Computer Security Symposium 2021
26 - 29 October 2021

深層強化学習を用いたWebアプリの脆弱性検査のためのAIエージェント

谷崎 俊介[1,a)]　廣友 雅徳[1,b)]　白石 善明[2]

概要：Web アプリの脆弱性を突いたサイバー攻撃は脅威であり，Web アプリの脆弱性検査が重要である．脆弱性検査用のツールとして OWASP ZAP や Burp Suite などのスキャナーがあるが，これらのツールの利用には専門知識を必要とし，人手に頼る部分が多い．本稿では Web アプリの脆弱性検査の自動化を目指し，SQL インジェクション攻撃を自動的に行う強化学習のエージェントを提案する．強化学習のアルゴリズムとして Deep Q-learning を利用した．エージェントは Web アプリに存在する脆弱性のパターンを自ら学び，攻撃手法の最適化を続ける．エージェントの学習には，SQL インジェクション攻撃に成功すると特定の文字列を含む HTTP レスポンスを返す，Capture The Flag (CTF) 型の Web アプリを使用した．エージェントの学習が進むと，攻撃成功に必要な HTTP リクエストの送信回数は大幅に減少し，エージェントは攻撃に成功する可能性が高い HTTP リクエストを優先的に送信するようになった．

キーワード：SQL インジェクション，強化学習，脆弱性検査，Web アプリ，CTF

AI Agent Based on Deep Reinforcement Learning for Web Application vulnerabilities

Shunsuke Tanizaki[1,a)]　Masanori Hirotomo[1,b)]　Yoshiaki Shiraishi[2]

Abstract: Cyber attacks that exploit vulnerabilities in web applications are a threat, and vulnerability analysis of web applications is important. There are vulnerability scanners such as OWASP ZAP and Burp Suite for web applications, but the use of these tools requires specialized knowledge and requires a lot of manual labor. In this paper, we propose the reinforcement learning agent that automatically performs SQL injection attacks on web applications. We used Deep Q-learning for reinforcement learning algorithm. The agent learns the vulnerability patterns in web applications by itself and continues to optimize its own attack method. For agent learning, we developed CTF web applications that returns a flag when the SQL injection attack is successful. As the learning of the agent progressed, the number of HTTP requests sent for a successful attack decreased significantly, and the agent began to preferentially send HTTP requests that are likely to succeed in the attack.

Keywords: SQL injection, reinforcement learning, vulnerability analysis, web applications, CTF

1. はじめに

Web アプリの脆弱性を突いたサイバー攻撃は脅威であり，Web の登場から現在に至るまで，Web アプリの提供者や利用者に深刻な被害をもたらしている．SQL インジェクション攻撃などによる情報漏えいは，さらなるサイバー攻撃やサイバー犯罪に利用される可能性があるため，特に危険である．これらの被害を未然に防ぐためには，Web アプリの脆弱性検査が重要である．

Web アプリの脆弱性検査手法には，OWASP ZAP[1] や

[1] 佐賀大学理工学部
Faculty of Science and Engineering, Saga University
[2] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
[a)] tanizaki.shunsuke@hrtmlab.org
[b)] hirotomo@cc.saga-u.ac.jp

Burp Suite[2] などのスキャナーを用いた手法や，ソースコード解析などの手法が存在する．スキャナーを用いた脆弱性検査手法では，スキャナーの利用者に専門的なスキルが要求され，専門的なスキルを持っていたとしても手動で行わなければならない部分が多く，手間がかかる傾向がある．自動的なソースコード解析では，実際に攻撃を試して本当に脆弱性が含まれているのかどうかを確認しないため，間違った警告を行ったり，該当箇所が本当に脆弱なのかどうかを人が判断する必要があったりなどの問題がある．

本稿では，専門知識や人手を必要としない Web アプリの脆弱性検査を実現するために，Web アプリに対する攻撃を自動的に行う強化学習のエージェントを提案する．エージェントの実装には深層強化学習を利用し，学習アルゴリズムとして Deep Q-learning を採用した．Web アプリにある脆弱性のパターンを自ら学ぶエージェントは，攻撃手法を最適化し続ける．学習には，SQL インジェクション攻撃に成功すると特定の文字列を含む HTTP レスポンスを返す，CTF 型の Web アプリを使用した．エージェントの性能評価として，攻撃成功に必要な HTTP リクエストの送信回数や，実際に送信した HTTP リクエストの内容を確認する．最後に，提案手法の課題について考察する．

2. 関連研究

2.1 サイバー攻撃の自動化についての関連研究

バイナリファイルや Web アプリなどの，ソフトウェアに対する攻撃を自動化する研究分野があり，Automatic Exploit Generation (AEG) [3] と呼ばれている．AEG には，Symbolic Execution[4] と呼ばれるソースコード解析の技術や，ファジングが使われており，ソフトウェアの脆弱性発見やエクスプロイトの組み立てが全て自動的に行われる．AEG のシステムとして，コントロールハイジャッキング攻撃を行う MAYHEM[5] や，SQL インジェクション攻撃と XSS 攻撃を行う NAVEX[6] などがある．

2016 年 8 月，AEG の技術を競う Cyber Grand Challenge (CGC) が DARPA によって開催された [7]．CGC では，すべての参加チームにサーバーが 1 台ずつ与えられ，他チームのサーバーで動くソフトウェアへの攻撃と，自チームのサーバーで動くソフトウェアの脆弱性の修正が，すべてコンピュータによって行われた．攻撃側の技術である AEG に加え，バイナリファイルに含まれる脆弱性の修正も自動的に行われた CGC は，サイバー攻撃やソフトウェア修正の自動化に関する研究が注目を浴びるきっかけとなった．

Symbolic Execution やファジングを用いた現在の AEG では，対象とする脆弱性の発見率や攻撃の精度は高いものの，未知の脆弱性に対応できない．また，従来の AEG では一度開発を終えると性能が固定化され，学習によって性能を向上させていくことができない．

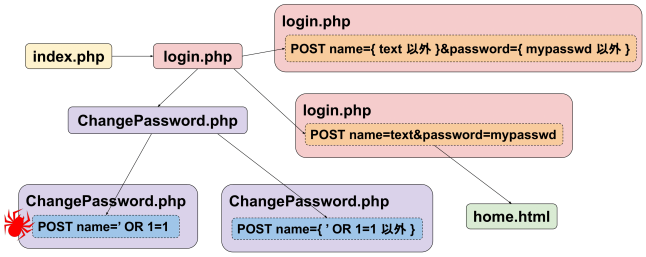

図 1 Web アプリの例
Fig. 1 Web application example

2.2 強化学習についての関連研究

強化学習を用いたシステムには様々なものが存在するが，中でも AlphaGo[8] が有名である．2015 年 10 月，Google DeepMind によって開発されたコンピュータ囲碁プログラム AlphaGo は，ハンディキャップなしでプロ囲碁棋士と勝負し，勝利した．ハンディキャップなしでプロ棋士に勝利したコンピュータ囲碁プログラムは AlphaGo が世界初であり，AlphaGo には強化学習が用いられている．

AlphaGo のようなゲームをプレイする強化学習のエージェント以外にも，自動運転や制御系システムなど，多くの分野で強化学習は利用されている．

強化学習が適用できる分野として，サイバー攻撃やサイバー犯罪も例外ではなく，AI を悪用したサイバー攻撃が今後の脅威になる可能性がある．しかし，ソフトウェアの脆弱性検査などのサイバーセキュリティ分野に強化学習を適用した研究例はまだ少なく，未解決課題が多く残っている．

3. Web アプリへの攻撃を行うエージェント

本章では SQL インジェクション攻撃を行うエージェントを提案する．なお，以下で述べる提案手法は SQL インジェクション攻撃以外の攻撃にも適用できる．具体的には，攻撃が成功したときに特定の文字列 (フラグ) を返すような CTF 型の Web アプリが用意できる攻撃である場合，対応が可能である．提案手法は変えずに，用意する CTF 型の Web アプリを変えるだけで，XSS 攻撃や OS コマンドインジェクション攻撃などにも対応可能である．

本章では，エージェントの攻撃手順について述べ，説明に図 1 と図 2 を用いる．図 1 は，提案手法が扱う Web アプリを有向グラフで示したものである．図 2 は，エージェントの攻撃手順を示した図である．

エージェントの攻撃手順は以下の 5 つの Step から成る．

Step 1 : 攻撃対象の Web アプリの URL を指定

提案手法が扱う Web アプリには，起点となる Web ページが存在する．図 1 の Web アプリでは，「index.php」が起点となる Web ページである．攻撃の開始時は，攻撃対象の Web アプリの URL をエージェントに指定する．例えば，図 2 にあるように「http://www.example.com/index.php」をエージェン

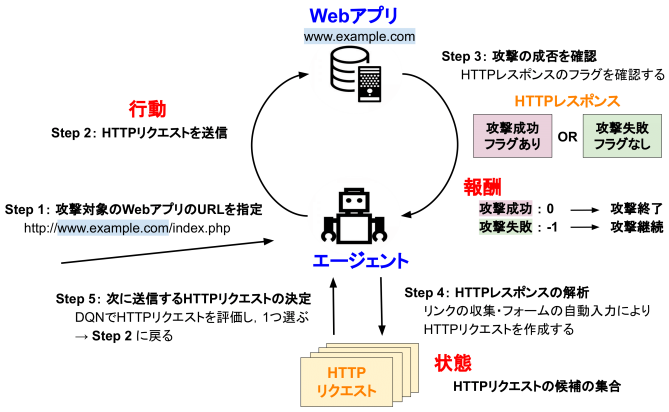

図 2 エージェントの攻撃手順
Fig. 2 Attack procedure

トに与える．エージェントは与えられた URL を使って，GET メソッドの HTTP リクエストを生成し，Step 2 へ進む．

Step 2： HTTP リクエストを送信

HTTP リクエストを送信し，HTTP レスポンスを受け取る．Step 3 へ進む．

Step 3： 攻撃の成否を確認

HTTP レスポンスにフラグが含まれているかを確認する．図 1 の Web アプリでは，POST メソッドで「ChangePassword.php」に対し，「name=’ OR 1=1」を送信すると，攻撃に成功する．よって，Step 2 でこの HTTP リクエストを送信していた場合，HTTP レスポンスはフラグを含んでいる．フラグが含まれていた場合は攻撃成功であり，報酬 0 を受け取って攻撃を終了する．フラグが含まれていない場合は攻撃失敗であり，報酬 −1 を受け取って Step 4 へ進む．報酬設計については 4 章で述べる．

Step 4： HTTP レスポンスの解析

HTTP レスポンスの解析を行う．具体的には，HTTP レスポンス内に含まれるリンクの収集と，フォームへの自動入力を行い，HTTP リクエストを作成する．図 1 の有向グラフでは，Web アプリに対して送信できる HTTP リクエストをノードで表現している．HTTP リクエスト A を始点，HTTP リクエスト B を終点とするエッジが存在する場合，HTTP リクエスト A を送信したときに返ってくる HTTP レスポンスを解析した結果，HTTP リクエスト B が作成できることを意味する．以降の説明では，図 1 の Web アプリにおいて，Step 2 で「login.php」に対して GET メソッドの HTTP リクエストを送信し，その結果として返ってきた HTTP レスポンスを解析する場合を想定する．「login.php」には「ChangePassword.php」へのリンクと，2 つのパラメータ「name」と「password」を持つログインフォームが存在する．よって，「ChangePassword.php」への HTTP リクエスト (GET メソッド) と，2 つのパラメータ「name」と「password」を持つ HTTP リクエスト (POST メソッド) が作成される．なお，作成する HTTP リクエストのメソッドは，GET または POST のいずれかとし，パラメータ数の上限を 2 とする．また，エージェントは攻撃に成功するまで 1 つの Web アプリ内を巡回し続けるため，外部ドメインに対する HTTP リクエストは作成しない．パラメータに渡す値は，以下の 8 つの文字列を含むリストを使用する．

1. ’OR 1=1#
2. ”OR 1=1#
3. 1 OR 1=1#
4. admin’#
5. admin”#
6. admin
7. test
8. mypasswd

2 つのパラメータを持つ HTTP リクエストを上記のリストを用いて作成する場合，$8 \times 8 = 64$ 個の HTTP リクエストの候補が作成される．つまり，GET メソッドの HTTP リクエストが 1 つ，POST メソッドの HTTP リクエストが 64 個，合計 65 個の HTTP リクエストが作成される．作成した HTTP リクエストを，次に送信する HTTP リクエストの候補に加え，Step 5 へ進む．

Step 5： 次に送信する HTTP リクエストの決定

HTTP リクエストの候補から，次に送信する HTTP リクエストを 1 つ選ぶ．次に送信する HTTP リクエストを選ぶ際，Deep Q-network (DQN) を使う．DQN に HTTP リクエストの候補をすべて入力し，HTTP リクエストに優先順位を付ける．優先順位の高いものほど，攻撃に成功する，あるいは攻撃成功に繋がる可能性が高いと判断された HTTP リクエストである．攻撃成功に繋がる HTTP リクエストとは，直接的な攻撃となる HTTP リクエストではないが，脆弱性を含むページに遷移するために必要な HTTP リクエストのことである．図 1 の Web アプリでは，POST メソッドで「ChangePassword.php」に対し，「name=’ OR 1=1」を送信すると，攻撃に成功する．この HTTP リクエストのように，パラメータを渡す HTTP リクエストは直接的な攻撃となる HTTP リクエストの例である．一方，GET メソッドによる「ChangePassword.php」への HTTP リクエストなどは，パラメータを渡さない HTTP リクエストであり，直接的な攻撃となる HTTP リクエストではないため，「攻撃成功に繋がる HTTP リクエスト」である．エージェントは DQN を使い，最も優先順位の高い HTTP リクエストを選んだあと，Step 2 へ戻る．エージェントは攻撃に成功するまで，

Step 2 ～ 5 を繰り返す．

4. 強化学習を用いたHTTPリクエストの選択

提案手法では，Web アプリへの攻撃プロセスを有限マルコフ決定過程として捉え，Web アプリへの攻撃に特化した強化学習のエージェントを実装した．使用した強化学習のアルゴリズムは，Deep Q-learning である．

有限マルコフ決定過程は 4 つの要素$<S, A, T, R>$で表される．S, A, T, R はそれぞれ，状態の有限集合，行動の有限集合，遷移関数，報酬関数である．以下に提案手法が扱う有限マルコフ決定過程の S，A，T，R の定義を示す．

- $S = \{s_1, s_2, \ldots, s_N\}$： HTTP リクエストの候補の集合
- $A = \{a_1, a_2, \ldots, a_M\}$： すべての送信可能な HTTP リクエストの集合
- $T : S \times A \times S \to \{0, 1\}$：遷移関数
- $R : S \times A \times S \to \{-1, 0\}$：報酬関数

状態の有限集合 S は，HTTP リクエストの候補の集合であり，エージェントが次に送信する HTTP リクエストを決定する際に用いる．エージェントは，HTTP リクエストを送信した結果として返ってくる HTTP レスポンスを解析し，HTTP リクエストを作成する．作成された HTTP リクエストは集合 S に要素として追加される．HTTP リクエストの候補から 1 つ選んで送信した場合，すなわち集合 S から 1 つの要素 s_0 を選んだ場合，要素 s_0 は集合 S から除外される．集合 S は有限集合である．これは，HTTP リクエストの候補を作成する際，HTTP リクエストのパラメータに渡す値として無数の組み合わせを考慮するのではなく，あらかじめ用意しておいた文字列のリストを使用して有限の HTTP リクエストを作成するからである．

行動の有限集合 A は，すべての送信可能な HTTP リクエストの集合である．エージェントが取る行動は，HTTP リクエストの送信であり，エージェントが送信可能な HTTP リクエストはすべて行動の有限集合 A に含まれる．

遷移関数 $T(S, a, S')$ は，状態 S において行動 a を選択したときに，状態 S' に遷移する確率を表している．エージェントは HTTP リクエストの候補の集合から 1 つ選んで送信し，返ってきた HTTP レスポンスを解析して HTTP リクエストの候補の集合に要素を追加する．このとき，HTTP リクエストを送信する前の HTTP リクエストの候補の集合が状態 S，選んだ 1 つの HTTP リクエストが行動 a，HTTP レスポンス解析後の HTTP リクエストの候補の集合が状態 S' である．提案手法で用意した Web アプリでは，HTTP レスポンスの内容はランダムに決定されない．つまり，HTTP リクエストの候補の集合 S から HTTP リクエスト a を選んで送信するという動作を何度繰り返したとしても，返ってくる HTTP レスポンスは同じであり，HTTP レスポンス解析後の HTTP リクエストの候補の集合 S' は常に同じである．よって，遷移関数 $T(S, a, S')$ は，0 または 1 の値を取る．

報酬関数 $R(S, a, S')$ は，状態 S において行動 a を選択し，状態 S' に遷移したときに得られる報酬を表している．HTTP リクエストの候補の集合 S を状態として持ち，攻撃に成功する HTTP リクエスト a' を送信した場合，エージェントは報酬 0 を受け取り，終了状態 S' へと遷移する．一方，攻撃に成功しない HTTP リクエスト a'' を送信した場合，エージェントは報酬 -1 を受け取り，状態 S'' へと遷移して攻撃を継続する．よって，エージェントが取りうる値は，0 または -1 のいずれかである．この報酬設計により，エージェントはなるべく早く攻撃に成功しようとする．

5. Deep Q-network

提案手法では，強化学習のアルゴリズムとして，Deep Q-learning を用いた．Deep Q-learning で使用するニューラルネットワークは DQN (Deep Q-network) と呼ばれ，次に送信する HTTP リクエストを選ぶ際に使用する．本章では，DQN に入力する HTTP リクエストの前処理や，DQN のアーキテクチャについて述べる．DQN の実装には TensorFlow 2.0 Keras の Functional API[9] を使用した．

5.1 HTTP リクエストの前処理

DQN には文字列である HTTP リクエストをそのまま入力することはできないため，HTTP リクエストに対して前処理を行う．提案手法で扱う HTTP リクエストのメソッドは，GET または POST のいずれかとし，パラメータ数の上限を 2 とする．

まずは 1 つの HTTP リクエストを以下のような 6 次元ベクトルで表現する．

- **[ファイル名, メソッド名, パラメータ 1, パラメータ 1 の値, パラメータ 2, パラメータ 2 の値]**

パラメータが存在しない場合は，パディングを行う．以下に例を示す．

- **POST メソッドで「login.php」に「name=test&pass=mypasswd」を送信する場合**
  ⇒ [login.php, post, name, test, pass, mypasswd]
- **GET メソッドで「index.php」にアクセスする場合**
  ⇒ [index.php, get, (pad), (pad), (pad), (pad)]

次に，6 次元ベクトルの各成分の文字列を整数に変換する．以下に例を示す．

- **[login.php, post, name, test, pass, mypasswd]**
  ⇒ $[4, 2, 5, 11, 7, 12]$
- **[index.php, get, (pad), (pad), (pad), (pad)]**
  ⇒ $[3, 1, 0, 0, 0, 0]$

文字列から整数への変換では，同じ文字列には同じ整数を，異なる文字列には異なる整数を割り当てる．以上が HTTP リクエストの前処理である．

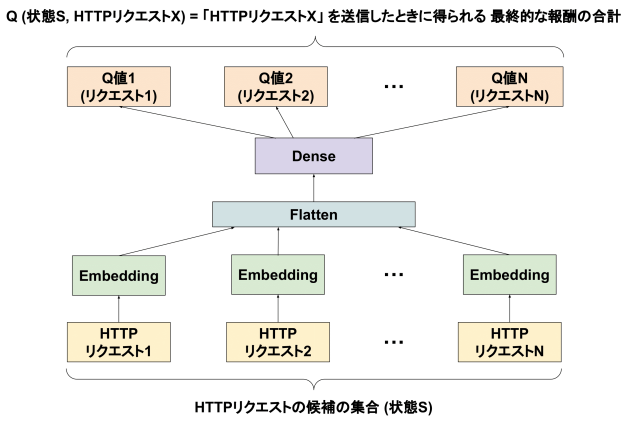

図 3 DQN アーキテクチャの概要
Fig. 3 DQN architecture overview

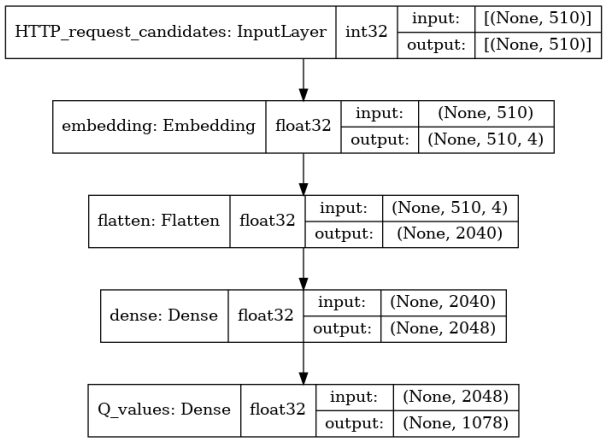

図 4 DQN アーキテクチャの詳細
Fig. 4 DQN architecture

5.2 DQN のアーキテクチャ

Deep Q-learning では, Q 関数を DQN で近似する．DQN が近似する Q 関数：$Q(s, a)$ は，状態 s において行動 a を選択したときに得られる最終的な報酬の合計を表わす関数である．あらゆる状態 s において，Q 関数の出力値である Q 値が最も高くなる行動 a を選択し続けることで，最終的な報酬の合計が最も高くなると期待できる．

エージェントは 1 つの Web アプリ内を巡回する過程で，HTTP リクエストの候補を DQN を使って評価し，攻撃に成功する，あるいは攻撃成功に繋がる可能性が最も高い HTTP リクエストを選び送信する．

図 3 は提案手法で用いる DQN アーキテクチャの概要，図 4 は DQN アーキテクチャの詳細である．

DQN には，HTTP リクエストの候補の集合 (状態 S) を入力する．1 つの HTTP リクエストは 6 次元ベクトルで表され，HTTP リクエストの候補数の上限を 85 とする．85 に満たない場合はゼロパディングを行うため，1 バッチ分の入力データは常に $6 \times 85 = 510$ 個の成分を持つベクトルである．

入力された HTTP リクエストは Embedding 層で処理される．1 つの HTTP リクエストは以下のような 6 次元ベクトルで表されると述べた．

– **[ファイル名，メソッド名，パラメータ 1，パラメータ 1 の値，パラメータ 2，パラメータ 2 の値]**

Embedding 層は単語の特徴量を表す役割を持ち，ファイル名やメソッド名，パラメータ，パラメータの値など，各単語を 4 次元ベクトルへと変換する．よって，6 次元ベクトルである HTTP リクエストが 6×4 の行列に変換される．

次に，Flatten 層によって行列をベクトルに変換し，その後データは 2 つの Dense 層で処理される．

DQN の出力は Q 値であり，Q 値は状態 S において特定の HTTP リクエストを送信したときの，最終的な報酬の合計の予測値である．図 4 では，出力は 1078 次元のベクトルとなっているが，これはエージェントが送信することのできる HTTP リクエストの種類数である．複数の Web アプリにおいて，それぞれの Web アプリを攻撃する際に作成される HTTP リクエストの種類は異なる．例えば，Web アプリ A に対して攻撃をするときに HTTP リクエスト 1 は作成される可能性があるが，HTTP リクエスト 2 はどのようにしても作成されることはないとする．一方 Web アプリ B に対して攻撃をするとき，HTTP リクエスト 2 は作成される可能性があるが，HTTP リクエスト 1 はどのようにしても作成されることはないなど，それぞれの Web アプリを攻撃する際に作成される HTTP リクエストの種類は異なる．次に送信する HTTP リクエストを決定する際，エージェントは Q 値を確認する．ただし，1078 個すべての Q 値を確認するわけではなく，候補に含まれる HTTP リクエストに対応した Q 値のみを確認する．図 3 の HTTP リクエストの数 N は候補の数であり，候補が 50 個存在していた場合は $N = 50$ となる．50 個の Q 値のうち，最も高い Q 値に対応する HTTP リクエストを，攻撃に成功する，あるいは攻撃成功に繋がる可能性が最も高いと判断し，送信する．

6. 実験結果

本章では，Deep Q-learning によって学習したエージェントの性能を評価する．エージェントの性能評価の指標として，攻撃成功に必要な HTTP リクエストの送信回数，送信した HTTP リクエストの内容，この 2 点を用いる．

6.1 実験環境

エージェントが攻撃対象の Web アプリ内を巡回する際，エージェントは Web アプリのソースコードにはアクセスしない．よって，エージェントのための Web アプリを用意する際，ソースコードを書く必要はなく，Web アプリのインターフェースを模倣したモデルを用意すれば十分である．本実験では，Web アプリのモデルを 300 個用意した．300 個のデータのうち，270 個を訓練データ，30 個をテス

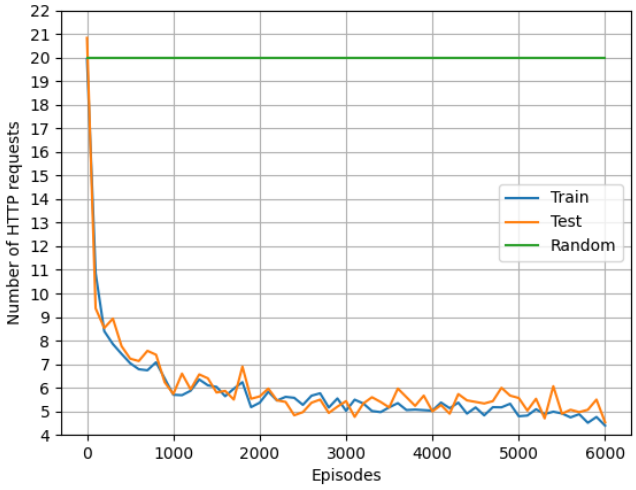

図 5 攻撃成功に必要な HTTP リクエストの送信回数の推移
Fig. 5 Number of HTTP requests sent for attack

トデータとした．Web アプリのモデルを作成する際は，図 1 のように，どのページにどのようなリンクやフォームがあるのかを想定し，作成した．

それぞれの Web アプリには，SQL インジェクションの脆弱性が必ず含まれている．送信すると攻撃成功となる HTTP リクエストが 1 つ以上存在し，そのうちの 1 つでも送信すれば，攻撃成功となる．本実験で用意した Web アプリは，最短で 3 ～ 4 回目，最長で 75 ～ 82 回目の HTTP リクエストで攻撃に成功するような設計になっている．

Web アプリに含めた SQL インジェクションの脆弱性は，クォーテーションを使って WHERE 句を TRUE にできるものや，ログインを回避できるタイプのものなど，なるべく実際の Web アプリに存在するような脆弱性を含めるようにした．

6.2 攻撃成功に必要な HTTP リクエストの送信回数

図 5 は，エージェントが攻撃に成功するまでに送信した HTTP リクエストの数の推移である．100 Episode 毎に，本実験で用意した全ての Web アプリをエージェントに攻撃させた．攻撃に成功するまでに送信した HTTP リクエストの数を記録し，訓練データとテストデータで別々に平均を取った．図 5 は訓練データとテストデータ，それぞれの平均値の推移を示している．DQN を用いずに，次に送信する HTTP リクエストをランダムに決定した場合，訓練データでは平均 20.01 回，テストデータでは平均 19.98 回であった．エージェントによる攻撃とランダムな攻撃を比較するために，図 5 にはランダムな攻撃による HTTP リクエストの送信回数 (= 20) も示している．

訓練データとテストデータでは，共に同じような推移をしていることがわかる．未学習の状態では，訓練データに対する攻撃では平均 19.94 回，テストデータに対する攻撃では平均 20.83 回の HTTP リクエストの送信で攻撃に成功した．これはランダムな攻撃による HTTP リクエストの送信回数とほぼ一致しており，未学習時のエージェントは次に送信する HTTP リクエストをランダムに選んでいる状態と変わらないと言える．学習が進むにつれ，攻撃成功に必要な HTTP リクエストの送信回数は減少していき，最終的にわずか約 5 回の HTTP リクエストの送信で攻撃に成功するようになる．

訓練データとテストデータ，それぞれの平均値だけでなく，Web アプリ毎に何回の HTTP リクエストの送信で攻撃に成功したのかを記録した．図 5 の $Episodes = 6000$ 時点のエージェントの場合，3 回または 4 回で攻撃に成功する Web アプリが全体の約 72.3%(217/300) を占めていた．一方，攻撃成功までに 10 回以上の HTTP リクエストを送信する必要があった Web アプリも一定数存在し，そのような Web アプリは全体の 5%(15/300) を占めていた．

作成した Web アプリは，すべてが互いに全く異なるわけではなく，似ている Web アプリが存在し，いくつかのタイプに分けることができる．攻撃成功までに 10 回以上の HTTP リクエストを送信する必要があった Web アプリを確認すると，そのすべてが同じタイプの Web アプリであり，図 1 の Web アプリと似た構造をしていた．次節で述べる図 6 の Web アプリは，図 1 とは異なるタイプの Web アプリである．図 1 の Web アプリの脆弱性は，ログインページではなく，パスワード変更に関するページに存在する．一方図 6 の Web アプリの脆弱性は，ログインページに存在する．また，脆弱性が存在する箇所だけでなく，攻撃成功となる HTTP リクエストの数も異なり，図 1 の Web アプリは 1 個，図 6 の Web アプリは 8 個である．300 個の Web アプリに存在する，全ての攻撃成功となる HTTP リクエスト (2890 個) を調べると，ログインページへの HTTP リクエストは 1826 個あったのに対し，パスワードの変更やリセットに関するページへの HTTP リクエストは 72 個しかなかった．したがって，図 1 のような Web アプリはデータ数が少なく，エージェントにとってイレギュラーな Web アプリだと言える．このような理由により，図 1 にあるような Web アプリに対する攻撃には，比較的多くの HTTP リクエストの送信が必要だったと考えられる．

次節以降は，学習済みのエージェントが送信した HTTP リクエストの内容について調査する．

6.3 調査に使う Web アプリの内容

テストデータの Web アプリを用いて，学習済みのエージェントが送信した HTTP リクエストの内容を確認する．図 6 は，調査に使う Web アプリを有向グラフで示したものである．

この Web アプリには「index.php」，「blog.php」，「login.php」の 3 つのファイルが存在し，以下のように HTTP リクエストを送信することで攻撃に成功する．

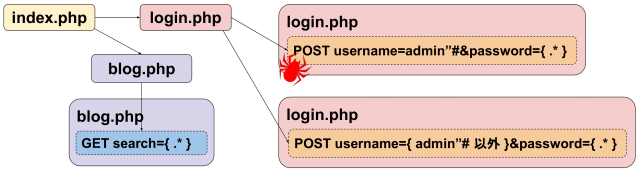

図 6 調査に使う Web アプリ
Fig. 6 Web application used for research

(1) **GET index.php**
(2) **GET login.php**
(3) **POST login.php**
username=admin”#&password={.*}

「index.php」は，Web アプリのホームページの役割を果たす．「blog.php」と「login.php」のリンクを含んでおり，最短で攻撃に成功するためには，「blog.php」ではなく「login.php」を選んで送信する必要がある．

「blog.php」は，ブログ記事の一覧ページであり，ブログを絞り込むための検索窓 (フォーム) が存在する．フォームのメソッドは GET，パラメータ名は「search」である．「blog.php」にアクセス後，パラメータ「search」には，8 つの文字列を含むリストを使用するため，8 つの HTTP リクエストの候補が新たに追加される．

「login.php」はログインページであり，SQL インジェクション攻撃によって認証を回避し，管理者権限でログインすることができる脆弱性を含んでいる．ログインフォームのメソッドは POST で，パラメータは 2 つあり，「username」と「password」である．「username=admin”#」とすることで，パラメータ「password」の値に関係なく，管理者権限でログインすることができる．これは，管理者が使用するユーザ名が「admin」という想定である．実際に攻撃を行う際，パラメータ「username」に与える値は「admin”#」以外にも考えられるが，エージェントが作成する HTTP リクエストの候補は 8 つの文字列を含むリストを使って作成されるため，「username=admin”#」とすることが唯一の攻撃成功条件である．

6.4 送信した HTTP リクエストの内容

以下は実際にエージェントが送信した HTTP リクエストである．図 6 の Web アプリに対して，エージェントは最短回数 (3 回) の HTTP リクエストの送信で攻撃に成功している．

(1) **GET index.php**
(2) **GET login.php**
(3) **POST login.php**
username=admin”#&password=’ OR 1=1#

送信する HTTP リクエストを決定する際，どのような HTTP リクエストの候補があり，どのような優先順位をつけたのかを確認する．

(1) **GET index.php**

攻撃を開始して最初の HTTP リクエストなので，候補は 1 つのみである．候補が 1 つしかないので，必然的にエージェントはその唯一の HTTP リクエストを最優先と判断し，送信する．

(2) **GET login.php**

最初の HTTP リクエストを送信した結果，図 6 より，「blog.php」と「login.php」への HTTP リクエスト (GET メソッド) が候補として作成される．これら 2 つの候補を評価する際，「login.php」が 1 位，「blog.php」が 2 位と判断され，その結果 GET メソッドで「login.php」へのアクセスが行われた．

GET メソッドによる「login.php」へのアクセスは直接攻撃に成功するような HTTP リクエストではないにも関わらず，1 位と判断されているのは注目すべき点である．これは，学習したエージェントが「blog.php」よりも「login.php」に遷移したほうが攻撃成功までに送信する HTTP リクエストの合計回数が少なくなると判断した結果である．

(3) **POST login.php**
username=admin”#&password=’ OR 1=1#

「login.php」にアクセスした結果，図 6 より，64 個の HTTP リクエスト (POST メソッド) が候補として追加される．これは，8 つの文字列を含むリストを用いて，2 つのパラメータを持つ HTTP リクエストの候補を作成したからである ($8 \times 8 = 64$)．優先順位は 1 位から 65 位まで存在し，表 1 は上位 10 個と下位 10 個の HTTP リクエストの候補である．

6 位と 10 位を除く上位 10 個の HTTP リクエストは，パラメータ「username」が「admin”#」または「admin’#」となっている．パラメータ「password」に関しては，特別な法則性は確認できない．学習データには「username=admin”#」ではなく「username=admin’#」が攻撃成功となるものも存在しているため，このような優先順位になっていると考えられる．

一方，下位 10 個の HTTP リクエストに関しては，65 位を除き，パラメータ「username」が「admin”#」と「admin’#」のどちらでもない．パラメータ「password」に関しては，上位のものと同様に，特別な法則性は確認できない．

パラメータ「username」にのみ法則性が現れていることから，エージェントはパラメータ「username」を SQL インジェクション攻撃に使用するパラメータとして認識していることがわかる．また，攻撃に成功する可能性が高い「admin”#」と「admin’#」がパラメータ「username」に含まれている HTTP リクエストの優先順位は高く，そうではない HTTP リクエストの

表 1 HTTP リクエストの候補の優先順位
Table 1 HTTP request candidate priority

	username	password
1 位	admin"#	' OR 1=1#
2 位	admin'#	1 OR 1=1#
3 位	admin'#	admin'#
4 位	admin'#	test
5 位	admin'#	" OR 1=1#
6 位	" OR 1=1#	mypasswd
7 位	admin"#	admin
8 位	admin"#	admin"#
9 位	admin'#	admin"#
10 位	admin	" OR 1=1#
56 位	1 OR 1=1#	' OR 1=1#
57 位	" OR 1=1#	admin'#
58 位	" OR 1=1#	' OR 1=1#
59 位	test	test
60 位	' OR 1=1#	admin'#
61 位	mypasswd	admin"#
62 位	mypasswd	" OR 1=1#
63 位	test	1 OR 1=1#
64 位	' OR 1=1#	" OR 1=1#
65 位	admin"#	1 OR 1=1#

優先順位は低いことから，エージェントはパラメータの値を決定する際，攻撃に成功しやすい文字列を優先的に選んでいると言える．

7. まとめ

本稿では，Web アプリに対して SQL インジェクション攻撃を自動的に行う強化学習のエージェントを提案した．提案手法では，強化学習のアルゴリズムとして Deep Q-learning を利用し，次に送信する HTTP リクエストを DQN によって決定した．エージェントの学習が進むと，攻撃成功に必要な HTTP リクエストの送信回数は大幅に減少し，エージェントは攻撃に成功する可能性が高い HTTP リクエストを優先的に送信するようになった．

SQL インジェクション攻撃を行うためには，「脆弱性のあるページに遷移すること」,「攻撃に使用するパラメータを特定すること」,「攻撃に成功する文字列を入力すること」これら 3 つの操作を行う必要がある．従来の Web アプリの脆弱性検査手法では，「脆弱性のあるページに遷移すること」と「攻撃に使用するパラメータを特定すること」は人間が行い，「攻撃に成功する文字列を入力すること」はランダム，あるいは人間によって行われている．提案手法では，これら 3 つの操作を「攻撃に成功する，あるいは攻撃成功に繋がる可能性の高い HTTP リクエストの選定」という抽象化された 1 つの操作と捉え，SQL インジェクション攻撃を行うエージェントを実装した．なお，本稿で述べた提案手法は SQL インジェクション攻撃以外の攻撃にも適用可能である．攻撃が成功したときにフラグを返すような CTF 型の Web アプリが用意できる攻撃である場合，対応が可能である．例えば，XSS 攻撃や OS コマンドインジェクション攻撃に成功したときにフラグを返す Web アプリを使ってエージェントを学習させれば，それらの攻撃を行うエージェントとなる．

しかし，提案手法にはいくつかの課題が残されている．提案手法ではエージェントの学習に使用する Web アプリを自前で用意したが，用意した Web アプリは実際に運用されている現実の Web アプリとは異なる．また，エージェントが送信することができる HTTP リクエストが有限 (1078 個) で，行動の有限集合 A に含まれていない HTTP リクエストの送信ができないという問題がある．よって，自前で用意したデータではなく，実際のペネトレーションテストの HTTP リクエスト・レスポンスのデータを収集して模倣学習に利用したり，次に送信する HTTP リクエストを動的に生成したりなど，実用的な Web アプリの脆弱性検査システムを開発するためには，提案手法の拡張が必要である．これらは今後の研究課題としたい．

参考文献

[1] OWASP ZAP. https://www.zaproxy.org/. Accessed: August 23, 2021.
[2] Burp Suite. https://portswigger.net/burp. Accessed: August 23, 2021.
[3] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, " AEG: Automatic exploit generation, " in Proc. of the Network and Distributed System Security Symposium, Feb. 2011.
[4] J. King, " Symbolic execution and program testing, " Communications of the ACM, vol. 19, pp. 386–394, 1976.
[5] S. K. Cha, T. Avgerinos, A. Rebert and D. Brumley, " Unleashing Mayhem on Binary Code, " Proceedings of the IEEE Symposium on Security and Privacy, pp. 380-394, 2012.
[6] Alhuzali A, Gjomemo R, Eshete B, Venkatakrishnan V, " NAVEX: Precise and Scalable Exploit Generation for Dynamic Web Applications, " In: 27th USENIX Security Symposium (USENIX Security 18); 2018. p. 377–392.
[7] Cyber Grand Challenge (CGC) (Archived). https://www.darpa.mil/program/cyber-grand-challenge. Accessed: August 23, 2021.
[8] AlphaGo. https://deepmind.com/research/case-studies/alphago-the-story-so-far. Accessed: August 23, 2021.
[9] The Functional API - Keras. https://keras.io/guides/functional_api/. Accessed: August 23, 2021.