XSS）は，外部からの入力に応じて動的に出力される Web ページにおいて，JavaScript など任意のコードを実行可能な入力を受け付けてページを出力してしまう脆弱性，あるいはそれを利用した攻撃を指す．サニタイジングと呼ばれる無害化処理により攻撃を防ぐことができるが，セキュリティ意識や知識の欠如，あるいは予期しない原因により脆弱性が残存することがある．近年でも，XSS は Web サイトにおける代表的な脆弱性の一つである [1]．

**図 1** は XSS 脆弱性攻撃の例を示す．図 1 (a) は，Web サイトにおいてユーザの入力値 “__USER_INPUT__” が入力されたときのレスポンスである．ここでは，`textarea` タグ内に入力値が表示される．図 1 (b) は，攻撃者が XSS 脆弱性を狙った入力値（赤字下線部）を入力したときのレスポンスである．このとき，入力値先頭の`</textarea>`により，元より存在する開始タグと合わせて `textarea` タグが閉じられるため，その後の `script` タグが評価される．したがって，`script` タグ内のスクリプト `alert(1);` が動作する．Web サイトへの値の入力方法は，GET パラメータや POST リクエストなどが挙げられる．例えば GET パラ

メータに XSS 脆弱性を狙った不正な文字列を埋め込み，それを含む URL を配布することで，その URL にアクセスした Web サイト訪問者に不正なスクリプトを動作させることが可能となる．

XSS は一般に，反射型，格納型，DOM ベース型の 3 種類に分類される．このうち反射型は，URL パラメータなどに設定された値が Web サーバの処理を経由してそのまま Web ページ上に出力される種類である．これにより攻撃者は不正なスクリプトを埋め込んだリンクを偽メールや偽 Web サイトに記載し，ユーザがクリックするように誘導することが可能となる．ユーザがそのリンクをクリックすることで，攻撃者が設定した不正なスクリプトにより，情報の窃取やマルウェアのダウンロードなどが意図せず実行される危険性がある．本稿では，XSS の中でも特に攻撃が容易である反射型 XSS を対象とする．

### 2.3 強化学習による Web サイト攻撃

近年，強化学習を利用した Web サイトへの攻撃手法が提案されており [2], [3], [4], [5]，Web サイトの URL や特徴から公開ページ上にリンクの無い隠しファイルへのアクセスや，SQL インジェクションを試みる方法等が検討されている．さらに発展させ，ネットワークシステムのペネトレーションテストへの応用も提案されている [6].

文献 [5] では，強化学習を用いた反射型 XSS の攻撃文字列生成手法を提案している．典型的な反射型 XSS の攻撃文字列を 5 つのセクションに分割し，それぞれのセクションに対して攻撃に有効な文字列が配置されたか否かを人間が判定することで，強化学習による攻撃文字列の学習を実現する．文献 [5] の評価実験では，既存の反射型 XSS 脆弱性検知のためのオープンソースツールと比較して，脆弱性検知までのリクエスト数を大きく抑えることに成功している．その一方で，訓練時に人間が介在して状態観測を行う必要がある点，Q 学習を用いた検討のため状態，行動数の増加に伴い計算の収束性が悪化するという問題がある．

## 3. シナリオ

本稿の提案手法は，Web アプリケーションの脆弱性検査の方法として用いる防御者の観点と，反射型 XSS 脆弱性を突いて攻撃する攻撃者の観点がある．

**防御者の観点**: 防御者の観点では，提案手法を Web アプリケーションの脆弱性検査の方法として用いる．従来の反射型 XSS 脆弱性検査では，予め与えられた既知の検査文字列をもとに，検査対象の Web ページに対して総当たり的にリクエストを繰り返すことでその検査文字列を試行し，攻撃が成立するか否かを判定する．総当たり的な検査は検査項目を網羅的に確認できる一方で，Web ページにおける負荷や検査時間の増大といった問題が生じる．一般的な Web サーバにおけるサーバ負荷への影響は小さくとも，近年普及している IoT デバイスが備える簡易的な Web アプリケーションに対する検査においてはデバイスの負荷や検査時間を短縮することが求められる．また，Web Application Firewall (WAF) を有効にしている稼働中の環境等においては，脆弱性スキャンを実施することは上記の理由や WAF によるスキャン通信の遮断などにより，現実的でない．WAF そのものにおける脆弱性を含めて，稼働中の環境を対象とした脆弱性検査においては，少ない試行回数で攻撃が可能となる手法が望まれる．

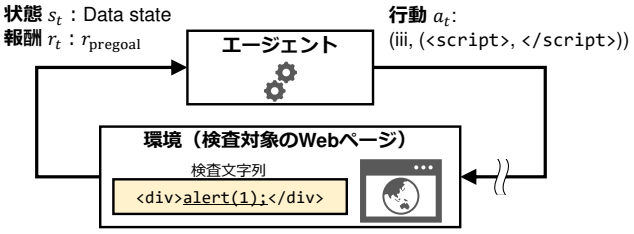


**図 2** 提案手法の概要.
**Fig. 2** Overview of the proposed method.

**攻撃者の観点**: 攻撃者の観点では，提案手法を利用して攻撃者は効率的な攻撃を試みる．攻撃者が Web アプリケーションを攻撃する場合，従来の脆弱性スキャンツールでは多数のリクエストが発生するため，攻撃を検知される可能性が高い．そのため，攻撃者はできるだけ少ないリクエスト数で脆弱性を発見できることが望ましい．

以上より，防御者及び攻撃者の観点において，できるだけ少ない試行回数で脆弱性を発見できる手法が望まれる場合がある．そこで本稿では，強化学習を用いてこれを実現する手法を提案する．

## 4. 提案手法

本稿では，強化学習を用いた反射型 XSS 検査手法を提案する．提案手法ではまず，反射型 XSS 攻撃に使われる既知の文字列の断片を与える．反射型 XSS 攻撃に使われる既知の文字列を知っていたとしても，検査対象となる Web ページに合わせて反射型 XSS 攻撃が成功するような文字列を構成するためには，その文字列を組み合わせる効果的な方法を習得していなければならない．そこで提案手法では，文字列を組み合わせる操作と，Web ページの構文解析にもとづく状態観測を通じて，強化学習により自律的な検査文字列の構成方法を学習する．

### 4.1 概要

反射型 XSS は，ユーザから任意の入力を受け付ける Web ページにおいて，ユーザからの入力（リクエスト）の一部をサーバの出力 (レスポンス) に反映させる機能における脆弱性である．攻撃者は，反射型 XSS の脆弱性を悪用し，Web ページのユーザである第三者に対して任意のスクリプ

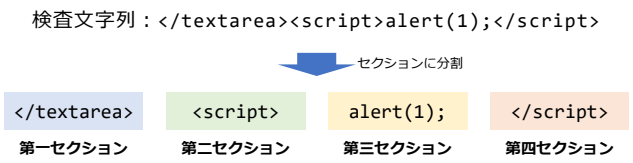


**図 3** 検査文字列のセクション．
**Fig. 3** Sections of a test string.

ト（ペイロード）を実行させることを意図する．攻撃者はペイロードの前後に文字列を付与することで，その文字列をリクエストの一部として入力した場合に，ペイロードが実行可能な状態でレスポンスとして出力される．このように，ペイロードが実行可能な状態となる，すなわち反射型 XSS 攻撃が成功するためには，ペイロードの前後に文字列を付与する操作が必要となる．そのような操作の結果得られる文字列全体を本稿では検査文字列と呼ぶ．反射型 XSS の検査においては，この検査文字列をいかに効率的に生成するかが重要な課題となっており，本稿ではその方法を強化学習で学習する．

2.1 節で述べたように，マルコフ決定過程に従う環境であれば強化学習アルゴリズムにより最適な方策 $\pi$ を獲得できる．したがって，タスクの習得には，行動 $\mathcal{A}$ と状態 $\mathcal{S}$，報酬 $r$ をいかに定めるかが重要である．**図 2** に提案手法の概要を示す．検査対象の Web ページとなる環境とエージェントとの間で，状態観測と報酬獲得，行動選択を繰り返す．提案手法においては，習得するタスクは，反射型 XSS 攻撃を成功させるための文字列を構成する操作である．次節以降では，反射型 XSS 攻撃を成功させるための文字列を構成する操作を習得するための行動空間 $\mathcal{A}$ における行動と，状態空間 $\mathcal{S}$ における状態，報酬 $r$ を説明し，最後に環境との相互作用を通じた反射型 XSS の検査手法を提案する．

### 4.2 行動

提案手法における重要なタスクは，検査文字列の生成である．強化学習の行動においては，検査文字列を構成するための文字列の操作から構成される．

**検査文字列のセクション**: 検査文字列の操作にあたり，まずは検査文字列を複数のセクションに区分する．ここで定義される 4 つのセクションにもとづき，指定されたセクションに対して文字列の置換や変換操作を行動として定義する．提案手法においては，検査文字列を以下の 4 つに区分する．

- **第一セクション**: 主として，直前のタグや構文解析上の状態（コンテキスト）を閉じる役割を担う．
- **第二セクション**: 主として，ペイロードのコンテキストを別のコンテキストに変えるための役割を担う．
- **第三セクション**: ペイロード文字列を当てはめる．
- **第四セクション**: 主として，ペイロードのコンテキストを閉じる役割を担う．

**図 3** は，検査文字列をセクションに区分する例を示す．検査文字列において，`alert(1);` はペイロードにあたり，第三セクションに該当する．`script` タグは，ペイロードを囲むように記述されており，開始タグ`<script>`は第二セクション，終了タグ`</script>`は第四セクションに該当する．`</textarea>`は `script` タグの直前に記述されており，検査文字列の外側に記述されている `textarea` タグを閉じる役割を果たすため，第一セクションに該当する．

上記のセクションをもとに，行動を定義する．行動 $a \in \mathcal{A}$ は，行動要素 (1) 操作対象のセクションと，行動要素 (2) 操作の内容のタプルから構成される．

**行動要素 (1) 操作対象のセクション**: 操作を行う対象とするセクションの組合せは，以下の行動種別 i–iv に示される 4 通りとする．

- **行動種別 i**: 第一セクションの文字列を置換する．主として，直前のコンテキストを閉じるための文字列操作を行う．
- **行動種別 ii**: 第二セクションの文字列を置換する．主として，ペイロードのコンテキストを別のコンテキストに変えるための文字列操作を行う．
- **行動種別 iii**: 第二セクションと第四セクションの文字列を，対応する文字列の組で置換する．主として，ペイロードの前後を操作することでペイロードのコンテキストを別のコンテキストに変えるための文字列操作を行う．
- **行動種別 iv**: 第四セクションの文字列を置換する．主として，ペイロードのコンテキストを閉じるための文字列操作を行う．

**行動要素 (2) 操作の内容**: 操作の内容としては，文字列の置換操作がある．行動要素 (1) 操作対象のセクションで指定されたセクションの文字列を，ここで指定した文字列に置換する．具体的な操作としては，まず準備として反射型 XSS における既知の検査文字列から，予めセクション毎に分割した断片を候補文字列として与える．このとき，候補文字列には空文字も含める．実際の操作においては，行動種別において選択された操作対象のセクションに該当する候補文字列を選択し，当該セクションの文字を置換する．このような操作により，エージェントが行動する毎に各セクション内の文字列が別の文字列へと置き換えられる．

文字列の置換操作に加え，文字列の変換操作を想定される．変換操作では，行動要素 (1) 操作対象のセクションにて指定されたセクション内の文字列に対し，ここで指定されたエンコード方式を用いてエンコードすることで，表記上で元の文字列と異なる文字列へと変換する．エンコード方式を変えることで，Web アプリケーションにおけるサニタイジング処理を回避できる可能性があるため，その脆弱性を狙った操作である．

**表 1** 行動の例.
**Table 1** Examples of the actions.

| | 行動 | | | 行動 | |
|---|---|---|---|---|---|
| | 種別 | 文字列 (またはその組) | | 種別 | 文字列 (またはその組) |
| 1 | i | (空文字) | 14 | ii | `onerror="` |
| 2 | i | `*/</style>` | 15 | iii | (空文字) |
| 3 | i | `</textarea>` | 16 | iii | (", ") |
| 4 | i | `-->` | 17 | iii | ('; , ;//) |
| 5 | i | "□ | 18 | iii | ("; , ;//) |
| 6 | i | '□ | 19 | iii | (0; , ;//) |
| 7 | i | 0□ | 20 | iii | (\r\n, ;//) |
| 8 | i | `'>` | 21 | iii | (*/, ;/*) |
| 9 | i | `">` | 22 | iii | ('); , ;//) |
| 10 | i | `">` | 23 | iii | (""); , ;//) |
| 11 | ii | (空文字) | 24 | iii | (`expression(`, `);`) |
| 12 | ii | `javascript:` | 25 | iii | (`<script>`, `</script>`) |
| 13 | ii | `onerror='` | 26 | iv | (空文字) |

* 表中の "(a, b)" は，2 つの文字列 "a"，"b" の組を表す．また，"□" は半角スペースを表す.

**表 2** 状態の例.
**Table 2** Examples of the states.

| | レスポンス | `alert(1);` の状態 |
|---|---|---|
| (a) | `<div>alert(1);</div>` | Data state |
| (b) | `<textarea>alert(1);</textarea>` | RCDATA state |
| (c) | `<script>alert(1);</script>` | Script data state |

**行動の構成**: 行動要素 (1) 操作対象のセクションと，行動要素 (2) 操作の内容との組合せによりタプルから行動を構成する．**表 1** に行動の例を示す．"種別" 列が行動要素 (1) 操作対象のセクションにおける行動種別を，"文字列 (またはその組)" 列が行動要素 (2) 操作の内容を示す．なお，表 1 の操作としては，文字列の置換操作のみを想定する．

表 1 の行動は，5 章の評価実験においても用いる．

### 4.3 状態

**構文解析にもとづく状態の設定**: 提案手法では，Web ページのレスポンスとして得られた文字列に対し構文解析を行い，ペイロードの文字列に当てはまる構文解析上の状態を，強化学習における状態として定義する．強化学習が観測する状態は，現在の状態からある行動を取ることで，確率的に次の状態に遷移するようなものであることが望ましい．また，反射型 XSS の検査における目的が検査文字列の生成であることから，検査文字列と合わせて考えたときにペイロードが実行可能であるかどうかを判別できる事象を状態として定義する．

状態の例を**表 2** に示す．ペイロードを "`alert(1);`" として，これを含む検査文字列を与えたときのレスポンスと，その時の構文解析上の状態の対応を示す．なお，ここでは HTML の構文解析上の状態として HTML5 の仕様[*1]を参照する．表 2 (a) では，ペイロードは `div` タグ内に出力されている．この場合，HTML5 の仕様としては "Data state" 状態となる．この時，文字列は画面上に表示される文字列となる．表 2 (b) では，ペイロードは `textarea` タグ内に出力されている．この場合，HTML5 の仕様としては "RCDATA state" 状態となり，`div` タグ内に表示される場合とは異なる状態となる．この時，文字列は一部の HTML タグを含んでいたとしても文字列として表示される．例えば，`script` タグを含んでいたとしてもその中のスクリプトは実行されず，単に表示されるのみである．表 2 (c) では，ペイロードは `script` タグ内に出力されている．この場合，HTML5 の仕様としては "Script data state" 状態となり，上記とはさらに異なる状態となる．この状態の文字列は Web ブラウザによりスクリプトとして解釈され，実行される．

以上に例示されるように，構文解析上の状態を参考にすることで，ペイロードがコンテンツとして表示される文字列か，スクリプトとして実行され得る文字列かなどの情報を得ることができる．提案手法では，このような構文解析上の状態をもとに状態空間 $\mathcal{S}$ を定義する．それにより，Web ページのレスポンスとして得られた文字列からペイロードの状態を観測する．

**状態の集合**: ここでは，状態をいくつかの集合に区分する．まず，一連の操作が完了する状態の集合をゴール状態集合として定める．提案手法では，強化学習を用いて検査文字列を効率的に生成するため，強化学習により一連のエピソードが完了した状態では，検査文字列中のペイロードは実行可能な状態となっているはずである．そのため，ゴール状態集合を以下のように定義する．

**定義 1** ゴール状態集合 $\mathcal{S}_\mathrm{g}$ は，ペイロードが実行可能である状態 $s$ の集合である．

提案手法における目標は，ゴール状態集合の要素となる状態を得ることである．

次に，ゴール状態以外の状態を細分化する．ここでは，必要な行動の数をもとに，ゴール状態集合までの距離 $d$ を以下のように定める．

**定義 2** 状態集合 $\mathcal{S}_d$ は，ゴール状態集合までの距離が $d$ である状態 $s_d$ の集合である．ただし $\mathcal{S}_0 = \mathcal{S}_\mathrm{g}$ として，$s_d$ は以下のように定められる．

- あるゴール状態 $s_0 \in \mathcal{S}_0$ において，ある行動 $a$ を適用して遷移した状態 $s$ に関して，$s \notin \mathcal{S}_0$ であれば，$s$ は距離を $d = 1$ として $\mathcal{S}_1$ の要素である．
- ゴール状態集合までの距離が $n$ である状態 $s_n \in \mathcal{S}_n$ において，ある行動 $a$ を適用して遷移した状態 $s$ に関して，$s \notin \mathcal{S}_i, 0 \leq i \leq n$ であれば，$s$ は距離を $d = n + 1$ として $\mathcal{S}_{n+1}$ の要素である．

ゴール状態集合までの距離 $d$ は，ゴール状態集合 $\mathcal{S}_\mathrm{g}$ から再帰的に得られる．4.2 節と本節で定める行動空間 $\mathcal{A}$ と状態空間 $\mathcal{S}$ より，任意の $d$ に関して $\mathcal{S}_d$ を得られる．

[*1] `https://html.spec.whatwg.org/`

### 4.4 報酬

報酬は，状態をもとに定義する．強化学習において，報酬の決め方は学習の収束性を左右する重要な要素の一つである．しかしながら，反射型 XSS を始めとするサイバー攻撃の試行においては，攻撃者の観点では攻撃の状態をつぶさに観測するのは難しい．そのため，状態を詳細に定義するのが難しく，攻撃の成否によってのみ報酬を定めざるを得ない状況にもなり得る．そこで，4.3 節で定義されるゴール状態集合とゴール状態集合までの距離をもとに，報酬を定める．本稿では，状態集合に対応して以下のような 3 種類の報酬を定める．

- $r_{\mathsf{goal}}$: ゴール状態 $s_{\mathsf{g}} \in \mathcal{S}_{\mathsf{g}}$ を得たときの報酬.
- $r_{\mathsf{pregoal}}$: ゴール状態集合までの距離 $d$ について，$d = 1$ となる状態 $s \in \mathcal{S}_1$ を得たときの報酬.
- $r_{\mathsf{other}}$: ゴール状態集合までの距離 $d$ について，$d \geq 2$ となる状態 $s \in \mathcal{S}_d$ を得たときの報酬.

表 1 に示される行動と表 2 に示される状態を用いて，状態と報酬との関係を例示する．表 2 において，(c) のレスポンスは “Script data state” 状態であり，ペイロード（下線部）が実行可能となる．したがって “Script data state” 状態はゴール状態集合 $\mathcal{S}_{\mathsf{g}}$ に分類され，この状態に到達したときの報酬は $r_{\mathsf{goal}}$ となる．次に，(a) の状態からは，表 1 に示される項番 25，(iii, (<script>, </script>)) の行動を適用することで，ペイロードが “Script data state” 状態となる．このように 1 つの行動でゴール状態となるため，(a) の “Data state” 状態はゴール状態までの距離が 1 であり，この状態に到達したときの報酬は $r_{\mathsf{pregoal}}$ となる．最後に，(b) の状態からは，表 1 に示される項番 3 及び項番 25 の行動を適用することで，ペイロードが “Script data state” 状態となる．このように 2 つの行動でゴール状態となるため，(b) の “RCDATA state” 状態はゴール状態までの距離が 2 であり，この状態に到達したときの報酬は $r_{\mathsf{other}}$ となる．

### 4.5 反射型 XSS 脆弱性検査

上記の状態，行動，報酬をもとに，強化学習アルゴリズムを用いてエージェントを訓練する．反射型 XSS 検査においては，上記で定義された状態はシステムの状態を完全に観測できる訳ではない．例えば，特定の動作に対するフィルタの有無を，少ない試行数から完全に判断するのは難しい．そこで，本稿では強化学習の結果得られる状態遷移確率を用いて次の行動を決定する．

反射型 XSS 検査のアルゴリズムを Algorithm 1 に示す．アルゴリズムでは，反射型 XSS の脆弱性に対応する検査文字列が見つかる場合，その文字列を返す．文字列が見つからない場合，null を返し，脆弱性を検知できなかった旨を知らせる．3 行目において，訓練済みの強化学習モデルで獲得している方策をもとに，状態 $s$ における行動 $a$ を取るときの確率 $p$ を得る．行動にあたり，14 行目でその状態

**Algorithm 1** 反射型 XSS 検査アルゴリズム

**Input:** 訓練済み強化学習モデル $\mathcal{M}$, 環境 $\mathcal{E}$, ペイロード文字列 $P$
**Output:** 検査文字列 $T$

```
 1: T ← P, H ← ∅, s ← 初期状態
 2: while s ∉ S_g do
 3:     L ← {(a, p) | p = π(a | s), a ∈ A}
 4:     L を p に関して降順に並べ替え
 5:     i ← 0
 6:     (a, p) ← L[i]
 7:     while (s, a) ∈ H and i < |L| do
 8:         i ← i + 1
 9:         (a, p) ← L[i]
10:     end while
11:     if i == |L| then
12:         return null
13:     end if
14:     H ← H ∪ (s, a)
15:     行動 a を実行し T を更新
16:     s ← 環境 E から状態を観測
17: end while
18: return T
```

$s$ と行動 $a$ を記録する．この確率 $p$ が高い順に行動 $a$ を試行し，15 行目において検査文字列 $T$ を更新する．そして，16 行目において行動 $a$ を試行した結果の状態を観測し，状態 $s$ を更新する．この時，検査文字列 $T$ が反射型 XSS への攻撃を成立させない場合や，Web ページが検査文字列 $T$ をフィルタしている場合には，状態 $s$ は変化しないか望ましくない状態となる．状態 $s$ が変化しない場合，次の試行において 7 行目から 10 行目に示されるループにより，状態 $s$ において前回よりも確率 $p$ が低い行動 $a$ を試行する．

## 5. 評価実験

提案手法をコンピュータ上に実装し，評価実験を行う．

### 5.1 実装

一連のプログラムは Python を用いて実装した．Web ページの構文解析には，Python から利用可能なオープンソースライブラリを使用している．構文解析ライブラリは Web ページを構成する言語ごとに使い分けており，HTML には Beautiful Soup[*2]，JavaScript には Esprima[*3]，CSS には cssutils[*4]をそれぞれ用いている．

強化学習を行う環境とエージェントは OpenAI Gym[*5]で動作する形式で実装した．強化学習アルゴリズムは stable-baselines3[*6]に含まれる PPO を使用した．なお，stable-baseline3 における強化学習アルゴリズムの行動と確率の一覧の取得には，既存クラスを拡張し独自に実装している．

[*2] https://www.crummy.com/software/BeautifulSoup/
[*3] https://github.com/Kronuz/esprima-python
[*4] https://github.com/jaraco/cssutils
[*5] http://gym.openai.com/
[*6] https://github.com/DLR-RM/stable-baselines3

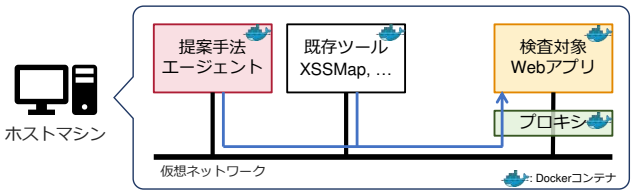


**図 4** 実験環境.
**Fig. 4** Experimental environment.

### 5.2 実験方法

強化学習の訓練時は，Web ページを HTML ファイルとしてローカル環境に書き出し，これを参照して訓練する．訓練に使用した HTML は 19 種類あり，WAVSEP [*7]に含まれる反射型 XSS 脆弱性と同種のものを再現した，WAVSEP とは異なる内容の Web ページが含まれる．

**図 4** は，評価時の実験環境の構成を示す．脆弱性検査の評価時は，Docker コンテナを用いて構築した仮想環境内に簡易的なサーバを構築し，WAVSEP を Python で再現実装したアプリケーション（検査対象 Web アプリ）を動作させる．ここには WAPSEP で用意される XSS 脆弱性を含む Web ページから 21 種類を実装している．このアプリケーションに対し，訓練済みのエージェント（提案手法エージェント）や既存ツールを用いて脆弱性を検査する．検査対象 Web アプリへはプロキシを介して通信し，このプロキシにおいてリクエスト数を計測する．

エージェントに用いる強化学習アルゴリズムとして，PPO [7] を用いる．提案手法における状態空間 $\mathcal{S}$ と行動空間 $\mathcal{A}$ はともに離散値を取るので，離散状態空間と離散行動空間に対応する強化学習アルゴリズムを選定する必要がある．この条件を満たす代表的なアルゴリズムとしては，PPO の他に Q 学習，DQN [8]，A2C [9] が挙げられる．これらのうち，比較的少ない試行回数で効率的，すなわちサンプリング効率の良いアルゴリズムである PPO を選択する．

PPO を用いた訓練では，訓練のステップ数を 50,000 回とした．また，学習率は 0.001，更新までのステップ数を 64 として設定した．その他のパラメータは，学習に使用したライブラリ stable-baselines3 の初期設定を用いる．

エージェントが取る行動は表 1 の通りである．観測される状態は表 2 を含む 23 種類としている．報酬は $r_{\text{goal}} = 10$，$r_{\text{pregoal}} = -t/2$，$r_{\text{other}} = -t$ として設定している．ただし，$t$ はエピソード内のステップ数である．

比較対象として，2020 年以降に更新されている反射型 XSS 脆弱性を検知する機能を有するオープンソースツールを利用した．これらのオープンソースツールは既知の反射型 XSS 脆弱性を検知するための検査文字列のリストや検査ロジックが予め実装されている．実験においては，検査対象となる Web ページに対し，提案手法及びオープンソースツールを用いて脆弱性検査を行う．このうち，脆弱性を検知するまでの最小のリクエスト数を算出し，1 ページあたりの平均最小リクエスト数を算出する．

[*7] https://github.com/sectooladdict/wavsep

**表 3** 比較に用いたオープンソースツール.
**Table 3** Open-source tools used in the comparison.

| ツール名 | URL |
|---|---|
| XSpear | https://github.com/hahwul/XSpear |
| XSSer | https://github.com/epsylon/xsser |
| XSSMap | https://github.com/Jewel591/xssmap |
| Wapiti | https://github.com/wapiti-scanner/wapiti |
| w3af | https://github.com/andresriancho/w3af |

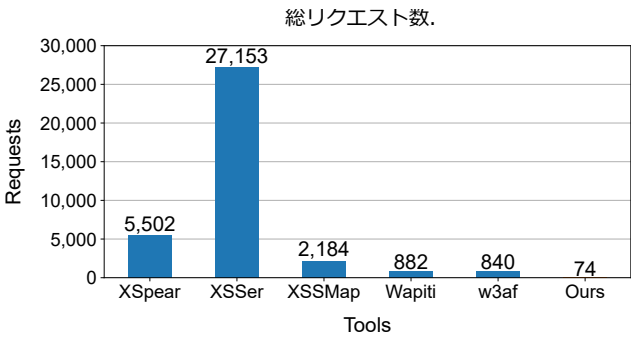


**図 5** 実験結果 （総リクエスト数）.
**Fig. 5** Experimental result (total requests).

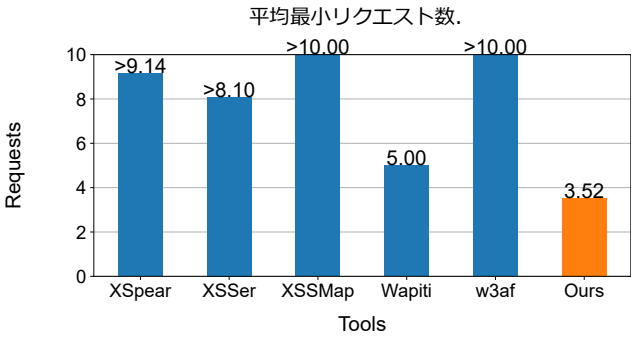


**図 6** 実験結果 （平均最小リクエスト数）.
**Fig. 6** Experimental result (average of minimum requests).

### 5.3 実験結果

提案手法の訓練では，50,000 ステップでローカル環境の 19 種類の Web ページを対象にエージェントを学習した．一連の訓練に要した時間は 2 分程度である．

**図 5** に，検査対象の Web サイトに対する総リクエスト数を示す．検査対象の Web ページは合計 21 種類あり，これらのページの検査のため各ツールがページをリクエストした回数を示す．図 5 より，オープンソースツールでは数百から数万回のリクエストを要する．これは，オープンソースツールは脆弱性を網羅的に検査するのが目的であり，各種ツールが備える検査文字列のリストを総当たり的に試行するためである．これにより各種ツールでは網羅的に脆弱性を検査することが可能となるが，このような方法は検査時間や Web サイトへの負荷が多くかかる．一方で，

**表 4** 文献 [10] との比較.
**Table 4** Comparison to the results in [10].

| | モデル 1 [10] | モデル 2 [10] | モデル 3 [10] | 提案手法 |
|---|---|---|---|---|
| 総リクエスト数 | 237.0 | 222.2 | 271.6 | 73.6 |

提案手法では合計 74 リクエストであった．これは，提案手法では少なくとも一つの脆弱性検知を目的として，効率的に検査文字列を生成するためである．提案手法では，検査文字列のリストや検査ロジックを実装することなく，強化学習による訓練だけで効率的な検査文字列の構成を実現している．実験結果より，オープンソースツールと比較して提案手法は目的を達成していることが示される．

**図 6** は，21 種類の Web ページにおいて，1 ページあたりで平均した最初に脆弱性を発見するまでの最小の試行回数を示す．なお，1 ページあたりの試行回数の上限は 10 回として算出し，21 種類の Web ページにおいて一つでも試行回数が 10 を超えたものには数値に '>' を付加している．図 6 に示されるように，6 個中 4 個のツールで '>' が示されており，少なくとも一つ以上の Web ページで脆弱性を発見するまでに 10 回以上のリクエストを要している．一方で，提案手法は平均 3.52 リクエストであり，平均的に最小の試行回数で脆弱性を発見することに成功している．以上の比較より，提案手法は既知の反射型 XSS 脆弱性の情報をもとに，強化学習を用いた方策の獲得により，検査文字列の効率的な生成に成功したことを示す．

**表 4** に，文献 [10] との比較を示す．文献 [10] は，学習環境に人間が介在する Human-in-the-loop を前提とした強化学習による脆弱性検知手法 [5] をもとに，PPO を用いて再評価した文献である．実験では文献 [10] と同じ 19 種類の Web ページを用い，乱数のシード値を変えて 5 回実験した際の，すべての Web ページ検査における総リクエスト数の平均を示す．表 4 より，提案手法の総リクエスト数が最小となり，文献 [10] の 3 つのモデルより大きくリクエスト数を減少させることに成功している．この結果から，提案手法におけるセクションの分割方法や状態の定義方法が，自律的な検査文字列の構成により適合していると言える．

### 5.4 課題

提案手法では強化学習の行動は既知の検査文字列の断片から構成されており，また方策の獲得には既知の脆弱性を含む Web ページを用いて学習している．そのため，既知の検査文字列では検知できない脆弱性や，学習する Web ページに含まれない脆弱性は，提案手法での検知が難しい可能性がある点が課題となる．

この課題を解決する手段の一つとして，文字列断片を自動的に生成する手法の導入が考えられる．例えば，系列データに対する機械学習アルゴリズムを利用した文字列断片の生成が挙げられる．提案手法と文字列断片の生成をどのように組み合わせるかは，今後の検討課題である．

## 6. おわりに

本稿では，強化学習を用いた自律的な反射型 XSS 脆弱性の検査手法を提案した．提案手法では，予め与えた反射型 XSS 攻撃に使われる既知の文字列の断片の置換操作と，構文解析による状態観測を通じて，従来手法で必要だった人間の介在なしに自律的に検査文字列を構成する方法を強化学習で学習する．その結果，強化学習による最適な方策の獲得から，検査時に Web サイトへのリクエスト回数を大幅に減らすことが可能となる．評価実験を通じ，既存ツールとの比較において，提案手法は最も少ないリクエスト回数で脆弱性を発見できることを示した．今後は，未知の脆弱性に対する効率的な検知が課題である．

### 参考文献


[1] 情報処理推進機構：ソフトウェア等の脆弱性関連情報に関する届出状況 [2020 年第 4 四半期（10 月〜12 月）]，情報処理推進機構（オンライン），入手先 ⟨https://www.ipa.go.jp/security/vuln/report/vuln2020q4.html⟩（参照 2021-08-19）.

[2] Zennaro, F. M. and Erdodi, L.: Modeling penetration testing with reinforcement learning using capture-the-flag challenges and tabular Q-Learning, (online), available from ⟨https://arxiv.org/abs/2005.12632⟩ (2020).

[3] Erdodi, L., Sommervoll, A. A. and Zennaro, F. M.: Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents, (online), available from ⟨http://arxiv.org/abs/2101.03118v1⟩ (2021).

[4] Fang, Y., Huang, C., Xu, Y. and Li, Y.: RLXSS: Optimizing XSS detection model to defend against adversarial attacks based on reinforcement learning, *Future Internet*, Vol. 11, No. 8 (2019).

[5] Caturano, F., Perrone, G. and Romano, S. P.: Discovering reflected cross-site scripting vulnerabilities using a multiobjective reinforcement learning environment, *Computers & Security*, Vol. 103, No. 102204 (2021).

[6] Hu, Z., Beuran, R. and Tan, Y.: Automated penetration testing using deep reinforcement learning, *Proc. EuroS&P Workshops*, pp. 2–10 (2020).

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O.: Proximal policy optimization algorithms, (online), available from ⟨http://arxiv.org/abs/1707.06347⟩ (2017).

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. A.: Playing atari with deep reinforcement learning, (online), available from ⟨http://arxiv.org/abs/1312.5602⟩ (2013).

[9] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. and Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning, *Proc. International Conference on Machine Learning*, pp. 1928–1937 (2016).

[10] 長谷川健人，披田野清良，清本晋作：反射型クロスサイトスクリプティングにおける強化学習を用いた攻撃の評価，情報科学技術フォーラム (2021).