L-003

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
Systematic Evaluation of Reflected Cross-Site Scripting Attacks Using Reinforcement Learning

長谷川健人 * 披田野清良 * 清本晋作 *
Kento Hasegawa Seira Hidano Shinsaku Kiyomoto

1. はじめに

機械学習技術の発展に伴い，AI セキュリティへの関心が高まっている．近年では，AI 技術の活用により，サイバー攻撃の高度化あるいは容易化の可能性が指摘されている．とりわけ，強化学習を利用した自律的なサイバー攻撃が注目されている．強化学習の応用により，攻撃者にとっては効率的な攻撃や労力の軽減を実現できる．防御者の観点では，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に対する防御や脆弱性検知への応用を検討するため，こうした自律的な攻撃の検討は重要な研究課題の 1 つである．

本稿では，強化学習を利用したサイバー攻撃の性質の評価を目的とする．攻撃の代表的な例として反射型クロスサイトスクリプティング (XSS) に着目する．強化学習により，攻撃者がユーザに実行させたいスクリプト (不正スクリプト) について，反射型 XSS の脆弱性を悪用する文字列 (攻撃文字列) を構成することを目指す．Human-in-the-loop を用いた先行研究をもとに，攻撃フレームワークとして 3 つのモデルを提案する．比較的新しい強化学習アルゴリズムを用いて実験を行い，攻撃モデルを評価する．

本稿の貢献を以下に示す．

- 先行研究をもとに強化学習を用いた反射型 XSS 攻撃フレームワークを 3 つのモデルとして提案する．
- いくつかの代表的な強化学習アルゴリズムを用いて，シミュレーションによる実験を通じて評価する．
- Human-in-the-loop を用いた手法の限界を明らかにし，自律性の高い攻撃モデルを実現する方針を示す．

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が環境の状態を観測し，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．一般に，マルコフ決定過程 $(\mathcal{S}, \mathcal{A}, T, r)$ を仮定して定式化される．$\mathcal{S}$ は状態空間，$\mathcal{A}$ は行動空間，$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ は状態遷移確率関数，$r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ は報酬関数である．ある状態 $s \in \mathcal{S}$ において，エージェントが出力する行動の確率分布 $\pi(\cdot \mid s)$ は方策と呼ばれる．時刻 t において環境から獲得する報酬を r_t とするとき，強化学習では時刻 H までの一連の行動 (エピソード) の期待累積報酬 $\mathbb{E}_\pi \left[\sum_{t=0}^{H} r_t \right]$ を最大化する方策 π の獲得を目的とする．

*株式会社 KDDI 総合研究所, KDDI Research, Inc.

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生成/表示される 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生成/表示してしまう脆弱性，あるいはそれを利用した攻撃を指す．サニタイジングなどの無害化処理により攻撃を防ぐことが可能であるが，セキュリティ意識や知識の欠如，あるいは予期せぬ原因で脆弱性が存在することがある．

いくつかある XSS 脆弱性の種類のうち，反射型はユーザから入力された値がそのまま Web ページ上に出力されるものである．この脆弱性を悪用し，例えば攻撃者は不正スクリプトを埋め込んだリンクをユーザにクリックさせ，ユーザにスクリプトを実行させることが可能となる．攻撃の実現性が高いことから，本稿では反射型 XSS に着目する．

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 [1, 2, 3]，Web サイトの URL や特徴から公開されていない隠しファイルへのアクセスを試みる検討や，SQL インジェクションを試みる方法が検討されている．文献 [3] では，Human-in-the-loop を用いた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，各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の補助と併せて判定することで，強化学習による攻撃文字列の学習を実現する．既存ツールと比較してより少ないリクエスト数で脆弱性検知が可能となる一方で，フレームワークに人間が依存する部分が多く存在する．本稿では，Human-in-the-loop を用いた手法の性能とその限界を調査し，自律性の高い攻撃モデルに向けた方針を示すのが目的である．

3. 強化学習による反射型 XSS の攻撃文字列生成

強化学習による反射型 XSS の攻撃文字列生成フレームワークを提案する．まず，先行研究 [3] をもとに，強化学習による反射型 XSS のための状態，行動空間を設計する．次に，攻撃フレームワークとして，エージェントの学習する内容が異なる 3 つのモデルを提案する．

3.1. 状態，行動，報酬の設定

2 章で述べたように，強化学習ではエージェントの方策 π の獲得が目的である．状態空間 $\mathcal{S}$ と行動空間 $\mathcal{A}$，報酬関数 r を先行研究 [3] にもとづき設定する．

* 赤字部分はユーザ入力

図 1: 反射型 XSS の例.

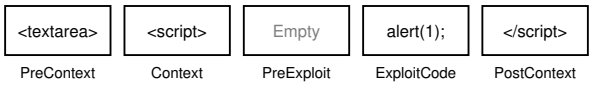

図 2: 攻撃文字列におけるセクション.

行動，状態: 反射型 XSS では，Web ページに入力された不正スクリプトの状態に応じて前後に HTML タグや引用符を挿入することで，Web サイトで不正スクリプトを実行可能にする．図 1 に攻撃文字列の例を示す．図 1 の攻撃文字列では，不正スクリプト ‘`alert(1);`’ を実行可能とするため `script` タグで囲む．ところが，`textarea` タグ内では `script` タグ内のスクリプトは通常実行されないため，`script` タグの前に閉じタグ ‘`</textarea>`’ を挿入する必要がある．これにより，`textarea` タグの外側となる `script` タグ内のスクリプトは Web ブラウザにより実行される．

図 2 に，攻撃文字列を 5 つのセクションに分割する例を示す．ExploitCode は，攻撃者が設定する不正スクリプトである．PreContext は，直前のタグを閉じる文字列である．Context と PostContext は，攻撃文字列を意図する位置に配置するための文字列である．PreExploit は，図 2 では空文字となるが，必要に応じて正しい攻撃文字列を生成するための文字列である．i 番目のセクション $(1 \leq i \leq 5)$ に対し，適用可能な文字列の集合 $(\mathcal{X}_i)$ を予め準備しておく．状態 $s \in \mathcal{S}$ は，セクション i に当てはめられた文字列を $x_i \in \mathcal{X}_i$ として，$(x_1, x_2, x_3, x_4, x_5)$ により示される．行動 $a \in \mathcal{A}$ は，セクションの位置 i とそこに当てはめる文字列 x_i から構成されるタプル (i, x_i) により示される．

報酬: ある時刻 t における報酬 r_t は，それぞれのセクションに対し目標の攻撃文字列と一致する文字列を選択した場合に正の値 r^+ とする．また，全く異なる文字列を選択した場合は負の値 $r^-(\ll -1)$ とする．

3.2. 攻撃フレームワーク

強化学習を用いた反射型 XSS 攻撃フレームワークを説明する．フレームワークは，学習フェーズとテストフェーズから構成される．学習フェーズでは，エージェントを用いて方策を学習する．テストフェーズでは，学習したエージェントを用いて攻撃文字列を構成し、攻撃を試行する．先行研究 [3] での Human-in-the-loop を用いた手法をもとに，本稿では 3 つのモデルを提案する．図 3 に 3 つのモデルの概要を示す．

3.2.1. モデル 1: 目標学習型

モデル 1 では，目標ごとにエージェントを学習する．ここで目標 $g = (x_1^{(g)}, g_2^{(g)}, g_3^{(g)}, g_4^{(g)}, g_5^{(g)})$ とは，反射型 XSS の攻撃が成功する攻撃文字列の状態 s である．

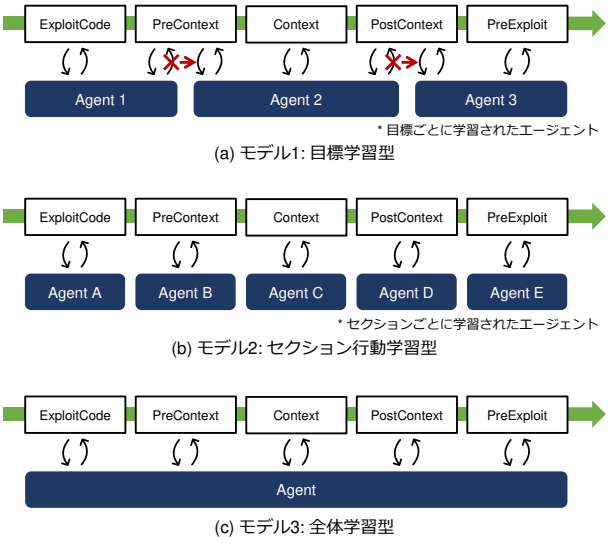

図 3: 提案モデルのイメージ図.

表 1 (4 章で説明) の各行 (#) に対応する．先行研究 [3] は，モデル 1 において Q 学習を利用したものである．

学習フェーズ: j 番目の攻撃文字列を目標 g_j と定め，これに対応するエージェント A_j を学習する．学習では，A_j を用いてランダムに行動を選択し，行動後の状態 s が目標 g_j からどの程度離れているかに応じて報酬を得る．この操作を全ての j に対して行う．

テストフェーズ: 攻撃文字列の生成は，状態の初期値としてすべてが空文字の状態から始める．図 3(a) に概要を示す．攻撃文字列は，ExploitCode，PreContext，Context，PostContext，PreExploit の順にセクションを選択し，構成する．まず，攻撃文字列を目標 g_j と定め，それに対応する A_j を用いる．現在の状態から，A_j を用いて現在のセクションで取り得る行動とその確率を得る．確率の高い行動から順に行い攻撃文字列を構成し，Human-in-the-loop における観測者 (例えば人間) が環境を観測する．観測者が目標と合致すると判断した場合，次のセクションに対して同様の操作を行う．いずれの行動でも目標と合致しないと判断した場合，$j+1$ 番目の攻撃文字列を目標とし，同様の操作を全ての j に対して繰り返す．

3.2.2. モデル 2: セクション行動学習型

モデル 2 では，各セクション毎に現在の状態から次の最適な行動を選択するエージェントを学習する．図 3(b) に示されるように，各セクションに対応する 5 つのエージェントを学習する．

学習フェーズ: モデル 1 と同じセクションの順序で，j 番目の目標 g_j について以下の手順により学習する．まず，k 番目のセクションにおける行動をエージェント A_k で学習する．次に，$x_u^{(g_j)} = x_u (1 \leq u \leq k)$ と仮定し，$k+1$ 番目のセクションを学習する．この操作を各目標 g_j に対して $k = 5$ まで繰り返す．

テストフェーズ: 攻撃文字列の生成は，状態の初期値としてすべてが空文字の状態から始め，学習フェー

ズと同様にセクションを選択する．k 番目のセクションにおいて，対応するエージェント A_k を用いて次に取り得る行動とその確率を得る．確率の高いものから順に行動に従い攻撃文字列を構成し，観測者が目標と合致すると判断した場合，$k+1$ 番目のセクションに対して同様の操作を行う．

3.2.3. モデル 3: 全体学習型

モデル 3 では，ただ 1 つのエージェントを用いて学習する．図 3(c) に概要を示す．モデル 3 では，ただ 1 つのエージェントを用いることで，Human-in-the-loop によるフレームワークにおいて可能な限り自律性の高い攻撃モデルを構築する．

学習フェーズ: 1 つのエージェント A を用いる．j 番目の攻撃文字列を目標 g_j として設定した上で，ランダムに行動を繰り返す．目標の攻撃文字列を構成し終えると，次の攻撃文字列を目標として同様の操作を行う．ただし，すべての目標 g_j を学習するため，目標文字列を構成し終えない場合は一定のステップ数で行動を打ち切る．

テストフェーズ: 学習フェーズで学習したエージェント A だけを用いて，モデル 2 における攻撃フェーズと同様の操作を行う．

4. 評価実験

3 章で提案した 3 つのモデルを用いて評価し，結果をもとに強化学習を用いた攻撃を考察する．

4.1. 実験環境

実験には Intel Core i7-9700 CPU，32GB メモリを搭載するコンピュータを使用した．実装には Python (3.8) と，ライブラリ Gym [†]を利用する．Q 学習以外の強化学習アルゴリズムには Python のライブラリ stable-baselines [‡]で提供される実装を用いる．

4.2. 実験方法

実験では，WAVSEP [§]で用意される XSS 脆弱性を含む Web ページから 19 種類 (文献 [3] と同様) を選択し，それぞれに対応する攻撃文字列を表 1 として定める．各攻撃文字列を順に目標として定め，学習したモデルで攻撃を行い評価する．実験はシミュレーションで行う．モデル内の観測者は，エージェントの行動の結果得られた文字列について，現在の目標と合致するかを機械的に判定するものとする．実験は 5 回行い，平均的な性能を確認する．

強化学習アルゴリズムは，Q 学習，DQN [4]，A2C [5] および PPO [6] の 4 種類を用いる．Q 学習では割引率を 0.98 とし，学習率を 1.0 から学習の進行にしたがって減衰するよう設定する．DQN，A2C および PPO では，ライブラリの初期値を利用する．

[†]https://github.com/openai/gym
[‡]https://github.com/hill-a/stable-baselines
[§]https://github.com/sectooladdict/wavsep

表 1: 攻撃目標とその具体例.

#	PreContext	Context	PreExploit	ExploitCode	PostContext
1		<script>		alert(1);	</script>
2	</textarea>	<script>		alert(1);	</script>
3	-->	<script>		alert(1);	</script>
4		javascript:		alert(1);	
5	"	onerror=	"	alert(1);	
6	'	onerror=	'	alert(1);	
7		onerror=	"	alert(1);	"
8	'>	<script>		alert(1);	</script>
9	">	<script>		alert(1);	</script>
10	"">	<script>		alert(1);	</script>
11	';			alert(1);	//
12	";			alert(1);	//
13	;			alert(1);	//
14			%0d%0a	alert(1);	
15	*/		%0d%0a	alert(1);	/*
16	'){}			alert(1);	//
17	"){}			alert(1);	//
18	</style>	<script>		alert(1);	</script>
19	*/</style>	<script>		alert(1);	</script>
種類	16	4	4	2	5

*種類には空文字を含む.

4.3. 実験結果

モデル 1–3 による，Q 学習，DQN，A2C，PPO の 4 種類の強化学習アルゴリズムを用いた実験の結果を表 2 に示す．(平均) 学習時間は，1 回の実験で学習に要した時間を，使用するエージェント数 (モデル 1: 19, モデル 2: 5，モデル 3: 1) で割った値である．総ステップ数は，学習した 19 種類の目標に対し，正解の攻撃文字列を生成するために要したステップ数の総計を示す．

4.3.1. モデル 1

モデル 1 では，各目標に対して 10,000 ステップで攻撃文字列を学習した．モデル 1 は，PreContext まで決定できればエージェントにより効果的に攻撃文字列を構成できるはずである．4 種類の強化学習アルゴリズムのうち，A2C や PPO では，短時間の学習かつ比較的少ない回数で攻撃を実現できた．

モデル 1 では，攻撃文字列を追加する場合，1 つのモデルだけを学習すれば良い．1 つあたりの学習時間は高々数十秒であり，追加のコストは他のモデルと比べて低い．

4.3.2. モデル 2

モデル 2 では，各ステップ毎に 10,000 ステップとして学習した．モデル 2 は，各ステップを確率的に選択するはずである．学習する状態数が少ないため，他のモデルと比較して少ないステップ数で攻撃を実現出来る．特に DQN，A2C，PPO では他のモデルと比較して最小の総ステップ数であり，学習した中で可能な限り良い行動を選択できていると言える．

4.3.3. モデル 3

モデル 3 では，50,000 ステップとして学習した．モデル 3 においては PPO の学習時間が少なく，総ステップ数も最小で済む結果となった．状態・行動数が他よりも多いため，価値ベースのアルゴリズムである Q 学習や DQN よりも，A2C や PPO が有効であることが確認できた．

表 2: モデル 1–3 の実験結果 (5 回の平均値 (*s.d.* は標準偏差)).

	モデル 1		モデル 2		モデル 3	
	平均学習時間 [s]	総ステップ数	平均学習時間 [s]	総ステップ数	学習時間 [s]	総ステップ数
Q 学習	14.74 (*s.d.* 0.03)	266.6 (*s.d.* 4.2)	4.11 (*s.d.* 0.03)	302.4 (*s.d.* 6.1)	65.45 (*s.d.* 0.85)	312.6 (*s.d.* 16.2)
DQN	30.98 (*s.d.* 0.51)	298.4 (*s.d.* 4.2)	31.67 (*s.d.* 1.06)	221.4 (*s.d.* 0.8)	165.95 (*s.d.* 5.06)	304.4 (*s.d.* 29.4)
A2C	19.86 (*s.d.* 0.16)	234.6 (*s.d.* 2.0)	19.90 (*s.d.* 0.24)	221.0 (*s.d.* 0.0)	105.48 (*s.d.* 2.69)	280.6 (*s.d.* 11.5)
PPO	5.26 (*s.d.* 0.02)	237.0 (*s.d.* 2.5)	6.26 (*s.d.* 0.04)	222.2 (*s.d.* 2.4)	27.11 (*s.d.* 0.22)	271.6 (*s.d.* 2.9)

表 3: モデルの比較.

	モデル 1	モデル 2	モデル 3
攻撃ステップ数	◐	●	–
追加学習のコスト	●	◐	–
リソース	–	◐	●
自律的な学習への応用	–	–	●

●: 良い / ◐: 中間 / –: 悪い

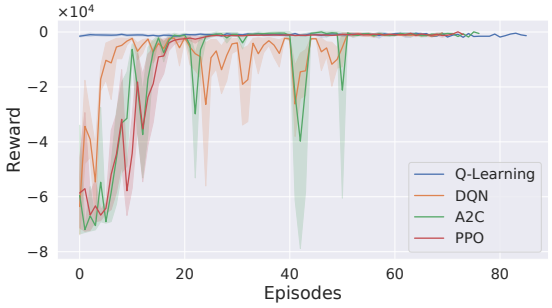

図 4: モデル 3 学習時の累積報酬値.

4.4. モデルおよびエージェント間の比較

モデル間の比較を表 3 に示す．それぞれの項目に対して，各モデルは一長一短である．攻撃ステップ数は，表 2 の総ステップ数よりモデル 2 が最良となる．追加学習のコストは，表 2 の平均学習時間よりモデル 1 が最良となる．リソースは，ここでは必要なエージェント数の観点で考慮すると，エージェントが 1 つのモデル 3 が最良となる．自律的な学習への応用の観点では，単一のエージェントを用いて学習が可能なモデル 3 が最も適している．

状態・行動空間が 2.1 節に示すマルコフ決定過程に従えば，自律性の高いモデル 3 であっても効率的な学習が可能なはずである．しかしながら，モデル 3 はどの強化学習アルゴリズムでも総ステップ数の観点で最悪である．これは，状態・行動空間がマルコフ決定過程に従わないため，方策 π を効果的に学習出来ないためと考えられる．例えば PreContext セクションを選ぶ際，前の状態はどの攻撃目標においても同一の ExploitCode(`alert(1);`) となるため，行動を選択する確率分布は一様となる．先行研究 [3] では Human-in-the-loop を前提としたアルゴリズムのため，今回利用した状態・行動空間で機能するが，自律的な攻撃においては状態・行動空間の再設計が必要となる．具体的には，環境の状態をエージェントが正しく認識するため，観測者を介すことなく，不正スクリプトの出力位置をマルコフ決定過程に従う状態として表現することが挙げられる．

次に，モデル内でのエージェント間を比較する．図 4 に，モデル 3 で学習した際の各エピソード毎の報酬を示す．横軸は試行したエピソード数，縦軸は累積報酬，線の上下の領域は信頼区間を示す．Q 学習 (Q-Learning) は安定して高い報酬を得るように見えるが，表 2 において総ステップ数が最も多く，効率的な学習は難しい．一方，PPO では 20 エピソード以降は比較的安定した値を得られていることが分かる．また，表 2 においても安定的な結果を得ていることから，モデル 3 において比較した中で最も効率的に学習できていると言える．

5. まとめ

本稿では，反射型 XSS を対象として，強化学習を用いた攻撃文字列の生成手法に着目し，複数の強化学習アルゴリズムを用いて評価した．評価を通じ，PPO などのアルゴリズムが Human-in-the-loop を用いたフレームワークにおいても有効に機能することを確認した．しかし，Human-in-the-loop の手法にもとづく応用だけでは自律性の高い学習が困難であることも明らかとなった．本稿では，自律性の高い学習に向けて，マルコフ性を満たす状態・行動空間の設計が必要である方針を示した．今後は，自律的な反射型 XSS 攻撃の実現に向けて状態，行動，報酬の設計を行い，脆弱性検知への応用検討を進める．

参考文献

[1] F. M. Zennaro and L. Erdodi, “Modeling penetration testing with reinforcement learning using capture-the-flagchallenges and tabular q-learning,” 2020. [Online]. Available: https://arxiv.org/abs/2005.12632

[2] L. Erdodi, A. A. Sommervoll, and F. M. Zennaro, “Simulating sql injection vulnerability exploitation using q-learning reinforcement learning agents,” 2021. [Online]. Available: http://arxiv.org/abs/2101.03118v1

[3] F. Caturano, G. Perrone, and S. P. Romano, “Discovering reflected cross-site scripting vulnerabilities using a multiobjective reinforcement learning environment,” *Computers & Security*, vol. 103, no. 102204, 2021.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

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

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017. [Online]. Available: http://arxiv.org/abs/1707.06347