修士論文

深層強化学習による Web アプリケーションに対するペネトレーションテストの自動化に関する研究

指導教員　　松浦幹太　教授

東京大学大学院 情報理工学系研究科 電子情報学専攻

48-206425　　久野　朔

令和　4 年 1 月 25 日提出

内容梗概

近年、サイバー攻撃による情報の流出やシステムの改竄などの危険性が問題視されている。これに対抗する方策の一つとして、実際に対象環境に対して疑似的な攻撃を行い、侵入につながりうる脆弱性を発見するペネトレーションテストは非常に有効であるとされる。しかし、これには十分に訓練された人員が必要であり、大きなコストが要求される。 この問題を解消するために強化学習・深層学習・深層強化学習などを用いてペネトレーションテストを自動化・効率化する研究が存在している。しかし、実際のペネトレーションテストにおいて利用される脆弱性および複数のツールの情報を直接利用し、なおかつ強化学習・深層強化学習の本領ともいえる状態の遷移をペネトレーションテストに根差した形で取り入れた研究は確認した限りでは存在していない。本稿では、多様な攻撃手法と対象の種類が存在しており、非常に使用頻度の高い Web アプリケーションというカテゴリを対象としたペネトレーションテストの効率化のために深層強化学習を用いて、既存のツールおよび exploit を統合することを目標とする。最初にペネトレーションテストそのものの概要・問題点と、既存の機械学習技術を用いたペネトレーションテストの自動化・効率化に関する研究を紹介し、その後、一つ目の実験として単純なペネトレーションテスト環境の再現として、著名な Web アプリケーション 15 種類の現存しているバージョンと公開されている exploit を元に模擬環境を作成し、それぞれのアプリケーションに対し、バージョンに適応する exploit を見つけ出すタスクを設定する。その後、このタスクに PPO アルゴリズムを用いた深層強化学習を適用し、学習を行ったエージェントが正しい exploit を見つけ出せることを示す。 次いで、exploit だけではない多数のツールの効力と、状態遷移の概念を導入した模擬環境を作成し、これに対し同様に深層強化学習を行い、より複雑なペネトレーションテスト分野における深層強化学習の有効性について検討する。

目次

Chapter 1　序論

1.1　ペネトレーションテストとは

1.1.1　ペネトレーションテストの概要

ペネトレーションテストとは、優れた技術と信頼を持ったセキュリティエンジニアが診断対象となる企業ネットワークなどに擬似的なサイバー攻撃を行い、システムの侵害可能性を検証することでセキュリティ上の欠陥をあぶりだすテストのことであり、攻撃者が組織のコンピューティングシステムとネットワークにアクセスしたときに発生する可能性のあるリスクを特定するために使用される。[1] 基本的にテストを提供する側の人間とテストを受ける側の組織の人間、双方の合意をもってシナリオを作成し、それに則って実際のアセスメントが行われる。例としては、「内部不正者の手によってコンピュータを一つ奪われてしまい、それを足掛かりとして攻撃を行う」「マクロウイルスが仕込まれた標的型メールを社員が開いてしまい、そこから侵入される」「Web アプリケーションの更新が遅れており、脆弱性を突いて侵入される」などが考えられる。その後は基本的にシステムを平行に移動しながら権限を昇格していくラテラルムーブメントを行い、最終的にドメインの管理者権限や機密情報 (もしくは、機密情報と定められたファイル) などを取得するのが主な目標となる。

メリットとして、実際に攻撃者目線からシステムをテストすることが出来るため、社内診断では見つけづらい脆弱性を発見できる可能性がある点、シナリオ次第で単にプログラム上だけで発生する脅威だけではなく人間がもたらす標的型メールや水飲み場攻撃などのいわゆる人間を騙すサイバー攻撃であるソーシャル・エンジニアリングの手法に対する防御方法を知り、またそれにいかに対応できるだけの力を持っているか評価ができる点、そして単に脆弱性を見つけ出すだけではなく、実際の攻撃が発生する前に問題点を埋めるための緩和計画を見積もれる点となる。

なお、Web アプリケーションやクラウド、ネットワークなどシステムに様々なパケットの送信・リクエストを行い、OWASP top 10[2] に示されるような様々な脆弱性を発見する脆弱性診断とは、対象となるスコープがすでにリリースしているものであるか、これからリリースするサービスであるという差がある場合や、もしくは攻撃時のシナリオの有無、最終目標などが異なることによる違いが存在する。一方で利用される技術や考え方、使用されるツールには重複している部分もあり両者は混同されやすい。また、両者を組み合わせた VAPT(Vulnerability Assessment and Penetration Testing) という考え方も存在している。[3][4]

1.1.2　ペネトレーションテストの分類

1.1.2.1　起点による分類 [5]

- 外部からのペネトレーションテスト
  外部からのペネトレーションテストでは、インターネットと組織内ネットワークの間を区切る境界防御に対応するテストとなる。ネットワークの外を起点とし、インターネットからアクセス可能な公開サーバーや Web アプリケーションなどを起点に攻撃を行い侵入する。脆弱性診断とも類似点が存在する。今回の実験においては、この一部として Web アプリケーションへのペネトレーションテストを扱う。

- 内部からのペネトレーションテスト
  内部からのペネトレーションテストは、例えば前述したシナリオの一つである内部不正者の手によってコンピュータが奪われ、それを起点に内部ネットワークのアクセスを許すなど、すでに内部への侵入を果たされている状態から始まるペネトレーションテストである。昨今重要視されている考え方である「境界を設けず、内部ネットワークであっても安全であると考えない」ゼロトラストに対応しており、内部不正者やすでに外部からの侵入を受けた状態で、さらに権限を昇格される、ラテラルムーブメントによりネットワーク内を移動されて重要な被害が広がるなどの可能性について検証することが出来る。

1.1.2.2　形式による分類

- 通常のペネトレーションテスト
  単純にネットワークの外からの攻撃グループの存在や内部不正者などのシナリオ上の仮定を置き、それに従って既知の技術 (exploit、パスワードクラッカーツール、脆弱性・ポートスキャンツールなど) を用いて攻撃を行うアセスメント。後述のレッドチームアセスメントとは異なりインシデントレスポンスを行うブルーチームがなく、対象のシステムを実際に攻撃する形で診断することで脆弱な箇所を見つけるものであり、脆弱性診断にも近いものとなる。
- レッドチームアセスメント
  攻撃を行うレッドチームに対し、フォレンジックによる攻撃の識別、封じ込め、根絶、回復と言ったインシデントレスポンスを行い、攻撃に対処するブルーチームを企業・組織内の CISRT(Computer Security Incident Response Team。組織内でセキュリティインシデントの対応を行うチーム) や SOC(Security Operation Center。ネットワークやデバイスなどを監視し、インシデントが発生した際に対応を行うチーム) などが担うことで、実際に組織が攻撃を受けてしまった際における対応力を養うアセスメントを指す。単純に脆弱性を見つけるペネトレーションテストとはシナリオに沿って行うという点は同様であるが、目的が大きく異なる。よりサイバー攻撃の現実に近い形でアセスメントを行うことが出来るため、単に脆弱な部分を特定するだけではなく実際の対応力を養うことにもつながる。

Microsoft など組織内にレッドチームを作り、実際に攻撃のテストを行っている企業も存在する。
必ずしもペネトレーションテストの一種ではなく、また別のセキュリティ施策であるとする考え方も存在している。[6]

- TLPT
脅威ベースペネトレーションテスティング。表記は「Threat-Led Penetration Test」「Threat Intelligenceled Penetration Test」「Threat-based Penetration Test」「Threat Intelligence-based Penetration Test」など英語でも諸説ある。多くレッドチームアセスメントと同一視されているが、TLPT は脅威をもとにオープンソース等から収集・分析された脅威インテリジェンスをもとに作成されるシナリオを用いるため、同様に当該シナリオが対象組織や業態に対する昨今の脅威動向を踏まえたものなのかで同一視の度合いが変わってくることもあるため分けて記述を行った。初めに提唱したのは英国の金融当局であり、米国や中国などで、金融機関など特に高い信頼性を求められる組織において多く利用される。[7]

1.1.2.3　実行形式による分類 [8]

- ホワイトボックスペネトレーションテスト
事前に診断対象のシステムに対してある程度の情報を与えられた状態から、それを生かして攻撃を行うタイプのペネトレーションテスト。脆弱性のありうる場所・リスクの高い場所などをあらかじめ推測しながら進めることが出来るため、時間対効果は高くなりやすいと考えられる。
- ブラックボックスペネトレーションテスト
事前に診断対象の情報がない状態から脆弱性スキャンなどで探りつつ攻撃を行うペネトレーションテスト。より現実に即した形であるといえるが、情報収集に時間が必要となる分、時間に対して見つけられる脆弱性やセキュリティの穴は少なくなる可能性がある。
- グレーボックスペネトレーションテスト
ホワイトボックスとブラックボックスの中間として、部分的に情報を共有した状態で行うペネトレーションテストを指す。効率よく情報を取得しつつ、短時間である程度現実味のあるシナリオに沿ってアセスメントを進めることが出来る、両者の長所を得ることが出来ると考えられる。

1.1.2.4　対象による分類 [8]

- ネットワークペネトレーションテスト
ネットワーク全体と各構成要素に対して、脆弱性を見つけ出すことで将来的な攻撃に備える形式。Web サービスなどに対するペネトレーションテストもこれに一部含まれるものと考えられる。
- モバイルアプリケーションペネトレーションテスト
セキュリティとプライバシーの両方の要件を満たし、顧客に対して信頼してもら

う必要があるモバイルアプリケーションについて行うテスト。

- リモートネットワークテスト
  外部からのサイバー攻撃を模擬し、ファイアウォールを超えて内部にアクセスできる可能性について調べるテスト。
- ローカルネットワークテスト
  内部不正者が存在すること、および内部に侵入を許してしまったという仮定のもと、内部からの脅威に対応できるかを調べるテスト。
- デバイス盗難テスト
  使用されているデバイスが盗まれたという仮定のもと、内部のデータを手に入れたハッカーからの攻撃に対処できるかどうかを見極めるテスト。会社の廃棄物をどこに捨てるかを監視したり、別の従業員と交流を持って追加情報を入手したりできるようにする事前フェーズの計画が必要とされる。正しく社内を権限で分離すること、および新たなセキュリティポリシーを設定することが最大の防御法になるとされる。
- ソーシャル・リバースエンジニアリングテスト
  Pretexing:なりすましや Baiting:餌付けなどのソーシャルエンジニアリング攻撃に対応できるかを調べるテスト。
- 物理的ペネトレーションテスト
  物理的に企業や組織などに侵入しようとするハッカーを想定して、侵入を防ぐことが可能かを調べるテスト。テスターはハッカーとして従業員を装って組織内に侵入しようとする。
- フィッシング (スピアフィッシング) テスト
  標的型メールなどを駆使し、より強力なフィッシングなどのソーシャルエンジニアリングテクニックによる APT 攻撃に対する耐性を測るテスト。悪意のある電子メールや未知のソースからのリンクを回避するよう、従業員の意識とセキュリティのレベルを上げることが防御には重要であり、その試金石ともなりうる。

1.1.3　ペネトレーションテストの手順

まず、診断対象を明確にするために、対象組織との間にヒアリングが行われ、スコープやペネトレーションテストを実行する期間、どのようなシナリオに沿って行うかなどを決定する。その後、その内容をもとに具体的な実行計画が立てられる。攻撃そのものは基本的に、ロッキード・マーチンによって定義された Cyber kill chain[9] や米国の非営利団体 MITRE によって攻撃者の攻撃手法と流れをまとめたナレッジベース MITRE ATT-CK[10] によって定義される手順に沿って、ある程度定式化された流れで行われる。

1.1.3.1　Cyber kill chain[9]

1. 偵察 (Reconnaiance)
   実際の攻撃を行う前に行う偵察行為を指す。単純に脆弱性スキャナーを用いて会

社ホームページや内部システム、クラウドなどで使用されている技術などを割り出す他、特殊なツールを用いて対象組織の構成員の email アドレスを取得する、オープンソースで公開されている企業の github ページに存在するソースコードなどから重要な情報などが手に入らないかを調べるなどの攻撃対象のシステムに直接アクセスする以外の方法も含めて多角的に標的情報を集めることを指す。

2. 武器化 (Weaponize)
   1 で得た情報 (先述の通り、ペネトレーションテストの場合は実行の状況やシナリオによってどの程度の情報を与えられるか、また探せるかは異なると推察される) をもとに、必要な脆弱性を突いて任意のコードを実行させる攻撃用 exploit や標的型メールを用いて相手に excel や word のファイルを開かせ、標的組織の内部ネットワークへの侵入の糸口として実行させるマクロウイルスなど、攻撃に必要な武器となるプログラムを用意する段階を指す。後述するような既存のフレームワークを用いる場合も多いとされる。ペネトレーションテストの場合は標的の環境を変化・破壊することがあってはならないため、細心の注意を払って作成することになる。

3. 配送 (Delivery)
   2 で作成した攻撃用プログラムを実際に対象組織へと送り込む段階を指す。配送の手段としては、後述するように単純に脆弱性を突いて exploit コードを実行させる以外にも、1 で取得したメールアドレスへの標的型メールに添付する、社員にテスターが作成した Web サイトへとアクセスさせ、マルウェアをダウンロードさせるドライブバイダウンロード攻撃、組織内に落とした USB を組織の構成員に拾わせ、それを通して実行させるなど、様々な手法が存在する。

4. エクスプロイト (Exploit)
   3 で配送した擬似マルウェアなどの攻撃コードを実行する段階を指す。また、標的の脆弱性を発見しそれに対する exploit を実行することもある。この段階ではマルウェアの本体をそのまま実行するより、部分的なコードの実行にとどめ、次のマルウェア本体をダウンロードするインストールに繋げることになる。本論文では主に、既存の脆弱性を突いて攻撃を行い、任意のコードを対象環境内で実行すること (Remote Code Execution) を exploit と呼ぶこととする。exploit が可能であるということは次のステップ以降の攻撃を含め、対象環境内におけるほぼすべての侵害行為が可能である、非常に危険な状況である。

5. インストール (Install)
   攻撃者が用意したマルウェアの本体をインストールする。これが Windows Defender をはじめとするアンチウイルスソフトによって発見され、隔離されたなら攻撃は失敗となる。

6. C2 接続の確立 (command & control)
   マルウェアから C2 サーバーへの接続を行い、対象組織のコンピュータを制御で

きる状態にする。C2 サーバーとは別名を command & control サーバーといい、攻撃者と標的ネットワークを仲介するサーバーである。基本的に外部のネットワークから社内ネットワークへの直接のアクセスはできないため、対象システムにインストールした (擬似) マルウェアはサーバーへと一定の時間間隔をあけながら定期的に接続し、攻撃者が送ったコマンドなどの指令を持ち帰る。これを実行後、次に接続を行う際にはその結果を C2 サーバーに送り出すという仕組みとなっている。

7. 目的の実行 (Actions on Objective)
   標的マシンにおいて目的となる行動を行う。通常のサイバー攻撃では、攻撃者は内部ネットワーク内のさらなる侵入・横展開 (ラテラルムーブメント)・ドメイン内での権限昇格を行いながら侵害を拡大していく。

ペネトレーションテストにおいてもラテラルムーブメントや権限昇格は行われることは多いが、実際に機密データを盗むようなことは基本的にはなく、あくまで機密データに触れられるということを示すための目的、たとえば CTF のようにフラグとなるファイル・文字列が設定されることが多いと考えられる。
また、MITRE att& ck[10] は攻撃の内容とそれぞれで用いられる手法についてまとめている。Reconnaissance(偵察)、Resource Development(資源構築)、Initial Access(初期アクセス) Execution(実行)、Persistence(持続化)、Privilege Escalation(権限昇格) Defense Evasion(検知回避)、Credential Access(クレデンシャルアクセス) 、Discovery(探索)、Lateral Movement(横展開)、Collection(情報取得)、Command and Control(C2 接続)、Exfiltration(データの引き出し)、Impact(影響) の 14 種類であり、上述の要素をより細分化したものとなる。
以上のような流れに従い、ペネトレーションテストは行われる。期間終了後、具体的にアセスメントの結果をもとにレポートを作成し、どのような箇所・対応に問題があったのかなどを明確にしたうえで報告・指導を行い、対象組織はその改善に努める。

1.2　ペネトレーションテストで使用されるツール

ペネトレーションテストでは多種多様なツールが使用される。状況やシナリオ、想定する攻撃によっていくつかの種類に分かれている。

1.2.1　ポート・脆弱性スキャンツール

対象マシンに各ポートにパケットを送ることでポートスキャンを行い、開いているかどうかを調べる、もしくは脆弱性を突くパケットやサービスのバージョンを調べるパケットを送ることで既存の脆弱性の有無を調べるツールである。おもに偵察 (Reconnaissance) の場面で用いられ、攻撃の足がかりとなる脆弱性やサービスの有無を確認するために用いられる。

1. Nmap[11][12]
   セキュリティ業界においてデファクトスタンダードとされるオープンソースのポートスキャンツールであり、非常に高速に動作し、また後述するような様々な機能・オプションを有している。また、lua で書かれた、http のディレクトリを列挙するものや smb などの脆弱性をスキャンするものなど、多種多様な拡張スクリプトが開発されており、脆弱性スキャンツールとしても使うことが可能。代表的な機能 (オプション) としては、以下のようなものがある。

   - -p オプション:スキャンするポートの範囲を選択する。単純に特定のポートや全てのポートを選択するほかに、統計的に使われやすいポートを選択するオプションなども存在している。
   - -sV オプション: 標的に存在するサービスとバージョンの検出を行う。事前に用意された多種類のパケットを送った上で、その応答から動いているサービスとそのバージョンについて類推している。
   - -sT オプション: TCP ポートのみのスキャン。逆に UDP ポートのみを指定してスキャンを行うことも出来る。
   - –min-rate オプション: 秒間に送るリクエストの最小数を 1000 パケットに固定する。これによりスピーディなスキャンを行うことが出来るが、検知漏れが発生する可能性も高くなる。
   - -O オプション: 様々なパケットを送ることで対象の OS バージョンを取得するオプション。Windows、linux などに対応しているが、-sV オプション同様パケット応答からの類推であり、間違える可能性は存在する。
   - -A オプション:-O オプションによる OS のバージョンの取得と、-sV オプションによるバージョンの取得をまとめて行うオプション。

2. shodan[13]
   shodan はネットワークスキャンを行うための特殊な検索エンジンであり、ツールなどのインストールを行わずともブラウザがあれば Web から直接使用することができる。このツールではインターネットにつながったデバイスを検索し、コンピュータのみならず Web カメラや IoT デバイス、プリンタなど、ネットワークに繋げることができなおかつ外から接続可能なデバイスを列挙することが出来る。また、単に存在を確認するだけではなく、Nmap と同様送ったパケットからどのようなソフトウェアが使われているかを列挙することもでき、その情報を元に「パスワードのない VNC サーバーや RDP サーバーを列挙しそのスクリーンショットを取得する」、「CVE 番号などを利用して特定の脆弱性のスキャンを行う」など幅広い使い方が可能である。

3. wpscan[14]
   wordpress というブログ作成ソフトウェアに特化したスキャンツール。上記のほかにも wordpress を攻撃する際に使用されているプラグインやログインページ、

テーマなどの多くを列挙し、パスワードリストを用いての総当たり攻撃も可能とする。類似するプログラムとして、joomla のスキャンを行う joomscan、drupal のスキャンを行う droopescan も存在する。

4. sqlmap[15]
   mysql や postgresql、Microsoft SQL Server など多数の攻撃対象に対して、単純な union クエリを用いた攻撃から sleep を用いたタイムベースまでさまざまな手法がある sql インジェクションを検知するツールである。多量のパケットを送ることでその多くを検知しデータベースの内容の取得、データベースの種類によってはシェルコマンドの実行までを行うことが出来る。

1.2.2　脆弱性攻撃ツール

実際に攻撃を行う際に使用されるツールのうち、既存の脆弱性を突いてシェルを取得し、コンピュータを乗っ取るために使うツールを指す。

1. metasploit[16][17]
   metasploit は多数の exploit・スキャンツール・パスワード攻撃ツール・ポストエクスプロイトなどをまとめたペネトレーションテスト用フレームワークである。脆弱性攻撃ツールに分類しているが、ポストエクスプロイトツールやパスワードクラック系ツール・パスワード総当たり攻撃ツールなどを統合した非常に幅の広い用途を持つ統合ツールであり、既存の脆弱性を突く攻撃を行う際のデファクトスタンダードとなっている。基本的には Nmap など偵察ツールで取得した製品やアプリケーションの種類、バージョンなどを検索文字列として、search コマンドで可能性のある脆弱性 exploit を探し、選び出した中から対応するものを選択、exploit に必要なパラメータであるオプションを選択して実行する形になる。また、RPC サーバーを用いて metasploit をプログラムから操作するための API も提供しており、自動化を行うこともできる。本論文においてはこれが重要な機能となる。

1.2.3　ポストエクスプロイトツール

本節では、Metasploit などの攻撃ツールなどで任意コードを実行できる状態になった後、C2 サーバーへのアクセス、さらなる展開などの exploit の後の行動を行うためのツールについて記述する。

1. powershell empire[18]
   powershell empire は powershell を用いたペネトレーションテスト用フレームワークである。基本的にはグラフィカルなインターフェースを伴わない。主にビーコンを生成し、脆弱性やソーシャルエンジニアリングを用いてファイルを開かせることで実行へと移される。操作の方式としては metasploit によく似ており、利用するモジュールを選択し、対応するオプションを組み合わせ、実行するこ

とで結果を得るというものであり、初心者にもある程度簡単に扱うことが出来る。モジュールは基本的に powershell(Windows で使用できるプログラム言語であり、linux におけるシェルスクリプトの役割を持つ。Windows のシェルのコマンドや exe ファイル、bat ファイルなどを利用せずとも Windows 上で多様な情報の取得・改変などを実行することができ、攻撃者にもよく利用される) で実行され、種類も多様に揃っている。

1.2.4　パスワードクラック系ツール・パスワード攻撃ツール

主にハッシュ化されたパスワードをワードリストを用いて総当たり攻撃で解除 (クラック) する、もしくは既存のワードリストをもとにログイン試行を行うブルートフォースアタックを行う。これにより簡単なパスワードを設定していた場合、アプリケーションへのログインなどを行われる脆弱性を検出することが出来る。

1. john the ripper[19]
   パスワードハッシュの候補となる文字列を既存のワードリストやそれをさらに用いたルール (例として、使われやすいワードリストに ”2021 ” などの年号やクオーテーションマークなどをつける) を用いて生成し、mimikatz や SQL インジェクションなどの脆弱性などで取得したハッシュと比較することで元のパスワードを復元する、パスワードクラッキングを行うツール。高速でパスワードをクラックすることが可能である。また、他のプログラムと組み合わせることで Active Directory のチケットのクラッキングも可能である。

2. Hydra[20]
   既存のパスワードリストをもとに高速で標的のサーバーにアクセスすることで、総当たり攻撃を行うためのツール。対象となるプロトコルは http、ftp、smb など多種多様であり、http においても自由にリクエストを定義することで、対象となる Web サイトに合わせた形でログイン試行を行うことが出来る。このツールを用いることで、簡単なパスワードを Web サイトに使用している場合の情報漏洩の危険性をあぶり出すことができる。

1.3　ペネトレーションテストの欠点

ペネトレーションテストは脆弱性を見つけ出す、攻撃者の行動を模擬し、実際に阻止するための演習を行えるなど有用であるが、一方でいくつかの欠点も存在している。この章ではそれについて論ずる。

1.3.1　ペネトレーションテストによって発生するリスク

セキュリティ分野に限らずソフトウェアやハードウェアなどの製品テストにおいては基本的に、本番用環境とテスト用の環境を分離することでテストによる影響が実際

の本番用システムに対しては影響が及ばないようにするのが基本的である。しかし、ペネトレーションテストはセキュリティ意識と対策をテストすると言う都合上、採用されるシナリオによってはそれが難しくなることもある。環境を分離できない状態においては、ペネトレーションテストを行うことによってテストを受ける側もしくは行う側に様々な不利益のリスクが発生しうる。

たとえばテストによってシステム内のデータを操作することは、場合によってはシステムに可用性の面で悪影響をもたらす場合がある。また、テストによって使用されるデータは基本的に機密性の高い、外部に知られてはならないデータであることが多いため、テストを行う側の人間には高い法令遵守性と倫理観が求められる。

Sven Turpe 氏らは、対象システムの分離がなされていないペネトレーションテストによって生じうるリスクを 3 種類に分類し、それぞれにおいて例を挙げている。[21]

- テクニカルリスク (直接的にペネトレーションテストによって発生するリスク)
  - ターゲットもしくは接続しているシステムが誤作動する危険性
  - サービスが停止、もしくはパフォーマンスが落ちる危険性
  - データの改変・欠落・汚染が生じる危険性
  - 第三者へとデータが露見する危険性
  - 接続しているシステムへとデータがこぼれる危険性
  - 不可逆的な現実世界での応答を発生させてしまう危険性
  - 自動的な防御応答を発生させる危険性
  - 実際の攻撃に対する防御力を失う危険性
- 組織的リスク (テストを行うことによって起こりうる副作用のリスク)
  - 不必要なインシデントハンドリングの発生する危険性
  - テスト中及びテスト後の本来対処するべきインシデントに対する注目度が減少する危険性
  - 業務プロセスが停止する危険性
  - 第三者の影響によって評判の悪化する危険性
- 法的リスク
  - 法的義務の乱用
  - 誤って法に触れることを行ってしまうこと

これらのようなリスクが生じてしまう可能性に対する考慮は、ペネトレーションテストの環境隔離の難しさ故に常に重要となるものである。

1.3.2　ペネトレーションテストの難しさと、それに付随するコストの重さ

ペネトレーションテストは上記したとおり非常に複雑な体系を持ち、そのスコープや診断対象のアーキテクチャ、必要な知識・スキルもシナリオによって様々である。ま

た、頻繁に変更されるネットワークに対応することができる必要がある。そのため、よく訓練され、時に発生するエラーにも対応できる知識のあるテスターが必要不可欠となる。[22] 加えて、テストの内容によっては標的型メールや水飲み場攻撃などのソーシャル・エンジニアリング攻撃を行って企業に属する人間を騙すようなテクニック、もしくは実際にオフィスへと侵入を行うような単純にプログラム・情報システム分野のみに止まらない知識も多く必要になってくる。
このような知識の差が存在するために、システムの安全を保障できるようにするためには依頼する側にも正しいテスターの選定が求められ、安くないコストが必要となる。[8] また、比較的に期間が長く、実施する側にとっても大きなコストが必要とされる。このように、ペネトレーションテストは実施する側・依頼する側、共に大きなコストを要求する。また実施する側のチームメンバーはそれぞれが高いセキュリティリテラシーを持っている必要があり、かつ犯罪行為へと引き寄せられることのないような教育が必要となる。[23] こうした教育へのコストもまた大きなものとなる。
本論文においては、この問題に対応するため、深層強化学習を用いた省力化・効率化のためのフレームワークを構築した。

1.4　本論文の貢献

本論文の貢献は主に以下のようなものとなる。

- 既存のペネトレーションテストの機械学習技術を用いた省力化について調査を行ったこと。
- Web 領域におけるペネトレーションテストにおいて既存のツール・実際に存在する exploit およびアプリケーションの情報を用いてバージョンからの exploit の探索、および認証による多層的な状態空間の変化を伴う exploit の探索の二種類の現実に即した模擬環境タスクを作成したこと。
- 上記した二種類のタスクの探索において深層強化学習を用いて成果を上げることでペネトレーションテスト分野における深層強化学習の有用性を示し、また複数の深層強化学習アルゴリズムを用いてそれぞれの長所と短所について比較を行ったこと。

1.5　本論文の構成

本論文では、第二章でペネトレーションテストを機械学習・強化学習・深層強化学習で自動化させる試みについて紹介し、第三章・第四章でその内容を受けて提案した新たな手法と行った実験について論じ、第五章で今後の課題について述べて結論とする。

Chapter 2　先行研究

第 2 項で述べたように、ペネトレーションテストには「大きなコストが要求される」「法やルールを遵守し、また遵守させる必要がある」の、大きく分けて二つのリスク及び欠点が存在している。
法・ルールの遵守という面では、事前にスコープをしっかりと定め、予期せぬ行動をペネトレーションテスターが誤ってしてしまう余地のないよう、使用する手法・手段などをよく話し合う以外に有効な対策があるとは言い難い。
一方で、大きなコストが要求されることに関しては、プランニング言語や機械学習を用いた自動化をメインとして、いくつかの方向性でペネトレーションテストの過程の研究がなされている。

本論文では、機械学習技術を用いた研究に着目する。ペネトレーションテストの強化学習・深層強化学習などを用いた自動化に関しては、主に二つのアプローチが存在している。

1. attack graph の探索を自動化するアプローチ

   このアプローチでは、attack graph という有向グラフを利用して自動化を行う。attack graph は、攻撃につながりうるような要素をグラフ化したものである。[24] 今回紹介する例では、おもに Mulval[25] と言うツールによってスキャンの結果から自動で生成されたものが利用されている。

   例として、生成されたグラフは図 2.1 のようになる。

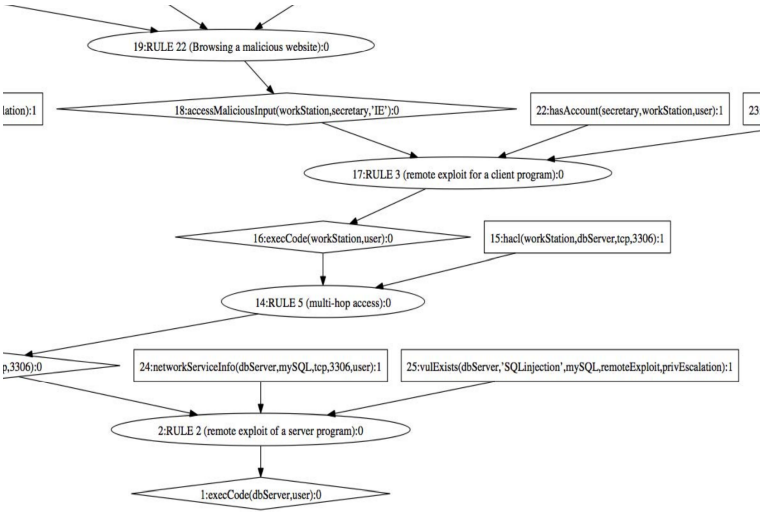


図 2.1: Mulval によって生成された attack graph

図 2.1 のグラフにおいて示されている意味について説明する。それぞれのノードの意味としては、

- execCode(対象、権限)
  特定の権限でコードを実行することを示す。
- hacl(始点, 目標, プロトコル, ポート)
  始点から目標へのネットワークアクセスを意味する。利用するプロトコルとポートも併記されている。
- vulExists(対象, 脆弱性, 性質)
  脆弱性の存在を表すノード。脆弱性を持つ対象アプリケーション、その脆弱性の種類、性質などを表している。
- RULE
  RULE 5:multi-hop access、RULE 3: remote exploit for a client program のように、「このようにして攻撃する」という定義。

のようなものがある。このほかにも、and・or など、様々なノードの種類が存在している。グラフには複数のマシンからなるネットワークトポロジーが要点を絞って抽出され、これを利用することで攻撃に至る道筋と取得可能な権限を要点だけを絞って一覧することが可能となる。MulVal はこのようなグラフを nessus などのスキャンツールによって出力されたレポートのデータを元にして作成することができる。しかし、対象となるネットワークが数十・数百となってくると、ネットワークの構成要素・推測される脆弱性・ここのコンピューターのつながりなどグラフの要素はきわめて大きくなり、それに伴い可能性のある経路は爆発的に増加していく。この難点を解消するために強化学習・深層強化学習などを用いてノードの探索を自動化する形でペネトレーションテストの効率化・省力化を目指した研究がいくつか存在している。

2. 直接的にツールなどの exploit を自動化するアプローチ

   攻撃を行うツールや exploit などを機械学習によって状況に合わせて選択することで、Attack graph などを介さず直接的に自動化を行う。機械学習に用いるデータとしては対象コンピュータ・アプリケーションなどを何らかの形で状態空間として落とし込んだものを扱う。既存のフレームワークなどを流用することができるのが利点となる。

この項では、それぞれのアプローチにおける先行研究を紹介する。

2.1　attack graph の探索を自動化するアプローチ

2.1.1　Mehdi Yousefi 氏らの表形式 Q 学習を用いた attack graph の探索

2.1.1.1　Q 学習

まず Mehdi Yousefi[26] 氏らによる、 MulVal を用いて生成した attack graph を Q 学習を用いて経路探索を行うことで効率化を図る研究について紹介する。

Q 学習とは、強化学習の基本的な形態であり、

$$Q(St, At) \leftarrow Q(St, At) + aRt + 1 + maxaQ(St + 1, a) - Q(St, At)$$

に沿うように各状態の Q 値を更新していき、Q 値が大きくなるような行動をとるようにすることで報酬の最大化を目指す手法である。

attack graph の探索においてこの行動空間は他のノードへの移動であり、状態空間は現在の位置と見なせる。

この研究では attack graph の探索に Q 学習を利用しているが、攻撃グラフは多くの場合複数のゴールを持つ。通常のアルゴリズムそのままだと対応しきれないため、この研究ではアルゴリズムに変更を加えている。 具体的にはマトリクス R の中で現在の目標だけを抽出したサブマトリクス r を用意し、これを Q 学習テーブル更新に利用する形になる。 また、attack graph をそのまま Q 学習に使用することは難しいので、始点ノード、脆弱性ノード、終点ノードのみを取り出して簡略化すると言う手法を用いて遷移グラフ (transition graph) の形に変形している。

2.1.1.2　報酬

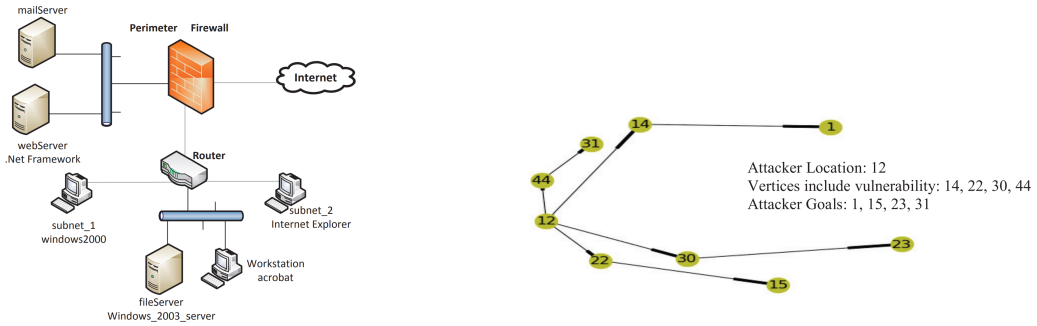
報酬の設定には、CVSS [27] の exploitabilty score を用いる。CVSS スコアとは、共通脆弱性評価システムのことを指し、情報システムの脆弱性に対するオープンで汎用的な評価手法を提供している。この評価手法では、それぞれの脆弱性に対して完全性・可用性・機密性の観点から報酬が決定されている。それぞれのノードに対する移動に際し、

- 後のノードに戻る:0
- 繋がっていないノードに移動しようとしたとき:-1
- 正しく終状態に達した場合:100

となっている。

2.1.1.3　検証

この実験においては、4 種類の脆弱性が用いられている。
図 2.2a のネットワークを検証に用いる。この形状のネットワークに対して mulval で attack graph を生成し、これを transition graph の形にすると図 2.2b のようになる。図 2.3 に試行時の 10step ごとの報酬の合計を表す。 これで正しい経路までたどりつけるよう、Q 学習を行ったところ、どの終ノード、すなわち各マシンでの root 権限でのコード実行をゴールと定めても最終的には報酬が安定した。このことから、Q 学習による経路探索の有効性が示されている。



(a) 実験用ネットワーク　　**(b)** 処理後の attack graph

図 2.2: 表形式 Q 学習を用いた attack graph の実験設定

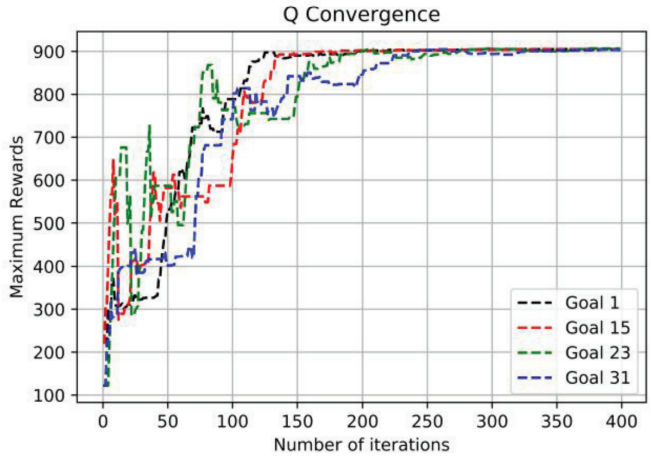


図 2.3: 表形式 Q 学習を用いた実行報酬の推移

2.1.2　Zhenguo Hu 氏らの DQN を用いた attack graph の探索

上記のアプローチは有用ではあるが、表形式である以上、表現可能な状態空間の数に限界がありやがてはメモリに大きな負担をかけるという欠点があった。Zhenguo Hu 氏ら [28] は、この欠点を補うために DQN(深層強化学習) を用いたフレームワークを提案している。 この研究の目的としては現状直接のペネトレーションテストへの使用ではなく、攻撃経路を学生に対し提示し、それに従って学生が攻撃を行うことによる教育への使用となっているが、将来的には metasploit などのペネトレーションツールと組み合わせることによってペネトレーションテストの直接の自動化も視野に入れていると説明されている。

2.1.2.1　DQN

DQN とは、Deep-Q-learning といい、Q 値の算出のために深層学習を用いる形式で、大きな状態空間であっても表現が可能となる利点がある。ただし、得たデータをそのまま順番通りに学習させてしまうと、時系列という形で依存関係が発生してしまう可

能性がある。そのため、依存関係を排除するために一度行動結果を保存し、あるタイミングでまとめて学習を行う experienced buffer、エピソード中はパラメータを固定する fixed target などの工夫が必要になる。今回使用されるモデルは図 2.4 のようになる。

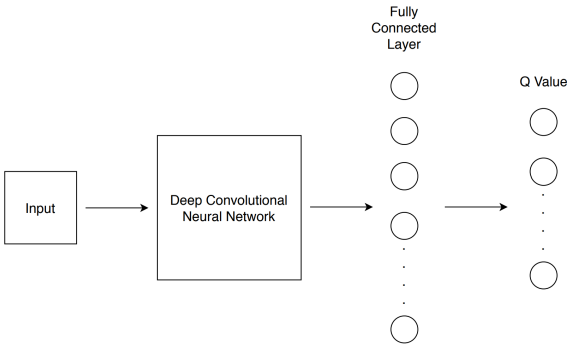


図 2.4: 使用される DQN モデル

2.1.2.2　フレームワークの構成

今回提案されるフレームワークは、主に 4 つの要素から構成されている。1 つめは スキャン用 webapi の shodan、2 つめは前項でも利用した attack graph 生成ツール MulVal で、 スキャンのデータを元に攻撃グラフを生成する。3 つめは生成された攻撃グラフを DQN に適した形に変換するアルゴリズム、4 つめは DQN である。

shodan は第一章で記述した通り、多くのハッカーなどに使われ、ターゲットネットワークをスキャンしその構造や脆弱性のあるデバイスなどを発見する Web アプリケーションである。

表形式 Q 学習を用いた研究と同様に attack graph は DQN にそのままの形では使用できないため変形する必要があるが、この場合は幅優先探索を単純に用いており、途中のノードの省略は特に行っていない。 DQN で扱うデータセットは、ホストをスキャンし生成するホストデータセット、既存の脆弱性データベースを用いた脆弱性データセット、そして DQN の学習に実際に用いる DQN データセットがある。これらのデータセットの使い方は、まずホストデータセットから MulVal で攻撃グラフを生成し、全てのノードに CVSS の脆弱性スコアを振る。最終的に、1 列目に開始状態のノード、2 列目以降に過程ノードのスコアの合計、最終列に最終ノードのスコアという伝送行列形式にしたものが DQN のデータセットとなる。

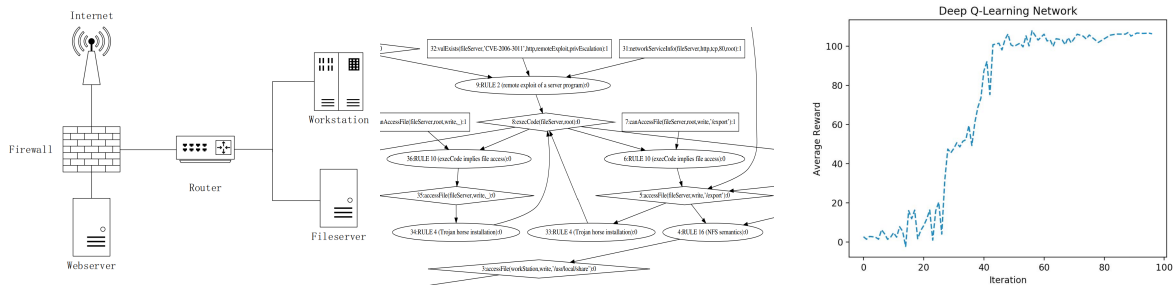
2.1.2.3　報酬関数

それぞれの報酬関数は次のようになる。

- 始状態:0.01
- exploit のノード:CVSS のスコア

- 後のノードに戻る:0
- 繋がっていないノードに移動しようとしたとき:-1
- コード実行やファイルアクセス:1.5
- 正しく終状態に達した場合:100

2.1.2.4　検証



(a) 実験用ネットワーク　**(b)** 生成された attack graph　**(c)** 学習結果

図 2.5: DQN を用いた attack graph の探索

実験の環境は図 2.5a に示すようなアーキテクチャを利用した。図 2.5b に示されているのが攻撃グラフである。同じネットワークのまま、脆弱性を変えて 3000 個の異なる attack graph を生成し、2000 個で学習、1000 個で検証を行った。図 2.5c が学習に際しての平均報酬の変化を示した結果である。最初は報酬が低いが、徐々に大きくなり取得された経路も正確になっていくのが読み取れる。

2.2　強化学習・深層強化学習を利用した直接的自動化

attack graph を構築する手法は攻撃のプランニングのみを自動化しており、実際の exploit 作業はユーザー自身が行う必要があった。以下に示す研究例はその部分を自動化する試みである。

2.2.1　AgentPen

Konstantin Pozdniakov 氏らが提案した、Q 学習を用いることで AgentPen という攻撃ロジックを組み立てる研究。[29] この研究では、いくつかの exploit を直接用意し、単純な環境に対して最適な攻撃への経路を正しく発見できるかを調べていた。今回使用されるモデルは図 2.6 のようになる。構成としては、特徴量を削減するニューラルネットワークであるオートエンコーダを用いて状態数を削減し、RNN を用いて学習させる形式になっている。

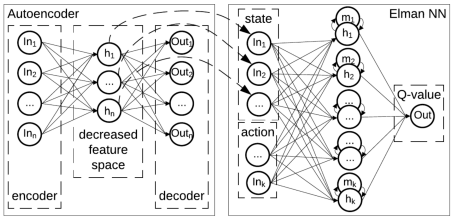


図 2.6: 使用されるニューラルネットワークモデル

TABLE I
AUDIT MDP STATES.

Audit MDP state	Description
Init	Initial audit state
Win	The host has a Windows OS
Lx	The host has a Linux OS
P445	The host has an open port 445
P135	The host has an open port 135
P22	The host has an open port 22
P2525	The host has an open port 2525
Serv1	The host has a vulnerable $RDP_{Service}$
Serv2	The host has a vulnerable $Msvcrt_{Service}$
Serv_SSH	The host has a $SSH_{Service}$
$Expl_1$	Remote $Exploit_1$ is executed
$Expl_2$	Remote $Exploit_2$ is executed
$Expl_3$	Local $Exploit_3$ is executed
SSH_brf	SSH brute force attack is performed
$Expl_{12}$	Remote $Exploit_1$ executed as 2^{nd} exploit
$Expl_{22}$	Remote $Exploit_2$ executed as 2^{nd} exploit
$Expl_{32}$	Local $Exploit_3$ executed as 2^{nd} exploit
Fail	Final(terminal) state, dead end, i.e. exploit failed
Succ	Final(terminal) state, target host hacked

TABLE II
AUDIT MDP ACTIONS.

Audit MDP action	Description
Win_check	Checks for Windows OS
Linux_check	Checks for Linux OS
Port445_check	Checks if Port445 is open
Port135_check	Checks if Port135 is open
Port22_check	Checks if Port22 is open
Port2525_check	Checks if Port2525 is open
$Service_1$_check	Check $Service_1$ process is running
$Service_2$_check	Check $Service_2$ process is running
SSH_check	Check SSH process is running
$Execute\_exploit_1$	Grants remote attacker admin privileges
$Execute\_exploit_2$	Grants remote attacker user-level privilege
$Execute\_exploit_3\_local$	Exploit local services, privilege escalation
Execute_SSH_brf	SSH brute force attack

(a) 状態空間　　**(b)** 行動空間　　**(c)** 学習結果

図 2.7: AgentPen の設定と結果

2.2.1.1　状態空間

図 2.7a が状態空間である。各ポートが開いているかどうか、サーバーが動いているか、OS が windows と linux のどちらであるかなどが定義されている。これらによって取りうる状態の数は非常に大きくなる。

2.2.1.2　行動空間

図 2.7b が行動空間である。各種 exploit、サービスの状態を調べる行動などが定義されている。

2.2.1.3　報酬

$Reward = Rmain + R1 + R2 + R3$ の形で定義される。このとき、

- Rmain:正しく終状態に達した場合の報酬
- R1:metasploit のレーティングのように、選んだ exploit の有効性による報酬
- R2:exploit が使われた場合の OS やサービスが受けるダメージに関する報酬
- R3:一回のエピソードごとの行動の回数に関する報酬。先んじて環境チェックを行い余計な動作を減らすことを期待する。

2.2.1.4　検証

定義した環境に対して実行させたところ、check os → exploit1 という形で管理者権限を取得する方法を見つけ出した。合計報酬は図 2.7c のように順調に上昇した。 また、攻撃中に設定を変えた場合でもそれに適応し、 check os → exploit2 → exploit3 という攻撃経路を見つけ出すことに成功した。

2.2.2　Deep exploit

本節では、先行研究の中で具体的なソースコードが示され、理論と実装に関する理解のしやすい Deep exploit[30] を例とし、どのように実装されているかを説明する。

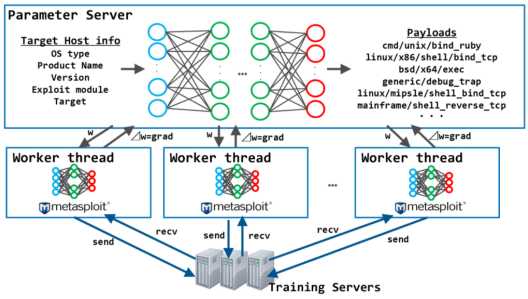


図 2.8: Deep exploit の概要

2.2.2.1　A3C

A3C は DQN から発展した強化学習のアプローチである。図に表すと図 2.8 のようになる。通常の DQN とは異なる 3 つの工夫を組み合わせたものであり、頭文字をとって A3C と呼ばれる。とくに CPU で学習する際などによく利用される。

- Asynchronous
  複数の強化学習エージェントを用意し、分散学習を行わせる手法。これにより所要時間を大きく減らすことができる。

- Actor-critic
  方策を出力する Actor と、状態価値を出力する critic の二つを同時に扱う手法。これにより戦略を立ててそれに従う policy ベース手法と状態の価値を見て判断する value ベース手法の両方を利用することができる。

- Advantage
  一定ステップ先まで Q 関数を更新していくことで、より確かな状態価値を取得する手法。

2.2.2.2　Nmap を利用した情報の読み込み

前章の Nmap を用いてターゲットへのスキャンを行い、対象サーバーの情報を得る。

2.2.2.3　状態空間・行動空間への落とし込み

状態変数は主に以下の 5 種類となる。

- ST_OS_TYPE
  OS の種類を示す。
- ST_SERV_VER
  exploit モジュールとターゲットを示す。
- ST_MODULE
  指定されるモジュールを示す。metasploit の search コマンドを用いて、exploit をサービス名から絞り込んでいる。
- ST_TARGET
  exploit のターゲットを指す。
- ST_SERV_NAME
  サービス名を指す。製品名を事前に定義したシグネチャを用いて数値に変換し、正規化を行っている。

行動空間は、可能な payload 全てのリストとなる。ニューラルネットワークの隠れ層は全部で 4 層となっており、それぞれ 50、100、200、400 のノードを有する。

2.2.2.4　深層強化学習による payload 取得と実行

二種類のモードが存在する。

- train モード
  exploit をランダムに選択し、実行した結果を取得していくことでニューラルネットワークへと経験を蓄積していく。
- test モード
  条件を満たす exploit を全て使用し、それぞれでニューラルネットワークから payload を選択することで攻撃を行う。

2.2.2.5　ポストエクスプロイト

取得したシェルを媒介として、ネットワーク内を Nmap でスキャンし、再帰的に次の標的に向けて攻撃を行う。

2.2.3　ニューラルネットワークを用いたペネトレーションテストツール選択フレームワークの作成

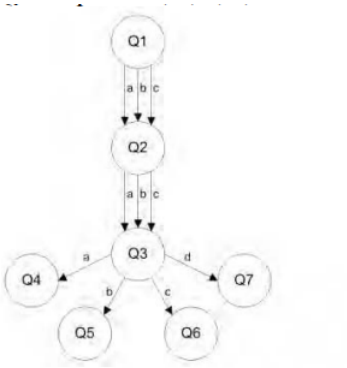
ペネトレーションテストにおいては、きわめて多種多様なツールとそのオプションの組み合わせによる攻撃パターンが存在している。そのため、取り得る攻撃の選択に時間がかかってしまうことがある。このような問題に対応するため、Artem Tetskyi 氏らは出力データが出力に複雑に依存するような状況を扱うアプリケーションにおいてより効果を発揮することが出来るニューラルネットワークをベースとして、ツールの選択を補助するアプリケーションを作成した。[31]

2.2.3.1　ニューラルネットワークに対する入力データ

図 2.9a は、ペネトレーションテストの専門家の意見を参考とし、いくつかのツールの持つ特性を分類したものとなる。必要とするツールを選択するために、ユーザーはいくつかの質問に答える必要が生じると述べられている。しかし、各特性に応じて全ての質問を用意し回答させるのは煩雑であるため、より細かな多数の質問に答える必要のあるエキスパートモードと、ツリー型の質問と応答のアルゴリズムを用いるシンプルモードの二種類が定義された。　ツリー形状のアルゴリズムの概観は図 2.9b のよ

Characteristic	Tools		
	sqlmap	*owasp-zap*	*wpscan*
Cost	0	0	0
Cross-platform	+	+/-	+
SQL- inj search	+	+	-
Exploitation of SQL-inj	+	+/-	-
Cracking hashes	+/-	-	-
Creating a site map	-	+	-
Importing a site map for testing	+	+	-
Export site map for testing	-	+	-
Testing the list of pages	+	+	-
Search XSS	-	+	-
Definition of CMS	-	-	+
Scanning CMS Wordpress only	-	-	+

(a) ツールの特性



(b) 質問ツリーの一例

図 2.9: ニューラルネットワークを用いたツール選択フレームワークの設定

うな形状となっている。ここで、Qi は質問を定義しており、 a、b、c、d が回答となっている。(このとき、質問 Q1 と Q2 は、ほとんどのペネトレーションテスト用アプリケーションに共通している機能的でない特性、たとえばコストやクロスプラットフォームに関する独立した質問である可能性が高く、Q3 以降の質問に関しては各アプリケーションごとに独立した特性である可能性が高い)

そして、これらの質問結果を入力データとして、使用すべきアプリケーションを選択するようになっている。

2.2.3.2　ニューラルネットワークへの追加データ

基本的にこの研究では、アプリケーションの学習用のデータはペネトレーションテストの専門家によって作成されたデータであるが、一つのグループに関する専門家の意見だけでは必ず偏りが発生しないとは限らない。この問題に対処するため、システムのユーザーが追加で情報をアップロードし、ニューラルネットワークを追加で学習できるようになっている。ただし、ユーザーが必ずしも意図的に間違った学習をさせないとも限らないため、これに対する対策として、専門家のデータとユーザーの集合知識を用いて学習されたデータの双方を用いてペネトレーションテストに用いることが推奨されている。

2.2.4　hack the box と metasploit、及び決定木を用いたペネトレーションテストの自動化

Ovidiu Valea 氏らは hack the box[32] という ctf の形態をとったペネトレーションテストの補助用フレームワークとして、決定木と metasploit を用いた自動化フレームワークを提案している。[33]Hack the box 以外に情報源がなかったためにデータセットは極めて小さいが、機械学習のペネトレーションテストへの応用可能性を示している。

2.2.4.1　フレームワークの概要

フレームワークとして、Nmap と metasploit を用いる。

- Nmap
  Nmap のスクリプト機能から、Nmap-vulners、および vulscan を用いる。これらのスクリプトはどちらも、SSH、RDP、SMB などのサービスの CVE と、見つかった CVE の潜在的な exploit に関する関連情報を見つけることで、Nmap のバージョン検出を改善するように設計されている。Nmap-vulners は、実行時にオンラインにある CVE をもとにした脆弱性データベースへとアクセスを行い、既存の脆弱性情報を取得する。一方で、vulscan は、vulscan を初めてダウンロードするときに事前構成された、コンピューター上のローカルデータベースにクエリを実行する。
  脆弱性情報を取得した双方のスキャンスクリプトは Nmap のスキャンを行い、サービスから得た情報と脆弱性情報を照らし合わせて表示する。この情報を元にデータセットを作成する。

- metasploit
  pymetasploit3 ライブラリから呼び出され、深層学習によって設定された exploit を実行する。

2.2.4.2　データセット

データセットは以下のデータを含む。

- ポート
  Nmap によって開いていることが検出された TCP または UDP ポートの番号。

- サービス
  そのポートで実行中のサービス

- CVE
  脆弱性スキャンスクリプトによって見つかったサービスに存在する脆弱性の CVE 番号 (CVE-(発見年)-(発見番号) のフォーマットになっている)

- exploit
  見つかったポート、サービス、CVE、およびオペレーティングシステムで最も使用されている exploit(Metasploit で最も一般的に使用されている exploit のトップ 20 を表す)

- OS
  サービスが実行されているオペレーティングシステム

これらの情報は数値間での依存関係が発生しないよう one-hot エンコーディングされ、合計で 38 列、それぞれ 0 か 1 かの状態空間へとマッピングされる。状態空間の一例を表したのが図 2.10 のようになる。

図 2.10: 状態空間の例

2.2.4.3　exploit の選択方法

exploit を選択するための機械学習手法として、決定木が採用されている。これはやや古典的な機械学習手法の一つであり、回帰問題に対して利用される。決定木には二種類の重要な要素が存在している。

- 情報利得 (information gain)
  分割を行うことで、どれだけのエントロピーを減らすことが出来るかを示す基準値。大きいほどよい。

- ジニ係数
  分類のエラーの可能性を最小限に抑えるための基準。

また、検証を行うための方法として K-means 法を用いた。これは K 個のデータセットに対し、K-1 個を学習に使用し、残る 1 個を検証のために使用する手法である。これを K 回にわたり繰り返すことで、全てのデータセットが情報に使用されることになる。

2.2.4.4　攻撃用フレームワーク

攻撃を行うフレームワークとしては、上述した metasploit を用いている。Deep exploit と同様、metasploit に内在している RPC サーバーを用いることで、プログラムから metasploit を呼び出すことが出来る。この場合は python ライブラリである pymetasploit3 を用いてより手軽に呼び出しを行っている。

2.2.4.5　Hack the box の利用について

Hack the box はオンラインのペネトレーションテストのトレーニングを提供している Web サイトであり、様々な脆弱性を持つサービスへのアクセスが可能となる。この研究では、これを用いて機械学習を行った。

研究が行われたとき、アクセス可能なマシンの数は有料プランでのみアクセス可能となるテスト環境である。すでに回答期間の終了した retired machine が 148、無料でアクセス可能な active machine が 20 であった。ただし、規則上回答を公開してよいのは retired machine のみとなっている。そのため、研究において機械学習に利用できるのはこの retired machine のみである。また、metasploit 内にあるアプリケーションのみでは、retired machine のうちでは 10 個のみであり、他のマシンにはクリック、ログインなどが必要となるため、より多くのツールが必要となる。そのため、学習に使用できたのはこの 10 個のマシンのみであった。

2.2.4.6　学習結果

10 個のマシンを用いて学習を行ったところ、作成されたフレームワークは 33%の正確性で 20 個の exploit の中から正しいものを見つけることに成功した。これは、二値分類ではなく多数のクラスへと分類するというタスクを考えると優れた精度であると述べられている。また、最終的な決定木の一例は図 2.11 に示される。これにより、

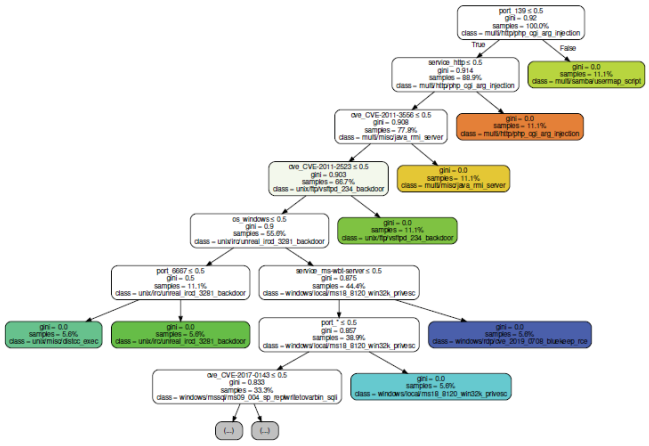


図 2.11: 決定木の一例

ペネトレーションテストの際の難易度を測り、ペネトレーションテスト担当者がターゲットを簡単に悪用できるかどうかを確認するために使用できるとしている。

2.2.5　強化学習とデータ学習手法を用いたラテラルムーブメントへの応用

第一章において、ペネトレーションテストとレッドチームについて述べたが、レッドチームアセスメントにおいては頻繁に、ペネトレーションテストにおいてもしばしば必要となってくるのが第一章で述べた Post-Exploitation である。これにはラテラルムーブメント (横展開)、クレデンシャルの取得などが含まれる。これらは deep exploit によって処理される、アプリケーションと開いたポートの状態により定義されるものとは異なり、状態遷移の存在する連続的なタスクであるため、深層強化学習がより有効と考えられる。前田龍星氏らは、これを A2C 強化学習アルゴリズムと powershell empire を用いて自動化するタスクを設定し、Q 学習・SARSA による学習と比較して、深層強化学習の優位性を示した。[34]

2.2.5.1　A2C 強化学習アルゴリズムについて

A2C は Deep exploit の項で述べた A3C から派生したアルゴリズムである。これは、A3C と同様方策を出力する Actor と、状態価値を出力する critic の二つを同時に扱う Actor-Critic、より確かな状態価値を取得するために先まで進めて行動を行う advantage の二つの手法はそのまま用いているが、分散学習 (Asynchronous) の部分は除外されている。ただし、経験の分散収集はそのまま行われ、一元的にまとめて学習を行うことで、学習効率のみを高める形に変更されている。

また、この研究では A2C アルゴリズムの他に代表的手法である Q 学習、policy-base の代表的手法である SARSA を用い、手法の比較と評価を行っている。

2.2.5.2　強化学習タスクの設定

- 状態空間
  状態空間は図 2.12a によって示される 10 種類のエントリによって表される。

- 行動空間
  行動空間として用いられるのは、powershell empire フレームワークである。powershell empire framework は第一章で述べたとおり、powershell を用いて様々なラテラルムーブメントを行えるフレームワークである。これに登録されているモジュールを行動空間とする。モジュールの一覧は図 2.12b に示されている。これらのうち、特に重要となってくるのは情報収集を行うキーロガーなどのモジュールを集めた Collection、mimikatz 等を用いて資格情報やトークンの収集を行う Credential、WMI や PS Command、DCOM 等を用いて、別のコンピュータへの移動を行う lateral movement モジュール群である。

- 報酬の設定
  このキャンペーンにおいては、報酬は lateral movement グループに分類されるモジュールを用いてラテラルムーブメントを行い、成功した場合に与えられるように設計されている。また、ラテラルムーブメント成功に際しても、新たなアカウントにおいて権限を得られなければ意味がない。そのため、新たなアカウントの制御を獲得できているかどうかで報酬に差をつける。また、同じ権限のアカウントを取得しても、ネットワーク内で行える行動の種類は同じとなる。より高いアカウントの制御を取得した場合はより価値の高いリソースにアクセスできる可能性が高いため、成功の価値が高くなる。よって、この試行においては、価値が小さければ r=10、大きければ r=50 とし、その他のモジュールは成功の可否に関わらず r=-1 と設定した。これは、実際の攻撃者が侵害の露見を避けるため目標を可能な限り早く達成しようとする状況に対応するとしている。

- 学習環境の設定
  学習を行うための環境を実環境で大量に用意するのは現実的ではない。この問題に対処するため、ネットワークの構成や攻撃に対する弱点が偏らず、一般性を持つように学習環境を設計した。具体的にはデータ拡張の手法により、ネットワーク設定にノイズを加え、モジュールの成功確率が 20%、50%、80%、全てランダム、モジュールの特性に応じて成功確率を一定範囲に定めた場合の合計 5 パターンに分けることで環境に多様性を持たせた。

- 強化学習エージェントの設定
  A2C の構成は中間層 3 層、入力層と出力層を合わせて全体で 5 層とした。また、ε-グリーディー法を用いて最初はランダムに行動選択を行い、学習の進行に比例して exploit を選択するように設定した。

表 1　状態の定義

エントリ名	概要
Computers Found	発見したコンピュータの数
Admin Access	Local Admin Access の有無
Compromised Computers	侵害済みコンピュータの数
Previous Module	前ステップで選択した行動
Admin	一般ユーザか，その他か
User Name	ユーザ名の取得有無
Password	平文の資格情報を取得有無
Hash	ハッシュ化された資格情報の取得有無
Rhost	脆弱なホストの有無
DCOM	動作中の DCOM Application の有無

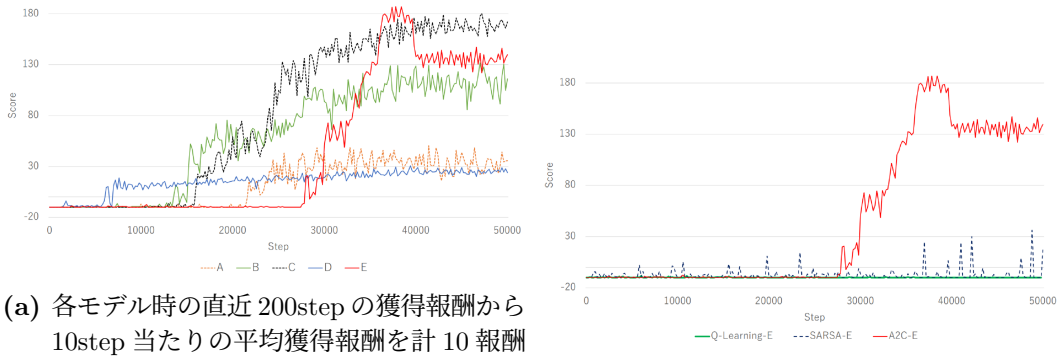
(a) 状態空間

表 2　PowerShell Empire のモジュールの分類

グループ名	登録数	グループ名	登録数
Code Execution	6	Management	30
Collection	24	Persistence	18
Credentials	23	Privesc	22
Exfiltration	2	Recon	3
Exploitation	3	Situational Awareness	52
Lateral Movement	12	Trollsploit	9

(b) 行動空間

図 2.12: powershell empire を用いた Post-exploitation 自動化実験の設定



(a) 各モデル時の直近 200step の獲得報酬から 10step 当たりの平均獲得報酬を計 10 報酬合計

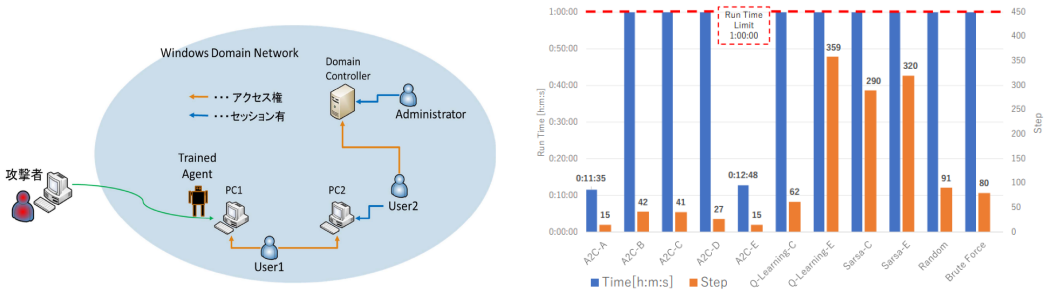
(b) SALSA 及び Q 学習との比較

図 2.13: powershell empire を用いた学習

2.2.5.3　実験結果

図 2.13a は、それぞれのネットワーク設定に対する報酬の推移を示している。A、E はスコアの収束した値からラテラルムーブメントへの行動を学習できているが、B、C、D では変動が激しく、学習終了間際でも 0 への収束がみられない。また、図 2.13b は A2C に加え、Q 学習と SARSA を実装して学習を行った。結果、報酬は A2C の方が大きくなることが分かった。

2.2.5.4　実証



(a) 実証用ネットワーク

(b) タスク終了までにかかる時間

図 2.14: powershell empire を用いたラテラルムーブメントの実証

学習済みのエージェントを用いて、図 2.14a に示すネットワークでの実証を行った。この環境では、エージェントは PC1 から行動を開始する。この構成においてエージェントがドメインコントローラー (最高の権限) を取得するためには、端末間の移動を 2 回、ユーザー間での行動を 2 回行う必要がある。図 2.14b にその結果を示す。結果として、A2C-A と A2C-E のみが最終的に一時間の制限時間の間にタスクを成功させることができた。

2.3　先行研究の問題点

本章で述べた通り、先行研究においては様々な形でペネトレーションテストの自動化・効率化というタスクを行い、その有用性を示している。しかし、二通りの自動化方法にはそれぞれの問題点が存在している。

1. attack graph を用いた自動化
   グラフ上の位置という形で状態の遷移の概念を取り入れているが、実際のツールなどを用いた現実的な攻撃方法までは考慮されておらず、ユーザーが主動で行う必要が生じてしまう。

2. 直接的にツールなどの exploit を自動化するアプローチ
   実際に使用するツールや攻撃の手法を考慮に入れ、より現実的な攻撃の自動化・効率化へのアプローチをしているが、ペネトレーションテストの過程で発生する状態遷移の概念を取り込みきれていない。遷移の概念に着目した [34] においても、対象が Post-exploitation に限られ、また一つのツールで賄える攻撃手法のみに限定されてしまっている。

実際の作業をユーザーが行うのではなく現実における複数の具体的なツールなどを用いての多種類の攻撃手段を自動で行えるよう実装し、かつペネトレーションテストの過程において発生する各種の状態遷移に着目する、両方を行った研究は存在していない。

本論文では、実際に存在する exploit およびツールの情報を利用しながら、状態遷移の概念に着目し深層強化学習による処理を行うフレームワークを提案する。

Chapter 3　アプリケーションのバージョンを利用したexploitの自動選出

3.1　タスク設定

前項で説明したペネトレーションテストの問題点・関連研究を踏まえ、行った実験を提示する。今回着目して作成・実験を試みたのは、Webアプリケーションに対するexploitの自動化である。Webアプリケーションに対する攻撃ツールは多種多様なものがあり、Artem氏らが述べているようにツールや攻撃方法の選択に煩雑な手間を必要とする。そのため、いくつかのexploitを統合し、既存のアプリケーション15種類に対して深層強化学習を用いて攻撃手段を選択・実行するようなフレームワークを提案する。第一の実験として、深層強化学習と単純化された状態空間を用いて、学習に必要な時間・エピソード数のスケール等を割り出す実験を行った。

3.1.1　実験における問題点

深層強化学習はdeep learningを用いた強化学習であり、与えられた状態に対して行動し、得られた報酬が最大化するような行動を行わせる機械学習手法である。この場合、本来であれば学習環境には実際に存在するアプリケーションを、行動には直接ツールを用い、結果から成功と失敗を判定して報酬を与える形で学習を行うことが必要だと考えられるが、以下に示す二つの理由のため難しい。

- exploitなど、ツールの実行に時間がかかる場合が多い

  ツールの実行の際にはターゲットとなるアプリケーションと通信を行うことになる。これには数秒かかる場合もある。そのため、実用的にこれらに対して学習を行わせ、多数のアプリケーションに適応するためには大きな時間がかかると推測される。

- 学習のための環境を多数セットアップするのが現実的ではない

  [34]でも述べられているように、学習のための環境を実際に多数用意するのは難しい。例として、phpmyadminに対するexploitである「phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution」が有効なバージョンは合計で62個存在するとされる。対して、このアプリケーションの合計バージョン数は230個存在する。学習を行ったバージョンのみならず、他のバージョンに対しても正しく判断できるようにするためには、相当数のアプリケーションが必要となり、学習のための環境を圧迫しやすくなる。また、一つ一つのアプリケーションごとに多数の環境を構築する労力も大きなものとなる。以上の理由から、実際のアプリ

ケーションを用いて学習を行った場合、取り扱えるアプリケーションの数は減ってしまうと考えられる。一方で、アプリケーションのバージョン自体は有限かつ基本的に既知である。(ただし、exploit の機能するバージョンに関しては、必ずしも exploit を提示しているデータベースの表記通りとは限らない)

代案として、アプリケーションの各バージョンを模擬し、実行された exploit などの action に対して机上で成功と失敗を判断する仮想的環境を構築し、それを用いて学習を行った後に、実証用の環境上で実際にツールなどを用いて正しく機能することを確認する方式を取った。

3.1.2　実験概要

作成する環境の概略を図 3.1 に示す。

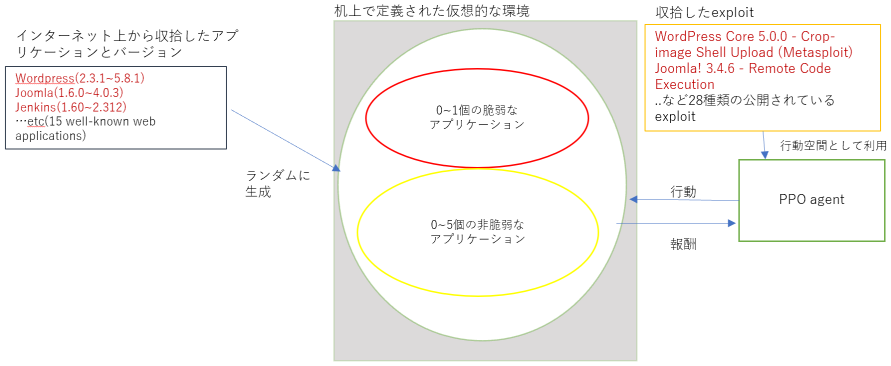


図 3.1: 定義する仮想的環境の概略

まずは 15 種類の Web アプリケーションを決め、それぞれの Web アプリケーションが持つバージョンを公式 Web サイトなどから取得する。次いで、各アプリケーションに対して exploit-db[35] が定義している exploit を調べ、その中から 1-4 個を選択した。これら集めた情報をもとにアプリケーションのバージョンを状態空間、対応する exploit や調査などを行動空間とした Web に対するペネトレーションテストのシミュレーション環境を仮想的に定義し、深層強化学習を用いて高い exploit 成功率を目指すことで、このタスクが深層強化学習で取り扱いが可能なレベルであるかどうかについて検証を行うことを目的とした。

3.1.3　状態空間

シミュレーション環境は、各学習エピソード開始時に初期化される際、15 種類の既存のアプリケーションを仮想的に定義する。次いで、15 種類のアプリケーションのいずれかが脆弱である環境と、「何も脆弱性がない」環境の計 16 種類の環境の中から 1 つがランダムに選択され、脆弱性のある場合は脆弱なバージョンが選択される。その後、さらに 0-5 個の非脆弱なアプリケーションがランダムに選択される。この時点で、

シミュレーション環境の中身は「脆弱なアプリケーション 0-1 つ」と、「非脆弱なアプリケーション 0-5 つ」となる。定義されたアプリケーションとそのバージョンの種類、脆弱なバージョンと非脆弱なバージョンの数は表 3.1 のようになる。

表 3.1: 実験 1 において定義した Web アプリケーション

アプリケーション名	総バージョン数	脆弱なバージョン数	非脆弱なバージョン数
wordpress	288	130	158
jenkins	912	599	313
joomla	140	34	106
drupal	304	188	116
phpmyadmin	230	65	165
jboss	43	35	8
getsimple	40	39	1
webmin	64	57	7
atutor	34	34	0
thinkphp	69	56	13
vbulletin	75	26	49
sugarcrm	109	30	79
couchdb	19	16	3
apache struts	74	64	10
apache tomcat	250	147	103

3.1.4　観測状態 (observation)

次いで、学習エージェント側から観測できる観測状態 (observation) を定義する。まず、自動で最初のスキャンツールによるスキャンが行われたものとして、各アプリケーションの有無だけが機械学習エージェントから確認できる観測状態として定義される。強化学習エージェントから最初に見られる状態空間は、存在するアプリケーションのバージョン分を 0、存在しない場合-1 とされる。その後、action の経過に伴い、各アプリケーションのバージョンへと変更されてゆく。

3.1.5　行動空間

行動空間として定義されるのは各アプリケーションに 1 種類ずつ定義された scan action と、各種の exploit action、及び give up action の 3 種類である。

- scan action

  この action が実行されると、アプリケーションのバージョン情報が取得され、観測状態 (observation) へとマッピングされる。例として、wordpress 4.9.8 に対して実行された場合、[0.4 0.9 0.8] として観測状態にマッピングされる。なお、詳

しくは後述するが、必ずしも現実では無条件にバージョンが取得出来るとは限らずユーザー認証などが必要な場合がある。今回は不要なものとしている。この action における成功とは、定義されているアプリケーションに対して行われ、バージョンを取得し観測状態にセットすることができた場合を指し、失敗はアプリケーションが開いていないなどで取得できなかった時を指す。

- exploit action

  脆弱性に対する exploit に相当する action。各 exploit ごとに対象となるバージョンを持ち、対象となるアプリケーションのバージョンがそれに含まれていれば成功、含まれていなければ失敗となる。また、成功した場合その時点で試行は終了となるが、失敗した場合は継続する。

- give up action

  「この環境には脆弱性が存在しない」ということを決定するアクションである。実行されると即座に試行が終了となり、脆弱性が存在しない環境であった場合のみ成功、存在する環境では失敗となる。ペネトレーションテストにおいては、余計な通信や同一通信の繰り返しが異常検知を作動させてしまうリスクを高めるため、できる限り不要な通信を避けることが重要になる。そのため、途中で不可能とみた場合試行を打ち切ることの重要性は高くなる。この action をうまく使うことで、実行する exploit をできるだけ的確に絞るのが望ましい。

定義した action の情報は表 3.2 のようになる。

なお、exploit の対象バージョンに対しては、exploit-db などで述べられているバージョンをそのまま用いている。例えば「Drupal < 8.6.9」などの場合、そのまま受け取ると初リリースの時点から 8.6.9 までの全てに脆弱性が存在することとなる。ただし、実際は「特定のバージョンから新しく追加された機能に内在する脆弱性」の場合、下限となるバージョンが存在する可能性もある。しかし、全てのアプリケーションの全てのバージョンに対し検証を行うことは現実的でないため、仮に表記に従うこととしている。

3.1.6　報酬

action ごとの報酬を表 3.3 に示す。全ての action において、同じものを繰り返した際には失敗となり、報酬は-1 となる。

表 3.2: 実験 1 における行動空間

アクション番号	action のターゲット	action のタイプ	exploit 名	脆弱性の対象バージョン数
0	wordpress	scan		
1	wordpress	exploit	WordPress 5.0.0 - Image Remote Code Execution	130
2	jenkins	scan		
3	jenkins	exploit	Jenkins < 1.650 - Java Deserialization	599
4	joomla	scan		
5	joomla	exploit	Joomla! 3.4.6 - Remote Code Execution	34
6	drupal	scan		
7	drupal	exploit	Drupal < 8.3.9 / < 8.4.6 / < 8.5.1　'Drupalgeddon2' Remote Code Execution	163
8	drupal	exploit	Drupal < 8.6.9 - REST Module Remote Code Execution	66
9	phpmyadmin	scan		
10	phpmyadmin	exploit	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	62
11	phpmyadmin	exploit	phpMyAdmin 4.8.1 - Local File Inclusion to Remote Code Execution	3
12	jboss	scan		
13	jboss	exploit	JBoss AS 3/4/5/6 - Remote Command Execution	35
14	getsimple	scan		
15	getsimple	exploit	GetSimpleCMS Unauthenticated RCE	38
16	getsimple	exploit	GetSimpleCMS PHP File Upload Vulnerability	33
17	webmin	scan		
18	webmin	exploit	Webmin 1.920 - Unauthenticated Remote Code Execution (Metasploit)	4
19	webmin	exploit	Webmin 1.962 - 'Package Updates' Escape Bypass RCE (Metasploit)	57
20	webmin	exploit	Webmin 1.910 - 'Package Updates' Remote Command Execution (Metasploit)	46
21	webmin	exploit	Webmin 1.580 - '/file/show.cgi' Remote Command Execution (Metasploit)	1
22	atutor	scan		
23	atutor	exploit	ATutor 2.2.4 'language_import' Arbitrary File Upload / RCE [CVE-2019-12169]	34
24	atutor	exploit	ATutor 2.2.1 - SQL Injection / Remote Code Execution	31
25	atutor	exploit	ATutor 2.2.1 - Directory Traversal / Remote Code Execution (Metasploit)	31
26	thinkphp	scan		
27	thinkphp	exploit	ThinkPHP 5.0.23/5.1.31 - Remote Code Execution	55
28	thinkphp	exploit	ThinkPHP - Multiple PHP Injection RCEs (Metasploit)	4
29	vbulletin	scan		
30	vbulletin	exploit	vBulletin widgetConfig RCE	26
31	vbulletin	exploit	vBulletin 5.1.2 Unserialize Code Execution	8
32	sugarcrm	scan		
33	sugarcrm	exploit	SugarCRM CE 6.3.1 - 'Unserialize()' PHP Code Execution (Metasploit)	5
34	sugarcrm	exploit	SugarCRM 6.5.23 - REST PHP Object Injection (Metasploit)	30
35	couchdb	scan		
36	couchdb	exploit	Apache CouchDB - Arbitrary Command Execution (Metasploit)	16
37	apache struts	scan		
38	apache struts	exploit	Apache Struts 2.0.1 < 2.3.33 / 2.5 < 2.5.10 - Arbitrary Code Execution	57
39	apache struts	exploit	Apache Struts < 1.3.10 / < 2.3.16.2 - ClassLoader Manipulation Remote Code Execution (Metasploit)	45
40	apache tomcat	scan		
41	apache tomcat	exploit	AJP 'Ghostcat' File Read/Inclusion (Metasploit)	110
42	apache tomcat	exploit	Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass / Remote Code Execution (2)	61
43		giveup		

表 3.3: 実験 1 における action の報酬

action の種類	成功時報酬	失敗時報酬
scan	0	-1
exploit	1	-1
give up	1	-5

3.2　PPO アルゴリズム (アプリケーションベースの環境生成) によるタスクの学習と評価

3.2.1　学習アルゴリズム

学習アルゴリズムは PPO[36] を用いた。PPO 学習アルゴリズムは A3C、A2C と同様 Actor と Critic を別々に定義するアルゴリズムであるが、一度の実行で方策が大きく変化しない。また、64 個のプロセスを用いて経験の収集の並列化を行った。別々に取得された行動とその報酬はひとつにまとめられた上でモデルの学習に使用される。
学習データの正規化に際しては、openAI でも利用されている、学習過程で集めたデータをもとに平均と分散を割り出し、徐々にアップデートしていくアルゴリズムを用いた。
ネットワークの構成は以下のようになる。

- モデル構造:3 層

- 入力:62 次元
- 隠れ層 1:64 次元
- 隠れ層 2:64 次元
- 出力:44 次元
- 学習率:2.5e-4
- エージェントの最大行動回数:10 回

3.2.2　学習結果

ある学習プロセスに着目し、1000 回のエピソード内で exploit action もしくは give up を成功させた回数の割合を記録し、3 回の試行の平均をとったグラフを図 3.2 に示す。正しく学習が進み、95%程度で安定して成功していることがわかる。

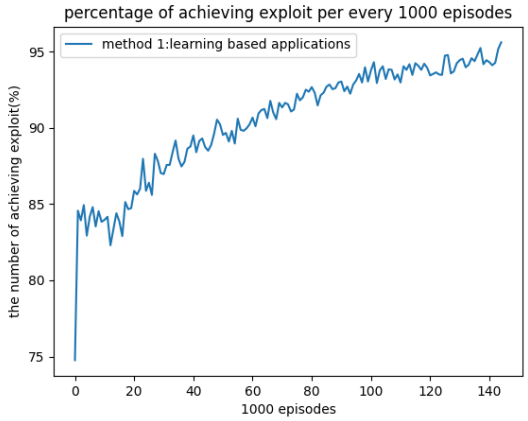


図 3.2: 手法 1 による学習過程

3.2.3　モデルの評価

成功確率が 95%に達した学習済みモデルを 3 つ用いて、評価を行った。評価の際には、ランダムに選択される各 exploit ごとに脆弱なバージョンを選択し、評価指標としては各エピソードの exploit を発見できたエピソード数の総試行内の割合とする。ただし、アプリケーションのバージョンが重複しないものを用いたため、一方の exploit の脆弱な範囲が他方の exploit に包含される場合、その exploit を使用しないこととしている。それぞれの評価環境においては、ランダムに生成される脆弱なアプリケーション 1 つとした single と、脆弱なアプリケーション 1 つと非脆弱なアプリケーション 4 つの 5 つとした five という環境を用い、それぞれ 10000 エピソードを 3 セット行って平均をとっている。結果を表 3.4 に示す。(使用するモデルは、成功率 96 ± 1.1%の範囲に収まるものとした。)

表 3.4: 実験 1・手法 1 における評価

action	vulnerable	vulnerable	single			five		
	target	version	success	failure	give up	success	failure	give up
1	wordpress	130	98.9%	0.0%	1.1%	90.6%	8.8%	0.6%
3	jenkins	599	98.4%	0.1%	1.6%	91.6%	7.1%	1.3%
5	joomla	34	99.4%	0.0%	0.6%	98.4%	1.5%	0.1%
7	drupal	122	98.5%	0.0%	1.5%	92.9%	6.0%	1.1%
8	drupal	32	91.2%	0.2%	8.5%	73.5%	22.1%	4.5%
10	phpmyadmin	62	99.1%	0.0%	0.9%	96.7%	2.4%	0.9%
11	phpmyadmin	3	58.6%	0.0%	41.4%	44.3%	44.2%	11.5%
13	jboss	35	98.2%	0.0%	1.8%	94.5%	3.7%	1.8%
15	getsimplecms	5	99.4%	0.0%	0.6%	91.8%	6.7%	1.5%
19	webmin	11	98.7%	0.0%	1.3%	86.5%	8.7%	4.8%
23	atutor	3	99.6%	0.0%	0.4%	89.6%	6.4%	3.9%
27	thinkphp	52	99.4%	0.0%	0.6%	96.1%	2.8%	1.1%
28	thinkphp	1	0.1%	71.3%	28.6%	0.1%	92.4%	7.5%
30	vbulletin	18	98.9%	0.0%	1.1%	96.7%	2.2%	1.1%
34	sugarcrm	25	99.6%	0.0%	0.4%	94.6%	4.9%	0.6%
36	couchdb	16	98.7%	0.0%	1.3%	89.4%	9.7%	0.9%
38	apache struts	19	95.5%	0.0%	4.5%	84.1%	12.6%	3.3%
39	apache struts	7	98.6%	0.8%	0.6%	86.9%	12.7%	0.5%
41	apache tomcat	86	98.9%	0.0%	1.1%	95.2%	3.9%	1.0%
42	apache tomcat	37	97.4%	0.0%	2.6%	86.5%	10.9%	2.7%
give up			0.0%	11.6%	88.4%	0.0%	74.8%	25.2%

3.2.4　ケーススタディ

現実のアプリケーションを利用した 16 種類の実証環境を用意し、学習を行ったモデルを用いてエージェントが具体的にどのような行動をとるのかについて検証した。モデルとしては成功率 95%に達した学習済みモデルを用いる。

3.2.4.1　目標要件

強化学習を経たエージェントは、以下の動作が可能であることが望ましい。

1. 脆弱なバージョンを判別し、的確な exploit を選択できること
2. 非脆弱なバージョンを識別し、exploit を行わないこと
3. 複数のアプリケーションに対して 1、2 を判断できること。
4. exploit に失敗し脆弱なバージョンが存在していないと判断できたとき、余計な行動を行わずにすぐに give up を行えること。

3.2.4.2　ケーススタディ環境

実証に用いる環境は 8 種類のテーマに対し、合計 16 種類の試行環境を用意した。用意した環境は表 3.5 に示す。

3.2.4.3　実装

脆弱な環境の実装は docker のコンテナを用いて行った。脆弱なアプリケーションをホスティングしているコンテナには、アプリケーションの公式サイトが公開しているコ

表 3.5: 実験 1・ケーススタディ環境

環境	環境の特徴	環境を構成するアプリケーション	各アプリケーションに対応する exploit
1	アプリケーションが存在しない環境	なし	43
2-1	アプリケーションが 1 つ、対象となる脆弱性の範囲が広い環境	joomla 3.4.3	5
2-2		thinkphp 5.1.30	27
3-1	アプリケーションが 1 つ、対象となる脆弱性の範囲が狭い環境	thinkphp 5.0.23	28
3-2		phpmyadmin 4.8.1	11
4-1	アプリケーションが 1 つ、脆弱性が存在しない環境	joomla 4.0.4	43
4-2		wordpress 5.5.7	43
4-3		jenkins 2.137	43
5-1	アプリケーションが 1 つ、本来はバージョン上 exploit が	sugarcrm 6.5.23	43
5-2	効くはずであるが、他の要因によって効かない環境	wordpress 4.9.8	43
6-1	アプリケーションが 5 つ、脆弱性の範囲が広い環境	joomla 3.4.3 phpmyadmin 5.1.1 jenkins 2.137 drupal 8.9.19 wordpress 5.5.7	5
6-2		thinkphp:5.1.30 tomcat 9.0.46 phpmyadmin 5.1.1 wordpress 5.5.7 drupal 8.9.19	27
7-1	アプリケーションが 5 つ、脆弱性の範囲が狭い環境	phpmyadmin 4.8.1 joomla 4.0.4 jenkins 2.137 drupal 8.9.19 wordpress 5.5.7	11
7-2		thinkphp:5.0.23 tomcat 9.0.46 phpmyadmin 5.1.1 wordpress 5.5.7 drupal 8.9.19	28
8-1	アプリケーションが 5 つ、脆弱性を持たない環境	wordpress 5.5.7 joomla 4.0.4 phpmyadmin 5.1.1 jenkins 2.137 drupal 8.9.19	43
8-2		phpmyadmin 5.1.1 tomcat 9.0.46 jenkins 2.137 drupal 8.9.19 wordpress 5.5.7	43

ンテナに加え、vulfocus[37] および vulhub[38] が github で公開しているものを用いた。一方、exploit に関しては exploit-db が公開しているスクリプトを用いるか、metasploit を pymetasploit3 経由で呼び出して使用した。

3.2.4.4　結果

各環境ごとに 10 エピソードの試行を行った結果と、その過程を表 3.6 に示す。

表 3.6: 実験 1・手法 1 のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1	100%	43	10	即座に give up
2-1	100%	4 → 5	10	joomla のバージョンをスキャン→ exploit を実行
2-2	100%	26 → 27	10	thinkphp のバージョンをスキャン→ exploit を実行
3-1	0%	26 → 27 → 27 → 27 → 27 → 27 → 27 → 27 → 27 → 27	10	thinkphp のバージョンをスキャン→誤った exploit を繰り返す
3-2	50%	9 → 11	5	phpmyadmin のバージョンをスキャン→ exploit を実行
		9 → 43	5	phpmyadmin のバージョンをスキャン→ give up
4-1	0%	4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	10	joomla のバージョンをスキャン→ joomla の exploit を実行し続ける
4-2	0%	1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 0	2	exploit → wordpress のバージョンをスキャン→ exploit
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 1	1	
		1 → 1 → 1 → 0 → 1 → 1 → 1 → 1 → 1 → 1 → 1	1	
		0 → 1 → 0 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1	1	
		1 → 1 → 1 → 0 → 1 → 1 → 0 → 1 → 1 → 1 → 1	1	
		1 → 0 → 1 → 1 → 1 → 0 → 1 → 1 → 1 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	1	
		1 → 0 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	1	
4-3	100%	2 → 43	8	jenkins のバージョンをスキャン→ give up
		3 → 2 → 43	1	jenkins を exploit →バージョンをスキャン→ give up
		2 → 3 → 43	1	バージョンをスキャン→ jenkins を exploit → give up
5-1	0%	32 → 34 → 34 → 34 → 34 → 34 → 34 → 34 → 34 → 34 → 34	8	sugarcrm のバージョンをスキャン→誤った exploit を続ける
		34 → 32 → 34 → 34 → 34 → 34 → 34 → 34 → 34 → 34 → 34	2	
5-2	0%	1 → 1 → 1 → 0 → 1 → 1 → 1 → 1 → 1 → 1 → 1	2	wordpress の exploit → scan → exploit を続ける
		1 → 0 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	2	
		0 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	2	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 0	1	
		0 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 1	1	
		1 → 1 → 1 → 0 → 1 → 1 → 0 → 1 → 1 → 1 → 1	1	
6-1	100%	2 → 9 → 4 → 5	2	各アプリケーションのスキャン→ joomla の exploit
		9 → 2 → 4 → 5	2	
		9 → 4 → 5	2	
		9 → 4 → 6 → 5	2	
		6 → 4 → 5	1	
		9 → 6 → 4 → 5	1	
6-2	100%	26 → 27	3	thinkphp のスキャン→ exploit
		9 → 40 → 26 → 27	1	
		40 → 9 → 26 → 27	1	
		6 → 40 → 9 → 26 → 27	1	
		9 → 40 → 6 → 26 → 27	1	
		40 → 9 → 6 → 26 → 27	1	
		9 → 40 → 26 → 6 → 27	1	
		9 → 40 → 6 → 26 → 7 → 27	1	
7-1	0%	9 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	4	各アプリケーションのバージョンをスキャン→ joomla の exploit を行い続ける
		6 → 9 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	2	
		9 → 6 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	1	
		2 → 9 → 6 → 1 → 4 → 5 → 5 → 5 → 5 → 5 → 5	1	
		2 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 6 → 5	1	
		6 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 7	1	
7-2	0%	26 → 40 → 27 → 9 → 27 → 27 → 27 → 27 → 27 → 27 → 27	1	各アプリケーションのバージョンをスキャン→ thinkphp と wordpress の
		26 → 40 → 9 → 27 → 27 → 27 → 27 → 27 → 1 → 27	1	exploit を行い続ける
		26 → 40 → 9 → 27 → 42 → 27 → 27 → 27 → 27 → 6	1	
		40 → 9 → 26 → 27 → 27 → 27 → 27 → 27 → 27 → 27 → 27	1	
		9 → 40 → 0 → 6 → 0 → 1 → 26 → 1 → 27 → 1	1	
		9 → 40 → 6 → 26 → 27 → 6 → 27 → 27 → 42 → 1	1	
		26 → 9 → 27 → 40 → 27 → 27 → 27 → 27 → 27 → 6	1	
		40 → 26 → 9 → 1 → 27 → 1 → 1 → 27 → 27 → 27	1	
		26 → 9 → 27 → 27 → 27 → 27 → 27 → 6 → 40 → 1	1	
		40 → 6 → 9 → 26 → 27 → 27 → 27 → 1 → 0 → 1	1	
8-1	0%	9 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	4	各アプリケーションのバージョンをスキャン→ joomla の exploit を行い続ける
		6 → 9 → 4 → 5 → 5 → 5 → 5 → 5 → 4 → 5 → 5	1	
		2 → 4 → 5 → 5 → 5 → 9 → 5 → 5 → 5 → 5 → 5	1	
		9 → 6 → 2 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5	1	
		4 → 5 → 5 → 5 → 5 → 9 → 5 → 5 → 5 → 5 → 5	1	
		9 → 2 → 6 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5	1	
		9 → 2 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	1	
8-2	0%	2 → 9 → 40 → 1 → 1 → 3 → 6 → 42 → 42 → 42	1	各アプリケーションのバージョンをスキャン→ wordpress、
		40 → 9 → 6 → 1 → 1 → 1 → 1 → 2 → 1 → 1	1	tomcat の exploit
		2 → 9 → 40 → 3 → 1 → 1 → 6 → 42 → 1 → 42	1	
		2 → 9 → 40 → 6 → 42 → 1 → 42 → 1 → 3 → 42	1	
		9 → 40 → 6 → 1 → 7 → 42 → 2 → 0 → 1 → 1	1	
		40 → 9 → 2 → 6 → 42 → 1 → 0 → 42 → 42 → 1	1	
		2 → 40 → 9 → 6 → 42 → 42 → 6 → 42 → 7 → 1	1	
		9 → 40 → 2 → 0 → 6 → 1 → 0 → 42 → 2 → 1	1	
		2 → 9 → 40 → 6 → 1 → 7 → 0 → 42 → 0 → 42	1	
		40 → 9 → 6 → 2 → 1 → 0 → 41 → 0 → 1 → 42	1	

3.2.5　考察

学習の過程は、最終的に 95%まで成功率を高めることができていた。このことから、タスクの複雑さとしては問題なく学習が可能であるレベルであると考えられる。[1]

モデルの評価においては、アプリケーションが一つだけの single では基本的に exploit を達成できていたが、複数のアプリケーションが存在している five では成功率が下がり、また give up の失敗率が 70%を上回り極めて高くなった。

ケーススタディにおいては、scan action を行い、対象のバージョンを確かめてから exploit action を行っていた。一方で、exploit に失敗した場合でも延々と同じ行動を繰り返し、行動回数の限界に達してしまうことが非常に多かった。環境 2、6 のように、想定解となる exploit が十分広い範囲に作用するような環境に対しては exploit を成功させることができていたが、3、7 のように狭い環境に対してはうまく機能しないということ、そして脆弱性のない環境でも give up は実行されにくいということが分かった。

原因としては、学習データのバイアスが原因と推測される。脆弱性の及ぶ範囲が広いほど、ランダムに生成される学習環境では多く出現するが、一方で範囲の狭い exploit は出現する確率が低い。そのため、学習が進みづらいと考えられる。

達成できた目標要件は 1 のみとなった。現状では範囲の広い exploit を見つけることはできるが、脆弱性のない環境や対象の少ない exploit は不安定になる。そのため、これを解消するための実験を行う。

[1] PPO アルゴリズムとしても様々なテクニックが存在し、その採用の有無などコーディングスタイルによって学習の結果に多少の違いは出ると推測されるが、本研究における結果と考察に影響するほどのレベルではないと考えられる。後に記述する実験においてもこれらは同様とする。

3.3　PPO アルゴリズム (exploit ベースの環境生成) によるタスクの学習と評価

前項の実験の結果を受けて、学習バイアスをなくすためにまず出現するアプリケーションのバージョンについて、exploit を基準に均等化しての学習を行った。その後、学習済みモデルを用いて実験 2 と同様の環境に対して実証を試みた。

3.3.1　実験設定

基本的には実験 1 の環境と同じであるが、アプリケーションの選択の際、脆弱なバージョンをランダムに選択するのではなく、想定解となる exploit を一つ選択し、それに対して脆弱なアプリケーションのバージョンをランダムに選択する形とする。これを exploit base の環境選出と呼称する。なお、ノイズとする非脆弱なアプリケーションの選び方は従来通りとなる。また、アプリケーションではなく exploit を基準に環境を決めているため、非脆弱な状態が生成される確率は 1/16 から 1/28 へと下がる。

3.3.2　学習結果

手法 1 と同様の設定で学習を行った。ある学習プロセスに着目し、エピソード内で exploit action もしくは give up を成功させた回数の割合を記録し、3 回の試行の平均をとった上で、手法 1 と比較したグラフを図 3.3 に示す。

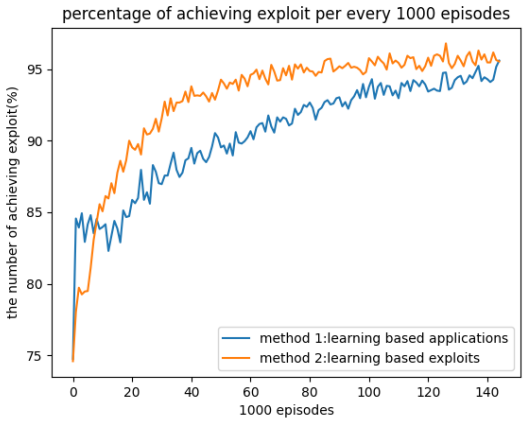


図 3.3: 実験 1・手法 2 における学習過程

3.3.3　モデルの評価

同じく成功確率が 95%に達した学習済みモデルを用いて、評価を行った。結果を表 3.7 に示す。

表 3.7: 実験 1・手法 2 における評価

action	vulnerable	vulnerable	single			five		
	target	version	success	failure	give up	success	failure	give up
1	wordpress	130	97.7%	0.0%	2.3%	77.5%	21.9%	0.6%
3	jenkins	599	98.2%	0.0%	1.8%	75.3%	23.9%	0.8%
5	joomla	34	99.6%	0.0%	0.4%	94.2%	5.7%	0.1%
7	drupal	122	97.7%	0.8%	1.6%	87.1%	12.1%	0.7%
8	drupal	32	98.6%	0.0%	1.4%	78.4%	21.2%	0.4%
10	phpmyadmin	62	98.5%	0.0%	1.5%	88.9%	10.3%	0.8%
11	phpmyadmin	3	99.9%	0.0%	0.1%	95.8%	4.0%	0.2%
13	jboss	35	97.2%	0.0%	2.8%	87.1%	12.2%	0.7%
15	getsimplecms	5	99.9%	0.0%	0.1%	77.9%	20.7%	1.4%
19	webmin	11	98.3%	0.0%	1.7%	72.2%	24.7%	3.1%
23	atutor	3	100.0%	0.0%	0.0%	92.6%	6.3%	1.1%
27	thinkphp	52	98.8%	0.1%	1.1%	94.0%	5.6%	0.5%
28	thinkphp	1	99.9%	0.0%	0.1%	97.2%	2.7%	0.1%
30	vbulletin	18	93.9%	0.0%	6.1%	93.8%	5.7%	0.5%
34	sugarcrm	25	99.7%	0.0%	0.3%	92.1%	7.6%	0.3%
36	couchdb	16	98.9%	0.0%	1.1%	75.2%	24.5%	0.3%
38	apache struts	19	92.9%	0.0%	7.1%	64.1%	33.1%	2.8%
39	apache struts	7	99.9%	0.1%	0.0%	75.7%	24.1%	0.2%
41	apache tomcat	86	99.5%	0.0%	0.5%	93.3%	6.4%	0.4%
42	apache tomcat	37	94.0%	0.0%	6.0%	87.7%	11.7%	0.7%
give up			0.0%	16.8%	83.2%	0.0%	90.9%	9.1%

3.3.4　ケーススタディ

手法 1 と同様にケーススタディを行った。結果を表 3.8 に示す。

3.3.5　考察

学習速度は手法 1 と比べて、やや早くなった。また、exploit をアプリケーション選択の基準としたことで、少ないアプリケーションであっても正しく選択させることには成功した。しかし、give up を正しくできない問題はより深刻となった。これは、give up が必要となる環境の生成される可能性が小さくなったためと推測される。

表 3.8: 実験 1・手法 2 のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1	100%	43	10	即座に give up
2-1	100%	4 → 5	9	joomla のスキャン→ exploit
		0 → 4 → 5	1	
2-2	100%	26 → 27	9	thinkphp のスキャン→ exploit
		0 → 26 → 27	1	wordpress と thinkphp のバージョンをスキャン→ thinkphp の exploit を実行
3-1	100%	26 → 28	10	thinkphp のスキャン→ exploit
3-2	100%	9 → 11	10	phpmyadmin のスキャン→ exploit
4-1	0%	4 → 5 → 5 → 5 → 0 → 5 → 5 → 5 → 5 → 5	3	joomla のスキャン→ exploit を続ける
		4 → 5 → 5 → 5 → 5 → 5 → 0 → 5 → 5 → 5	2	
		4 → 5 → 5 → 5 → 5 → 0 → 5 → 5 → 5 → 5	2	
		4 → 5 → 5 → 5 → 5 → 5 → 5 → 0 → 5 → 5	2	
		0 → 4 → 5 → 5 → 5 → 5 → 5 → 5 → 5 → 5	1	
4-2	100%	1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 43	5	wordpress の exploit → scan → give up
		1 → 1 → 1 → 1 → 1 → 1 → 0 → 43	3	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 43	1	
		1 → 1 → 1 → 1 → 0 → 43	1	
4-3	100%	2 → 43	8	jenkins の scan → give up
		3 → 2 → 43	1	
		0 → 2 → 43	1	
5-1	0%	34 → 32 → 34 → 34 → 34 → 34 → 34 → 0 → 34 → 34	3	sugarcrm のバージョンをスキャン→ exploit を続ける
		34 → 34 → 32 → 34 → 34 → 34 → 0 → 34 → 34 → 34	2	
		34 → 32 → 34 → 34 → 34 → 0 → 34 → 34 → 34 → 34	1	
		34 → 34 → 34 → 34 → 34 → 34 → 0 → 34 → 34 → 34	1	
		34 → 34 → 34 → 34 → 34 → 0 → 34 → 34 → 34 → 34	1	
		34 → 34 → 32 → 34 → 34 → 34 → 0 → 34 → 34	1	
		32 → 34 → 34 → 34 → 34 → 34 → 34 → 0 → 34 → 34	1	
5-2	30%	1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 1	3	wordpress の exploit を行い、スキャンを 1 度だけ挟む
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 1	1	
		1 → 1 → 1 → 1 → 0 → 1 → 1 → 1 → 1 → 1	1	
		1 → 1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 43	1	wordpress の exploit を行い、スキャンを 1 度だけ挟み、give up を行う
		1 → 1 → 1 → 1 → 1 → 0 → 1 → 1 → 43	1	
		1 → 1 → 1 → 1 → 1 → 1 → 1 → 0 → 43	1	
6-1	100%	6 → 9 → 2 → 4 → 5	2	各アプリケーションのスキャン→ joomla の exploit
		9 → 6 → 2 → 4 → 5	1	
		6 → 9 → 3 → 2 → 1 → 1 → 1 → 0 → 4 → 5	1	
		6 → 9 → 2 → 1 → 4 → 5	1	
		6 → 9 → 2 → 1 → 3 → 3 → 0 → 4 → 5	1	
		0 → 9 → 6 → 2 → 4 → 5	1	
		9 → 6 → 2 → 1 → 4 → 5	1	
		6 → 9 → 4 → 2 → 5	1	
		9 → 6 → 2 → 1 → 5	1	
6-2	100%	26 → 27	9	thinkphp のスキャン→ exploit
		6 → 26 → 27	1	drupal と thinkphp のスキャン→ exploit
7-1	100%	6 → 9 → 11	5	drupal、phpmyadmin のスキャン → phpmyadmin の exploit
		9 → 11	3	phpmyadmin の scan → exploit
		2 → 6 → 9 → 11	2	jenkins、drupal、phpmyadmin のスキャン → phpmyadmin の exploit
7-2	90%	26 → 28	8	thinkphp のスキャン→正しい exploit
		26 → 27 → 28	1	thinkphp のスキャン→間違った exploit →正しい exploit
		0 → 26 → 27 → 27 → 27 → 27 → 27 → 27 → 27 → 27	1	thinkphp のスキャン→間違った exploit を繰り返す
8-1	0%	6 → 9 → 2 → 4 → 5 → 5 → 0 → 5 → 5 → 5	4	各アプリケーションのスキャン→ joomla の exploit を繰り返す
		6 → 9 → 2 → 1 → 4 → 5 → 5 → 0 → 5 → 5	2	
		6 → 2 → 9 → 4 → 5 → 0 → 5 → 5 → 5 → 5	1	
		9 → 6 → 2 → 4 → 5 → 5 → 5 → 0 → 5 → 5	1	
		9 → 6 → 2 → 1 → 4 → 5 → 5 → 0 → 5 → 5	1	
		6 → 9 → 2 → 4 → 0 → 5 → 5 → 5 → 5 → 5	1	
8-2	0%	40 → 2 → 6 → 9 → 1 → 1 → 1 → 1 → 0 → 2	1	各アプリケーションのスキャン→ jenkins、wordpress の exploit を繰り返す
		40 → 6 → 3 → 0 → 9 → 3 → 2 → 2 → 3 → 3	1	
		40 → 6 → 9 → 2 → 0 → 3 → 3 → 2 → 3 → 3	1	
		9 → 6 → 40 → 2 → 1 → 1 → 0 → 3 → 3 → 3	1	
		40 → 6 → 2 → 9 → 0 → 2 → 1 → 3 → 2 → 2	1	
		40 → 6 → 9 → 0 → 3 → 2 → 3 → 3 → 2 → 3	1	
		40 → 6 → 2 → 9 → 1 → 1 → 1 → 0 → 3 → 3	1	
		40 → 6 → 9 → 2 → 1 → 0 → 3 → 3 → 2 → 3	1	
		40 → 6 → 9 → 2 → 1 → 1 → 1 → 0 → 3 → 3	1	
		40 → 9 → 6 → 2 → 0 → 3 → 3 → 3 → 2 → 3	1	

3.4　PPO-LSTM アルゴリズムによるタスクの学習と評価

LSTM は、リカレントニューラルネットワークの一種であり、時系列データを処理する際に有効である。これを PPO 強化学習アルゴリズムと組み合わせることによって、exploit が見つからない場合などを認識して give up で試行を終わらせることができるかどうかを調べる。

3.4.1　ニューラルネットワークの設定

LSTM(Long-short-term memory) は、前回の層を利用することで過去のデータの影響を持ち越す再帰的ニューラルネットワーク (リカレントネットワーク) の一種であり、長期間の依存性を記録することができる。これによって時系列データを学習することができる。今回は PPO アルゴリズムにおいて、中間層を LSTM に置き換えることで、「前の行動による結果」を学習し、より柔軟な行動ができるのではないかと考えた。

3.4.2　実験設定

学習において、2 種類の設定を用意した。

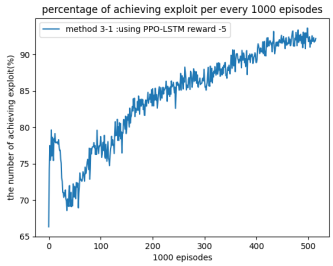
- 手法 3-1:give up の失敗時報酬が-5
- 手法 3-2:give up の失敗時報酬が-2

give up の失敗時報酬が小さいほど、失敗を重ねるより早く give up を行ったほうが合計報酬が小さくなるという理由から give up を行いにくくなる。学習段階で give up が多くなりすぎた場合、エピソードが途中で終了されることが多いため、得られる経験の数が少なくなり、学習のバランスが悪くなることが多い。

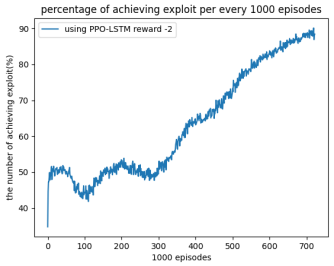
PPO のみで失敗時報酬を小さくして学習を行った場合、特定のアプリケーションに対する行動が成功しなくなることが多く学習過程が不安定になりやすかったが、LSTM で行った場合はこのような事態が起こらなかったため、give up の報酬に差をつけて比較を行うこととした。

3.4.3　学習結果

図 3.4 に，このアルゴリズムで学習を行う過程のある 1 つのプロセスの 1000 回中の exploit の成功率の、3 回の試行における平均の遷移を示す。



(a) 手法 3-1 による学習過程



(b) 手法 3-2 による学習過程

図 3.4: 手法 3-1、3-2 による学習過程

3.4.4　モデルの評価

手法 1、2 と同じく、同じく成功確率が 95%に達した学習済みモデルを用いて、評価を行った。結果をそれぞれ表 3.9、表 3.10 に示す。(ただし、手法 3-2 のみ、他モデルの平均成功率に比べて 0.2%ほど使用したモデルの平均成功率が高くなっているため、ごく僅かだけ結果が良くなりやすいと考えられる。そのため、大きな変化のみに着目する。)

表 3.9: 実験 1・手法 3-1 における評価

action	vulnerable	vulnerable	single			five		
	target	version	success	failure	give up	success	failure	give up
1	wordpress	130	97.5%	0.0%	2.5%	84.8%	14.2%	0.9%
3	jenkins	599	99.7%	0.0%	0.3%	94.4%	5.4%	0.1%
5	joomla	34	99.8%	0.0%	0.2%	95.6%	4.4%	0.1%
7	drupal	122	97.6%	0.0%	2.4%	88.6%	10.3%	1.1%
8	drupal	32	99.0%	0.0%	1.0%	53.2%	44.3%	2.5%
10	phpmyadmin	62	99.0%	0.0%	1.0%	95.0%	4.5%	0.5%
11	phpmyadmin	3	65.2%	15.7%	19.1%	35.5%	61.0%	3.5%
13	jboss	35	99.2%	0.0%	0.8%	91.2%	7.8%	0.9%
15	getsimplecms	5	100.0%	0.0%	0.0%	86.6%	11.7%	1.7%
19	webmin	11	98.4%	0.0%	1.6%	91.8%	6.9%	1.3%
23	atutor	3	99.9%	0.0%	0.1%	86.6%	11.8%	1.6%
27	thinkphp	52	100.0%	0.0%	0.0%	94.1%	5.1%	0.7%
28	thinkphp	1	33.5%	16.8%	49.7%	9.5%	81.7%	8.9%
30	vbulletin	18	99.6%	0.0%	0.4%	90.1%	9.2%	0.7%
34	sugarcrm	25	98.9%	0.0%	1.1%	87.4%	11.4%	1.3%
36	couchdb	16	98.1%	0.0%	1.9%	89.7%	9.2%	1.2%
38	apache struts	19	97.9%	0.5%	1.6%	79.3%	18.0%	2.7%
39	apache struts	7	100.0%	0.0%	0.0%	81.9%	17.6%	0.6%
41	apache tomcat	86	99.0%	0.0%	1.0%	86.3%	13.6%	0.2%
42	apache tomcat	37	99.6%	0.0%	0.4%	77.8%	21.2%	0.9%
give up			0.0%	2.0%	98.0%	0.0%	76.9%	23.1%

表 3.10: 実験 1・手法 3-2 における評価

action	vulnerable	vulnerable	single			five		
	target	version	success	failure	give up	success	failure	give up
1	wordpress	130	96.6%	0.0%	3.4%	81.7%	4.0%	14.3%
3	jenkins	599	97.6%	0.0%	2.4%	91.4%	2.3%	6.3%
5	joomla	34	99.7%	0.0%	0.3%	96.3%	1.0%	2.7%
7	drupal	122	91.1%	0.0%	8.9%	81.2%	4.2%	14.6%
8	drupal	32	93.3%	0.0%	6.7%	36.6%	15.7%	47.7%
10	phpmyadmin	62	95.3%	0.0%	4.7%	89.6%	2.0%	8.4%
11	phpmyadmin	3	63.2%	0.0%	36.8%	25.1%	18.4%	56.5%
13	jboss	35	100.0%	0.0%	0.0%	95.6%	1.1%	3.3%
15	getsimplecms	5	99.9%	0.0%	0.1%	91.4%	3.3%	5.3%
19	webmin	11	99.9%	0.0%	0.1%	91.4%	2.1%	6.5%
23	atutor	3	99.8%	0.0%	0.2%	91.9%	2.2%	5.9%
27	thinkphp	52	99.5%	0.0%	0.5%	96.4%	0.7%	2.9%
28	thinkphp	1	31.1%	0.0%	68.9%	2.3%	49.3%	48.5%
30	vbulletin	18	99.7%	0.0%	0.3%	92.3%	2.2%	5.6%
34	sugarcrm	25	97.4%	0.0%	2.6%	84.7%	3.7%	11.7%
36	couchdb	16	98.3%	0.0%	1.7%	83.1%	4.2%	12.7%
38	apache struts	19	98.0%	0.1%	1.9%	84.5%	4.3%	11.2%
39	apache struts	7	99.9%	0.0%	0.1%	93.4%	4.4%	2.2%
41	apache tomcat	86	99.6%	0.0%	0.4%	86.1%	3.4%	10.5%
42	apache tomcat	37	99.1%	0.0%	0.9%	67.2%	9.2%	23.6%
give up			0.0%	0.1%	99.9%	0.0%	24.7%	75.3%

3.4.5　ケーススタディ

手法 1、2 と同じ環境を用いて、ケーススタディを行った。各環境での行動を表 3.11、3.12 にそれぞれ示す。

表 3.11: 実験 1・手法 3-1 のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1	100%	43	10	即座に give up
2-1	100%	4 → 5	8	joomla のスキャン→ exploit
		5	2	スキャンを行わず、直接 exploit
2-2	100%	26 → 27	10	thinkphp のスキャン→ exploit
3-1	0%	26 → 27 → 43	9	thinkphp のスキャン→ exploit → give up
		27 → 26 → 27 → 43	1	
3-2	0%	10 → 10 → 9 → 10 → 43	4	phpmyadmin の exploit →スキャン→ give up
		10 → 10 → 10 → 9 → 10 → 43	3	
		10 → 10 → 10 → 10 → 9 → 43	1	
		10 → 10 → 10 → 10 → 43	1	
		10 → 9 → 10 → 43	1	
4-1	100%	4 → 36 → 5 → 43	2	joomla のスキャン→ exploit → give up
		4 → 5 → 5 → 36 → 36 → 43	1	
		4 → 5 → 5 → 13 → 43	1	
		4 → 36 → 5 → 5 → 43	1	
		4 → 5 → 5 → 5 → 36 → 36 → 43	1	
		5 → 4 → 36 → 5 → 43	1	
		4 → 5 → 5 → 43	1	
		4 → 5 → 5 → 5 → 36 → 5 → 43	1	
		4 → 4 → 36 → 5 → 43	1	
4-2	100 %	1 → 0 → 43	7	wordpress の exploit →スキャン→ give up
		1 → 43	2	wordpress の exploit → give up
		1 → 1 → 43	1	
4-3	100%	2 → 43	10	jenkins のスキャン→ give up
5-1	100%	34 → 43	6	sugarcrm の exploit → give up
		34 → 34 → 43	4	
5-2	100%	1 → 0 → 43	6	wordpress の exploit →スキャン→ give up
		1 → 43	3	
		1 → 1 → 0 → 43	1	
6-1	100%	2 → 0 → 6 → 9 → 4 → 5	6	各アプリケーションのスキャン→ joomla の exploit
		1 → 2 → 0 → 6 → 9 → 4 → 5	1	
		1 → 2 → 0 → 6 → 4 → 5	1	
		2 → 1 → 0 → 6 → 9 → 4 → 5	1	
		1 → 2 → 0 → 6 → 10 → 4 → 5	1	
6-2	100%	1 → 26 → 27	8	wordpress の exploit → sugarcrm のスキャン→ sugarcrm の exploit
		26 → 27	2	
7-1	0%	2 → 0 → 6 → 9 → 10 → 4 → 5 → 5 → 10 → 5	2	各アプリケーションのスキャン→ joomla、phpmyadmin の exploit を続ける
		2 → 0 → 6 → 9 → 4 → 5 → 10 → 5 → 5 → 5	2	
		2 → 0 → 6 → 9 → 10 → 4 → 5 → 10 → 5 → 5	1	
		1 → 2 → 0 → 6 → 9 → 10 → 4 → 5 → 10 → 10	1	
		2 → 0 → 6 → 9 → 4 → 5 → 10 → 10 → 10 → 10	1	
		1 → 2 → 0 → 6 → 9 → 10 → 10 → 4 → 5 → 10	1	
		2 → 0 → 6 → 9 → 10 → 4 → 5 → 5 → 5 → 10	1	
		1 → 2 → 0 → 6 → 9 → 4 → 5 → 10 → 5 → 5	1	
7-2	0%	1 → 26 → 27 → 0 → 27 → 40 → 27 → 27 → 27 → 27	2	各アプリケーションのスキャン→ sugarcrm の誤った exploit を続ける
		1 → 26 → 27 → 1 → 0 → 27 → 27 → 27 → 27 → 27	1	
		1 → 26 → 27 → 41 → 0 → 27 → 6 → 10 → 10 → 10	1	
		1 → 26 → 27 → 27 → 27 → 0 → 27 → 27 → 27 → 27	1	
		1 → 26 → 27 → 0 → 27 → 27 → 27 → 27 → 27 → 27	1	
		40 → 0 → 10 → 6 → 7 → 10 → 42 → 7 → 9 → 10	1	
		1 → 26 → 27 → 27 → 27 → 27 → 27 → 0 → 27 → 40	1	
		1 → 26 → 27 → 27 → 0 → 40 → 27 → 27 → 27 → 27	1	
		1 → 1 → 26 → 27 → 0 → 27 → 40 → 27 → 27 → 27	1	
8-1	0%	2 → 0 → 6 → 9 → 4 → 5 → 5 → 5 → 5 → 5	3	各アプリケーションのスキャン→ joomla の exploit を続ける
		2 → 0 → 6 → 9 → 4 → 5 → 10 → 5 → 5 → 5	2	
		1 → 2 → 0 → 6 → 9 → 4 → 5 → 10 → 5 → 5	1	
		1 → 2 → 0 → 9 → 6 → 4 → 5 → 10 → 5 → 5	1	
		1 → 2 → 0 → 6 → 9 → 4 → 5 → 10 → 5 → 2	1	
		2 → 0 → 6 → 9 → 10 → 4 → 5 → 5 → 5 → 5	1	
		1 → 2 → 0 → 6 → 9 → 4 → 5 → 5 → 5 → 5	1	
8-2	0%	1 → 2 → 0 → 40 → 6 → 10 → 10 → 7 → 10 → 10	2	各アプリケーションのスキャン→ phpmyadmin の exploit を続ける
		2 → 10 → 0 → 40 → 6 → 10 → 10 → 10 → 10 → 10	1	
		2 → 0 → 40 → 6 → 10 → 10 → 37 → 10 → 8 → 10	1	
		2 → 0 → 40 → 6 → 10 → 10 → 10 → 10 → 6 → 10	1	
		2 → 0 → 40 → 6 → 10 → 10 → 10 → 10 → 10 → 10	1	
		2 → 0 → 40 → 6 → 10 → 10 → 7 → 10 → 10 → 10	1	
		1 → 2 → 0 → 40 → 6 → 10 → 10 → 10 → 7 → 10	1	
		1 → 2 → 0 → 40 → 6 → 10 → 10 → 10 → 10 → 10	1	
		1 → 2 → 0 → 40 → 6 → 6 → 10 → 10 → 10 → 10	1	

表 3.12: 実験 1・手法 3-2 のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1-1	100%	43	10	即座に give up
2-1	100%	4 → 5	10	joomla のスキャン→ exploit
2-2	100%	26 → 27	10	thinkphp のスキャン→ exploit
3-1	90%	26 → 27 → 28	9	thinkphp のスキャン→ exploit
		26 → 27 → 21 → 43	1	thinkphp のスキャン→ exploit → give up
3-2	90%	9 → 11	6	phpmyadmin の exploit →スキャン→ give up
		10 → 9 → 11	3	
		9 → 43	1	
4-1	100%	4 → 5 → 5 → 43	3	joomla のスキャン→ exploit → give up
		4 → 5 → 4 → 43	2	
		4 → 5 → 5 → 5 → 43	2	
		4 → 5 → 5 → 28 → 5 → 43	1	
		4 → 5 → 4 → 5 → 43	1	
		4 → 5 → 5 → 5 → 5 → 28 → 43	1	
4-2	100%	1 → 43	9	wordpress の exploit → give up
		43	1	
4-3	100%	3 → 43	10	jenkins の exploit → give up
5-1	100%	34 → 43	10	sugarcrm の exploit → give up
5-2	100%	1 → 43	9	wordpress の exploit → give up
		43	1	
6-1	100%	6 → 4 → 5	9	drupal、joomla のスキャン→ exploit
		6 → 4 → 9 → 5	1	drupal、joomla、phpmyadmin のスキャン→ exploit
6-2	100%	26 → 27	8	thinkphp のスキャン→ exploit
		26 → 6 → 27	2	thinkphp と drupal のスキャン→ exploit
7-1	60%	6 → 4 → 5 → 7 → 9 → 11	3	各アプリケーションのスキャン、exploit → phpmyadmin の exploit
		4 → 5 → 7 → 9 → 7 → 7 → 7 → 7 → 7 → 7 → 7	1	
		6 → 4 → 5 → 9 → 11	1	
		6 → 4 → 5 → 7 → 1 → 7 → 9 → 11	1	
		6 → 4 → 5 → 7 → 7 → 7 → 7 → 7 → 9 → 11	1	
		6 → 4 → 5 → 7 → 9 → 0 → 3 → 7 → 43	1	
		6 → 9 → 4 → 5 → 2 → 1 → 7 → 41 → 41 → 41	1	
		6 → 4 → 5 → 7 → 9 → 7 → 7 → 7 → 8 → 0	1	
7-2	0%	26 → 27 → 6 → 40 → 9 → 10 → 1 → 43	1	各アプリケーションのスキャン、exploit → give up
		26 → 27 → 6 → 9 → 27 → 27 → 1 → 43	1	
		26 → 27 → 6 → 9 → 27 → 27 → 1 → 1 → 43	1	
		26 → 27 → 27 → 6 → 9 → 1 → 27 → 27 → 27 → 43	1	
		26 → 27 → 6 → 9 → 27 → 27 → 43	1	
		26 → 27 → 27 → 6 → 9 → 27 → 27 → 43	1	
		26 → 27 → 6 → 9 → 27 → 40 → 1 → 21 → 1 → 43	1	
		26 → 27 → 6 → 9 → 27 → 27 → 27 → 1 → 27 → 43	1	
		26 → 27 → 6 → 9 → 27 → 27 → 27 → 27 → 27 → 1	1	各アプリケーションのスキャン→誤った exploit を繰り返す
		26 → 27 → 27 → 6 → 9 → 27 → 27 → 1 → 27 → 27	1	
8-1	0%	6 → 4 → 5 → 7 → 9 → 7 → 1 → 1 → 1 → 7	1	各アプリケーションのスキャン、exploit → wordpress の exploit
		6 → 4 → 5 → 7 → 9 → 7 → 2 → 1 → 1 → 1	1	
		6 → 4 → 5 → 7 → 9 → 5 → 0 → 2 → 1 → 2	1	
		6 → 4 → 5 → 7 → 7 → 7 → 9 → 7 → 1 → 7	1	
		6 → 4 → 5 → 7 → 1 → 9 → 1 → 7 → 1 → 7	1	
		6 → 4 → 5 → 7 → 8 → 7 → 9 → 1 → 1 → 0	1	
		6 → 4 → 5 → 7 → 9 → 7 → 1 → 7 → 1 → 21	1	
		6 → 4 → 5 → 7 → 9 → 7 → 5 → 7 → 0 → 2	1	
		6 → 4 → 5 → 7 → 9 → 7 → 1 → 7 → 0 → 1	1	
		6 → 4 → 5 → 7 → 9 → 7 → 0 → 2 → 0 → 7	1	
8-2	90%	6 → 9 → 40 → 2 → 3 → 1 → 3 → 43	4	各アプリケーションのスキャン、exploit → wordpress の exploit → give up
		6 → 9 → 40 → 2 → 1 → 1 → 3 → 1 → 3 → 43	1	
		29 → 6 → 9 → 40 → 2 → 3 → 43	1	
		6 → 9 → 43	1	
		29 → 6 → 9 → 40 → 1 → 3 → 1 → 3 → 43	1	
		6 → 9 → 40 → 2 → 3 → 1 → 3 → 1 → 3 → 43	1	
		6 → 9 → 40 → 2 → 3 → 1 → 3 → 1 → 14 → 1	1	各アプリケーションのスキャン、exploit → wordpress の exploit

3.4.6　考察

手法 3-1、3-2 のどちらにおいても、LSTM を採用した場合、学習に大きな時間がかかることが分かった。特に手法 3-2 の場合において極めて長い時間がかかっている。また、LSTM ではスキャンだけではなく exploit も行ったうえで give up を行っていること、重複したアクションを行った回数が少ないこと、そして本来効くはずの exploit が効かないケーススタディの 5-1、5-2 において give up に成功していることから、exploit の失敗を学び give up する能力は備わっていると考えられる。目標要件 2 は達成できたといってよく、目標要件 4 の成功にも LSTM を用いた手法、特に手法 3-2 が近いと思われる。一方で対象バージョン数の少ない exploit を発見する能力では実験 2 の設定には及ばなかった。

3.5　本章の結論

本章で行った種類の手法の特徴を表 3.13 としてまとめた。

表 3.13: 実験 1 における各手法の特徴

学習手法	学習速度 (エピソード数)	single 時			five 時		
		exploit(範囲広)の学習性能	exploit(範囲狭)の学習性能	give up の学習性能	exploit(範囲広)の学習性能	exploit(範囲狭)の学習性能	give up の学習性能
1	やや早い (140 エピソード程度)	安定	不安定	やや不安定	安定	不安定	不安定
2	やや早い (120 エピソード程度)	安定	安定	不安定	安定	やや不安定	ほぼ失敗
3-1	遅い (500 エピソード程度)	安定	不安定	安定	安定	不安定	不安定
3-2	極めて遅い (700 エピソード程度)	安定	不安定	安定	安定	不安定	やや安定

学習の結果を見ると、手法 1、2、3-1、3-2 ともに、同時に観察されるアプリケーションの数が少ない状況では正しく学習が行われていると見ることができる。どの手法でも学習自体は最終的に全体として 95%の成功率を満たせることがわかった。手法ごとの特徴として、手法 1 は exploit の範囲が広いものはすべて成功できていた。また、give up も手法 2 ほど極端に失敗してはおらず、学習に必要な時間とバランスはよくなった。手法 2 では、すべての exploit に高い学習性能を示せた一方、give up が正しくできないことが極めて多くなっていた。手法 3-1、3-2 では学習が遅いものの、特に single の時は give up の成功率が極めて高いことから、exploit の経験をもとに give up をすることができているのだと考えられる。一方で、範囲が狭い exploit の学習は不安定となっていた。

総じて、どの手法にも一長一短が存在している。ただし、いずれの手法であってもアプリケーションの数が少ない場合、対象範囲の多い exploit を探索し発見することはできているので、アプリケーションのバージョンから exploit を探索するタスクは 15 種類程度であれば深層強化学習を用いて十分に処理可能なタスクであると考えられ、このようにして作成したアプリケーションをペネトレーションテストにおける簡易的なスキャンのように、効率化のための利用と考えて使用することは十分可能であると思われる。一方で、ペネトレーションテスト工程の完全な自動化を目指していくのであれば、give up の成功率の問題を避けることはできないと考えている。

Chapter 4　アプリケーションの状態遷移を考慮したペネトレーションテスト工程の自動化

4.1　タスク設定

4.1.1　前実験の問題点

前の実験ではバージョンを深層強化学習における状態として扱ったが、実際の exploit の成否は単に対象アプリケーションだけでなく、OS・DBMS・認証の有無など、様々な要因によって決定される。そのため、より実際の状況に近づけるためには、これらの要因についても考慮する必要がある。OS・DBMS に関しては docker を使った設定が難しいが、認証状態に関しては実験と組み合わせることができる。また、深層強化学習の本領ともいえる状態の遷移に関しても実現できると考えた。

4.1.2　模擬環境の設定

4.1.2.1　定義する Web アプリケーション

今回の模擬環境では、表 4.1 の八種類の Web アプリケーションに絞る。

表 4.1: 実験 2 において定義した Web アプリケーション

アプリケーション名	脆弱なバージョン数	非脆弱なバージョン数	データベースの有無	管理者によるコード実行
wordpress	130	158	使用	可能
jenkins	599	313	不使用	可能
joomla	34	106	使用	可能
drupal	188	116	使用	可能
phpmyadmin	65	165	使用	不可能
webmin	57	7	不使用	可能
sugarcrm	30	79	使用	不可能
tomcat	147	103	不使用	可能

数を減らした理由としては、今回の目的はより複雑な状態空間を定義した際、それを正しく処理できるかを調べることであるためである。また、データベースの有無は後述する SQL インジェクションにおいて重要になる。

4.1.2.2　認証

各アプリケーションには認証機能が存在するものとする。本来はユーザー名とパスワードに分かれているものであるが、本試行では簡単のため、ユーザー名を既知とし、

パスワードのみをクレデンシャルとして利用する。また、アプリケーションなどにおいて、クレデンシャルの使いまわしは重要な問題とされている。これにより、一つのアプリケーションからパスワードが流出すると、多数のアプリケーションへ攻撃が波及してしまうことがある。本試行ではこの概念を取り入れる。

認証を突破するパスワードを手に入れる方法は三種類定義されている。

- brute force
  対象のログインページに対してパスワードリストを用意し、総当たりで攻撃を行うことによって、パスワードを取得する方法である。本試行では metasploit のモジュール、独自定義した python スクリプトを用いる。

- SQL インジェクション→ hash のクラック
  SQL インジェクションは、データベースを使用しているアプリケーションに対して特殊なリクエストを行うことで、データベースコマンドを任意で実行することができる脆弱性である。これによって暗号化されたパスワードを取得することができるが、このパスワードの暗号化を解除するにはワードリストを用いた総当たりが必要になる。これをパスワードハッシュのクラックという。[39] 本試行では SQL インジェクションを python スクリプト及び sqlmap で、パスワードの暗号化の解除に john the ripper を用いる。

  基本的に john the ripper でハッシュをクラックするほうが、通信が不要で有限時間内に多くのパスワードを試行できる分、brute force を行うより難易度が低くなる。

  また、データベースが他のアプリケーションと接続しており、さらに SQL インジェクションを受けたアプリケーションがデータベース上の高権限ユーザーであった場合、そのデータベースと接続された全てのパスワードが取得されてしまう危険性がある。

- パスワードの使いまわし
  複数のアプリケーションを使用する際、それらに対して同一のパスワードを用いることは多い。[40][41] こうした環境の場合、一つのアプリケーションから漏洩したパスワードを用いてすべてのアプリケーションへの攻撃が可能となってしまう。

また、使用されるアプリケーションのうち phpmyadmin、sugarcrm を除く 6 種類のアプリケーションでは、管理者の権限を取得するとそれだけで任意のコードを実行できてしまう。これを考慮し、エージェントが取得しうる権限は「管理者の権限」「非管理者の権限」の二パターンに分けることとする。また、パスワードの強度は以下の 3 段階に定義される。

- 強度 1：brute force でパスワード取得可能、ハッシュのクラックも可能
- 強度 2：brute force でパスワード取得は不可能であるが、ハッシュのクラックは可能

- 強度 3：brute force でもハッシュのクラックでもパスワード取得不可能

4.1.3　行動空間

行動空間として定義されるのは以下となる。

- scan action

  この action が実行されると、アプリケーションのバージョン情報が取得され、観測状態 (observation) へとマッピングされる。また、アプリケーションのプラグインを取得しマッピングするものも加えられている。実環境では、wpscan、droopescan、metasploit を用い、足りないものは独自に実装を行っている。

- exploit action

  脆弱性に対する exploit に相当する action。各 exploit ごとに対象となるバージョンを持ち、対象となるアプリケーションのバージョンがそれに含まれており、かつ exploit ごとに必要な権限を取得している場合のみ成功となる。条件を満たしていない場合、失敗となる。exploit-db に脆弱性として定義されているもののほかに、wordpress、drupal、jenkins、joomla、webmin、tomcat に存在している管理者権限での任意コード実行もこれに含まれる。また、成功した場合その時点で試行は終了となるが、失敗した場合は継続する。実環境では、metasploit を用いて実装を行い、足りないものは独自に実装を行っている。

- login action

  brute force によるログインを行う。対象アプリケーションのパスワードの強度によって結果が決定される。パスワードの強度が 2 以上であるとき、ログインは不可能なものとしてパスワードの観測変数を-1 とする。実環境では、metasploit、wpscan を用いて実装を行い、モジュールやツールが存在しない場合は python を用いて独自に実装を行った。

- sql action

  sql インジェクションの脆弱性を突くアクション。脆弱性によって決められたバージョンが一致する、対象のプラグインを使用しているなどの条件を満たした際に成功する。成功した場合パスワードハッシュを取得する。また、データベースが管理者権限で動作しており、かつ他のアプリケーションとデータベースを共用していた場合、そのアプリケーションのハッシュも同時に取得できる。実環境では、python スクリプトと sqlmap を用いて実装を行った。

- give up action

  「この環境には脆弱性が存在しない」ということを決定するアクションである。実行されると即座に試行が終了となり、脆弱性が存在しない環境であった場合のみ成功、存在する環境では失敗となる。

- reuse action

  クレデンシャルの使いまわしを表現したアクションである。現在取得しているパスワードを利用して、各アプリケーションへのログインを行う。パスワードを持たない場合失敗となる。実環境への実装には、python を用いて独自に実装を行った。

- crack action

  強度 2 以下のパスワードハッシュのクラックを行う。まだクラックされていないハッシュが存在するとき成功、クラックされていないハッシュが存在するとき失敗となる。クラックに成功した場合、そのハッシュ取得元のアプリケーションの権限を得る。実際の実装には、john the ripper を用いた。

具体的な全 action は、表 4.2 のようになる。また、学習エージェントの最大行動回

表 4.2: 実験 2 における行動空間

ID	target	type	exploit 名	target version 数	必要な権限
0	wordpress	scan			0
1	wordpress	scan			0
2	wordpress	login			0
3	wordpress	exploit	WordPress 5.0.0 - Image Remote Code Execution	130	1
4	wordpress	exploit	WordPress Admin Shell Upload	all	2
5	wordpress	exploit	WordPress Simple File List Unauthenticated Remote Code Execution	all(plugin)	0
6	wordpress	sql	WordPress Loginizer log SQLi Scanner	all(plugin)	0
7	wordpress	exploit	Wordpress Drag and Drop Multi File Uploader RCE	all(plugin)	0
8	jenkins	scan			0
9	jenkins	login			0
10	jenkins	exploit	Jenkins <1.650 - Java Deserialization	599	0
11	jenkins	exploit	Jenkins-CI Script-Console Java Execution	all	2
12	joomla	scan			0
13	joomla	login			0
14	joomla	exploit	Joomla! 3.4.6 - Remote Code Execution	34	0
15	joomla	sql	joomla sql	1	0
16	joomla	exploit	joomla admin execution	all	2
17	drupal	scan			0
18	drupal	login			0
19	drupal	exploit	Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 'Drupalgeddon2' Remote Code Execution (PoC)	163	0
20	drupal	exploit	Drupal < 8.6.9 - REST Module Remote Code Execution	66	0
21	drupal	exploit	drupal admin execution	all	2
22	phpmyadmin	scan			1
23	phpmyadmin	login			0
24	phpmyadmin	exploit	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	62	1
25	phpmyadmin	exploit	phpMyAdmin 4.8.1 - Local File Inclusion to Remote Code Execution	3	1
26	phpmyadmin	sql	phpmyadmin sql	all	1
27	webmin	scan			0
28	webmin	login			0
29	webmin	exploit	Webmin 1.910 - 'Package Updates' Remote Command Execution (Metasploit)	46	1
30	webmin	exploit	webmin admin execution	all	2
31	sugarcrm	scan			0
32	sugarcrm	login			0
33	sugarcrm	exploit	SugarCRM 6.5.23 - REST PHP Object Injection (Metasploit)	30	0
34	apache tomcat	scan			0
35	apache tomcat	login			0
36	apache tomcat	exploit	Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass / Remote Code Execution (2)	61	0
37	apache tomcat	exploit	Apache Tomcat Manager Application Deployer Authenticated Code Execution	all	2
38		giveup			
39		crack			
40		reuse			

数は 10 回から 13 回に変更されている。

4.1.4　状態空間

- バージョン
  バージョンの定義のされ方は基本的に前試行に従う。

- パスワード
  管理者、非管理者ごとに 1 種類ずつ、パスワードを持つ。

- データベース接続 ID
  どのデータベースを使っているかを示す。

- データベース権限
  アプリケーションがデータベースのどの権限を使っているかを示す。通常ユーザーと、管理者の権限が存在する。

4.1.5　観測状態

学習エージェントから見える状態空間は、アプリケーションごとに以下のような構成となる。

- バージョン
  実験 1 と同様の方式でアプリケーションのバージョンをマッピングしたものとなる。

- 認証の有無
  そのアプリケーションにおける現在の権限を指す。

  - -1:アプリケーションが起動していない、もしくは強度が高いことで brute force ログインはできないことを示す。
  - 0:まだ brute force などを行っておらず、権限を持たないことを示す。
  - 1:アプリケーションに対し非管理者のユーザー権限を持つことを示す。
  - 2:アプリケーションに対し管理者権限を持つことを示す。

- サブ変数
  プラグインの存在など、アプリケーション固有の特徴を表す変数。現在のところは wordpress のプラグインを表すのみとなっている。

これらに加えて、末尾に取得されたハッシュの数、パスワードの数が追加される。

4.1.6　報酬

報酬は表 4.3 のようになる。

表 4.3: 実験 2 における action の報酬

アクションの種類	成功時報酬	失敗時報酬
scan	0	-1
exploit	1	-1
login	0	-1
sql	0	-1
give up	1	-5
reuse	0	-1
crack	0	-1

4.1.7　環境設定

学習に使う環境は、実験 1 の exploit base と同様に exploit をランダムに選択したうえで、それに対して脆弱になるように学習設定を行う。理由としては、ランダムに設定を行った場合、非脆弱な環境に偏ることが懸念されたためである。学習の決め方は表 4.4 の 11 通りとした。各環境の生成方法について、必要権限が 0 の場合は、単純に

表 4.4: 実験 2 における環境の生成方法

必要権限	属性名	パスワード取得方法	必要経路
0	privilege_0	不要	exploit
0	sub	不要 (wordpress のプラグインに対する exploit)	exploit
1	privilege_1_brute	brute force	login → exploit
1	privilege_1_password	パスワードの使いまわし	sql → crack → reuse → exploit
1	privilege_1_database	SQL データベースの繋がり	sql → crack → exploit
1	privilege_1_password and database	パスワードの使いまわし・SQL データベースの繋がり	sql → crack → reuse → exploit
2	privilege_2_brute	brute force	login → exploit
2	privilege_2_password	パスワードの使いまわし	sql → crack → reuse → exploit
2	privilege_2_database	SQL データベースの繋がり	sql → crack → exploit
2	privilege_2_password and database	パスワードの使いまわし・SQL データベースの繋がり	sql → crack → reuse → exploit
	give up	脆弱性なし	give up

脆弱なバージョン・プラグインを持ったアプリケーションが生成される。一方で権限が必要である場合、以下の 4 種類の生成方法が利用される。

1. brute
   exploit 対象となるアプリケーション (仮に A とする) の対応する権限のパスワード強度が 1 となる。login action を用いることで必要権限を取得できる。

2. password
   exploit 対象となるアプリケーション A の対応する権限のパスワード強度が 2 となり、SQL インジェクションが可能なアプリケーション B が別に生成される。B の持つパスワードのどちらか (管理者権限に対する exploit を有するアプリケーションの場合は非管理者、そうでなければランダム) が A の exploit に必要な権限と同じものとなり、SQL → crack action で取得したパスワードを reuse action によって使いまわすことで exploit に必要な権限が手に入る。

3. database
   exploit 対象となるアプリケーション A の対応する権限のパスワード強度が 2 となり、SQL インジェクションが可能なアプリケーション B が生成される。この時、A と B が同じデータベースを使用することになり、また B のデータベース権限は管理者に設定される。そのため、SQL → crack action で B だけでなく A のパスワードを取得することができ、exploit が可能となる。

4. password and database
   exploit 対象となるアプリケーション A の対応する権限のパスワード強度が 2 となり、SQL インジェクションが可能かつデータベースの管理者であるアプリケーション B と、B と同じデータベースを使用するアプリケーション C がそれぞれ別に生成される。C のどちらかの権限のパスワード (管理者権限に対する exploit を有するアプリケーションの場合は非管理者、そうでなければランダム) が A の exploit に必要な権限のパスワードと同じものとなり、SQL → crack action で取得したパスワードを reuse action で使いまわすことで exploit に必要なパスワードが手に入る。password と行うべき行動は同じであるが、違いは一緒にアプリケーション C の権限が手に入ることで、そちらに誘導されてしまう可能性が生まれることである。

以上の手法で脆弱なアプリケーションとその道筋となるアプリケーションが定義された後は、最小ですでに定義済みのアプリケーションの数、最大で 4 つとなるようランダムに非脆弱なアプリケーションを定義する。この際、「バージョン的には脆弱であるが権限を取得する方法がないため結果として非脆弱なアプリケーション」と、「バージョン的に非脆弱なアプリケーション」が等確率で生成されるものとする。

4.2　PPO アルゴリズムによるタスクの学習と評価

4.2.1　学習結果

64 個のプロセスで合計 3 回の学習試行を行い、あるプロセスにおける 1000 回の学習中の action の成功回数について平均をとった結果を図 4.1 に示す。なお、学習モデルの設定は入力と出力の次元がそれぞれ 41 次元、48 次元となっている他は実験 1 と同じである。

4.2.2　モデルの評価

最終的に成功率が 96 ± 1.1%に達した学習済みモデルを 3 つ用いて、各 exploit と、その生成のアルゴリズムに対して評価を行う。評価には 10000 回の試行を 3 回、合計 30000 回の試行を行ったデータを用い、指標は実験 1 と同様、exploit に成功したエピソード数の割合とした。結果を表 4.5 に示す。

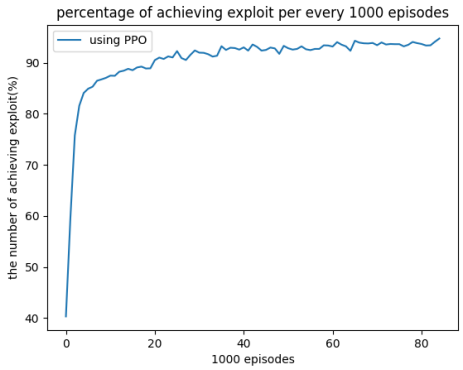


図 4.1: PPO アルゴリズムによる学習の過程

4.2.3　ケーススタディ

学習したモデルを用い、実際のアプリケーションおよび exploit を利用してケーススタディを行った。ケーススタディにおいては用意した環境に対して 13 回の行動を行い、行動パターンを分析した。用意した環境は表 4.6 のようになる。今回の実験では、パスワードを以下のように用意した。

- パスワード A:強度 1 のパスワード。
- パスワード B:強度 2 のパスワード。
- パスワード C:強度 3 のパスワード。

4.2.3.1　実装

実装においては、脆弱なアプリケーションには実験 1 と同様、vulfocus および vulhub が提供しているコンテナを用いた。また、exploit には pymetasploit3 を介して metasploit を、crack action には john the ripper を subprocess モジュールを介してコマンドラインから呼び出すことで用い、ツールのみを用いることができなかった exploit や reuse action は python によって独自に実装を行った。

4.2.3.2　結果

結果は表 4.7 のようになった。

(5-1 は設定ミスにより、PPO の場合だけ C となってしまった。実験には影響を及ぼさないと考えられる。)

表 4.5: 実験 2・PPO アルゴリズムでのモデルの評価

action id	action name	vulnerable versions	way of getting password	success	failure	give up
3	WordPress 5.0.0 - Image Remote Code Execution	130	privilege_1_brute	91.6%	8.2%	0.2%
			privilege_1_database	97.5%	2.5%	0.0%
			privilege_1_password	95.8%	4.1%	0.1%
			privilege_1_password and database	98.1%	1.8%	0.1%
4	WordPress Admin Shell Upload	all	privilege_2_brute	95.0%	5.0%	0.0%
			privilege_2_database	98.2%	1.8%	0.0%
			privilege_2_password	94.6%	5.4%	0.0%
			privilege_2_password and database	98.6%	1.4%	0.0%
5	WordPress Simple File List Unauthenticated Remote Code Execution	all	sub	94.6%	4.8%	0.7%
7	Wordpress Drag and Drop Multi File Uploader RCE	all	sub	94.7%	4.6%	0.8%
10	Jenkins < 1.650 - Java Deserialization	599	privilege_0	90.6%	8.8%	0.7%
11	Jenkins-CI Script-Console Java Execution	all	privilege_2_brute	97.3%	2.7%	0.0%
			privilege_2_password	92.2%	7.6%	0.2%
			privilege_2_password and database	92.3%	7.7%	0.1%
14	Joomla! 3.4.6 - Remote Code Execution	34	privilege_0	97.0%	2.9%	0.1%
16	joomla_admin_execution	all	privilege_2_brute	93.8%	6.1%	0.1%
			privilege_2_database	92.3%	7.6%	0.1%
			privilege_2_password	91.2%	8.4%	0.3%
			privilege_2_password and database	96.9%	3.0%	0.1%
19	Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 ' Drupalgeddon2 ' Remote Code Execution (PoC)	122	privilege_0	94.3%	5.1%	0.6%
20	Drupal < 8.6.9 - REST Module Remote Code Execution	32	privilege_0	91.5%	7.6%	0.9%
21	drupal_admin_execution	all	privilege_2_brute	94.4%	5.5%	0.1%
			privilege_2_database	91.1%	8.9%	0.1%
			privilege_2_password	93.8%	5.9%	0.3%
			privilege_2_password and database	95.7%	4.1%	0.2%
24	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	62	privilege_1_brute	94.2%	4.3%	1.5%
			privilege_1_database	91.0%	8.5%	0.5%
			privilege_1_password	93.1%	6.0%	0.9%
			privilege_1_password and database	94.5%	5.0%	0.4%
25	phpMyAdmin 4.8.1 - Local File Inclusion to Remote Code Execution	3	privilege_1_brute	98.8%	1.2%	0.0%
			privilege_1_database	93.5%	6.2%	0.3%
			privilege_1_password	94.2%	5.1%	0.7%
			privilege_1_password and database	94.9%	4.6%	0.5%
29	Webmin 1.910 - 'Package Updates' Remote Command Execution	46	privilege_1_brute	93.0%	6.5%	0.6%
			privilege_1_password	94.0%	5.0%	0.9%
			privilege_1_password and database	93.8%	5.9%	0.3%
30	webmin_admin_execution	all	privilege_2_brute	97.5%	2.5%	0.0%
			privilege_2_password	94.4%	4.7%	0.9%
			privilege_2_password and database	95.2%	4.5%	0.3%
33	SugarCRM 6.5.23 - REST PHP Object Injection (Metasploit)	30	privilege_0	92.6%	6.5%	0.9%
36	Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass / Remote Code Execution (2)	61	privilege_0	89.9%	9.4%	0.8%
37	Apache Tomcat Manager Application Deployer Authenticated Code Execution	all	privilege_2_brute	96.7%	3.3%	0.0%
			privilege_2_password	93.2%	6.8%	0.0%
			privilege_2_password and database	93.4%	6.6%	0.0%
38	action_giveup	all	giveup	0.0%	36.3%	63.7%

表 4.6: 実験 2・ケーススタディ環境

環境 環境	環境の特徴	環境を構成する アプリケーション	各アプリケーション 対応する exploit	管理者の パスワード	非管理者の パスワード	データベース 番号	データベース 権限
1-1	認証の必要のない exploit の発見が必要	joomla 3.4.3	14	C	C	1	非管理者
		drupal 8.9.19	なし	C	C	2	非管理者
		wordpress 5.5.7	なし	C	C	3	非管理者
		jenkins 2.137	なし	C	C	なし	なし
1-2		sugarcrm 6.5.23	33	C	C	1	非管理者
		drupal 8.9.19	なし	C	C	2	非管理者
		tomcat 9.0.1	なし	C	C	なし	なし
		jenkins 2.137	なし	C	C	なし	なし
2-1	管理者権限の 必要な exploit の発見が必要	wordpress 5.5.7	4	C	A	1	非管理者
		drupal 8.9.19	なし	C	C	2	非管理者
		tomcat 9.0.1	なし	C	C	なし	なし
		jenkins 2.137	なし	C	C	なし	なし
2-2		webmin 1.984	30	C	A	なし	なし
		drupal 8.9.19	なし	C	C	1	非管理者
		tomcat 9.0.1	なし	C	C	なし	なし
		jenkins 2.137	なし	C	C	なし	なし
3-1	ログインが不可能であり、 exploit も不可能	wordpress 5.5.7	4	A	C	1	非管理者
		drupal 8.9.19	なし	C	C	2	非管理者
		tomcat 9.0.1	なし	C	C	なし	なし
		jenkins 2.137	なし	C	C	なし	なし
4-1	データベースを用いたパスワード取得と 使いまわしが必要な exploit の発見が必要	webmin 1.890	29	B	C	なし	なし
		drupal 7.81	なし	C	C	1	非管理者
		joomla 4.0.4	なし	C	C	2	非管理者
		phpmyadmin 5.5.1	なし	B	A	1	管理者
4-2	データベースとパスワードの使いまわし が必要な複雑な環境	webmin 1.890	29	B	C	なし	なし
		wordpress 5.5.7	なし	B	C	1	非管理者
		sugarcrm 6.5.24	なし	C	C	2	非管理者
		phpmyadmin 5.5.1	なし	C	A	1	管理者
5-1	複雑な認証の流れがあるが、 exploit が成立しない	tomcat 9.0.1	なし	C	C	なし	なし
		drupal 8.9.19	なし	B	C	1	非管理者
		joomla 4.0.4	なし	C	C	2	非管理者
		phpmyadmin 5.5.1	なし	B(C)	A	1	管理者
6-1	複雑な認証の流れがあるが、 正しいのは権限 0 で行える exploit	tomcat 9.0.1	なし	C	C	なし	なし
		drupal 8.9.19	なし	B	C	1	非管理者
		joomla 3.4.3	14	C	C	2	非管理者
		phpmyadmin 5.5.1	なし	C	A	1	管理者
6-2		webmin 1.890	なし	C	C	なし	なし
		drupal 7.55	19	B	C	1	非管理者
		wordpress 5.5.7	なし	C	C	2	非管理者
		phpmyadmin 5.5.1	なし	C	A	1	管理者

表 4.7: 実験 2・PPO のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1-1	100%	12 → 14	9	joomla のスキャン→ exploit
		0 → 12 → 14	1	
1-2	100%	17 → 35 → 9 → 18 → 33	2	各アプリケーションのスキャン、ログイン試行→ sugarcrm の exploit
		17 → 9 → 35 → 18 → 34 → 36 → 36 → 36 → 0 → 33	1	
		17 → 35 → 9 → 0 → 18 → 8 → 10 → 33	1	
		17 → 35 → 9 → 18 → 34 → 33	1	
		17 → 35 → 9 → 18 → 31 → 33	1	
		17 → 9 → 18 → 35 → 8 → 34 → 0 → 33	1	
		17 → 35 → 9 → 10 → 0 → 33	1	
		9 → 17 → 35 → 18 → 34 → 0 → 33	1	
		17 → 18 → 35 → 9 → 8 → 31 → 0 → 33	1	
2-1	100%	2 → 4	1	各アプリケーションのスキャン、ログイン試行→ wordpress のログイン→ exploit
		17 → 35 → 1 → 18 → 0 → 9 → 34 → 0 → 2 → 4	1	
		17 → 35 → 9 → 18 → 0 → 36 → 1 → 2 → 0 → 4	1	
		17 → 35 → 9 → 0 → 2 → 4	1	
		17 → 18 → 35 → 0 → 0 → 9 → 10 → 1 → 2 → 4	1	
		0 → 17 → 2 → 4	1	
		17 → 0 → 35 → 2 → 4	1	
		17 → 18 → 35 → 9 → 0 → 2 → 0 → 4	1	
		17 → 35 → 18 → 9 → 0 → 34 → 36 → 1 → 0 → 1 → 2 → 4	1	
		0 → 17 → 35 → 9 → 2 → 4	1	
2-2	100%	17 → 35 → 9 → 18 → 28 → 30	3	各アプリケーションのスキャン、ログイン試行→ webmin のログイン→ exploit
		17 → 9 → 18 → 28 → 30	2	
		17 → 35 → 28 → 30	1	
		17 → 28 → 30	1	
		17 → 9 → 35 → 18 → 28 → 30	1	
		17 → 0 → 28 → 30	1	
		17 → 9 → 0 → 35 → 18 → 28 → 30	1	
3-1	0%	17 → 35 → 9 → 18 → 0 → 34 → 2 → 0 → 10 → 1 → 36 → 10 → 1	1	各アプリケーションのスキャン、ログイン試行を繰り返す
		17 → 35 → 2 → 18 → 0 → 9 → 10 → 0 → 34 → 1 → 1 → 36 → 10	1	
		17 → 2 → 18 → 9 → 1 → 35 → 0 → 0 → 34 → 1 → 1 → 8 → 8	1	
		17 → 35 → 18 → 9 → 0 → 1 → 2 → 0 → 34 → 1 → 10 → 8 → 1	1	
		17 → 8 → 9 → 18 → 35 → 0 → 1 → 0 → 1 → 2 → 1 → 1 → 34	1	
		17 → 1 → 18 → 35 → 0 → 0 → 9 → 34 → 1 → 1 → 2 → 36 → 10	1	
		1 → 17 → 35 → 9 → 0 → 0 → 1 → 18 → 36 → 2 → 1 → 1 → 10	1	
		35 → 17 → 18 → 9 → 0 → 10 → 1 → 1 → 0 → 2 → 1 → 34 → 1	1	
		17 → 1 → 2 → 35 → 9 → 0 → 18 → 0 → 1 → 34 → 36 → 36 → 10	1	
		2 → 17 → 35 → 9 → 1 → 0 → 1 → 1 → 18 → 36 → 34 → 1 → 1	1	
4-1	100%	23 → 26 → 39 → 40 → 29	8	phpmyadmin にログイン→ SQL を実行→ crack → reuse → webmin の exploit
		23 → 26 → 39 → 40 → 0 → 29	1	
		23 → 26 → 39 → 0 → 40 → 29	1	
4-2	0%	23 → 26 → 39 → 3 → 3 → 3 → 3 → 0 → 3 → 3 → 3 → 3 → 3	3	phpmyadmin にログイン→ SQL を実行→ crack → wordpress の exploit を繰り返す
		23 → 26 → 39 → 3 → 0 → 3 → 3 → 3 → 3 → 3 → 3 → 3 → 3	2	
		23 → 0 → 26 → 39 → 3 → 3 → 3 → 3 → 3 → 3 → 3 → 0 → 3	2	
		23 → 26 → 39 → 3 → 3 → 3 → 0 → 3 → 3 → 3 → 3 → 3 → 3	1	
		23 → 26 → 39 → 3 → 3 → 0 → 3 → 3 → 3 → 3 → 3 → 3 → 3	1	
		23 → 26 → 39 → 3 → 3 → 3 → 3 → 3 → 0 → 3 → 3 → 3 → 3	1	
5-1	0%	23 → 26 → 39 → 40 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40	4	phpmyadmin にログイン→ SQL を実行→ crack → reuse(以降繰り返し)
		23 → 26 → 39 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 40	2	
		23 → 26 → 39 → 40 → 40 → 40 → 40 → 40 → 0 → 17 → 40 → 40 → 40	1	
		23 → 26 → 39 → 0 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 40	1	
		23 → 26 → 39 → 40 → 0 → 40 → 40 → 24 → 40 → 40 → 40 → 40 → 40	1	
		23 → 26 → 39 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40 → 40	1	
6-1	0%	23 → 26 → 39 → 40 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40	3	phpmyadmin にログイン→ SQL を実行→ crack → reuse(以降繰り返し)
		23 → 26 → 39 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40 → 40	2	
		23 → 26 → 39 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40	1	
		23 → 26 → 39 → 40 → 40 → 40 → 40 → 7 → 40 → 40 → 40 → 40 → 40	1	
		23 → 0 → 26 → 39 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 0 → 40	1	
		23 → 26 → 39 → 40 → 40 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40	1	
		23 → 22 → 26 → 39 → 40 → 21 → 0 → 40 → 40 → 21 → 40 → 40 → 21	1	
6-2	0%	23 → 26 → 39 → 0 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 40	3	phpmyadmin にログイン→ SQL を実行→ crack → reuse(以降繰り返し)
		23 → 0 → 26 → 39 → 40 → 40 → 40 → 40 → 40 → 40 → 40 → 0 → 40	2	
		23 → 26 → 39 → 40 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40	2	
		23 → 26 → 39 → 40 → 40 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40	1	
		23 → 26 → 39 → 40 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40	1	
		23 → 3 → 26 → 39 → 40 → 40 → 0 → 40 → 40 → 40 → 40 → 40 → 40	1	

4.2.4　考察

学習においては、100 エピソードほどとさほど時間をかけることなく 95%に達した。実験 1 と比べ、状態空間の数が減り、また可能な行動回数が増えているためであると推測することができる。モデルの評価においては、exploit の成功率はどの条件でも 80%を超えていたことから、状態の遷移を学習することはできていると推測される。一方で、give up の成功率が 63.7%と低くなっていた。

ケーススタディの結果を見ると、データベースを用いない 1-1 から 2-2 までは必ず成功している。また、4-1 では成功しているものの 4-2 では wordpress の exploit に固執して同一の行動を続けてしまっている。5-1、6-1、6-2 についても同様に、ほぼ同じ行

動を時間切れまで続けることになっている。また、4-1 を注視すると、ログイン試行は行っているものの途中でアプリケーションのスキャンを行っていない。
このことから、各戦略に特化して学習してしまった結果、複数の可能性のある環境で他の行動がとれなくなってしまっていると推測される。

4.3　PPO-LSTM アルゴリズムによるタスクの学習と評価

4.3.1　実験設定

PPO アルゴリズムを用いた学習では、ケーススタディから見えた問題点として、一つの戦略に固執してしまい繰り返し同じ行動を行ってしまうという問題点があった。実験 1 の手法 2 と同様、PPO-LSTM を用いた学習を行うことで、時系列データの処理によるより柔軟な学習を目指す。

4.3.2　学習結果

64 個のプロセスで合計 3 回の学習試行を行い、あるプロセスにおける 1000 回の学習中の action の成功回数について平均をとった結果を図 4.2 に示す。学習モデルは入力と出力の次元がそれぞれ 41 次元、48 次元となっている点を除けば実験 1-3 と同じである。

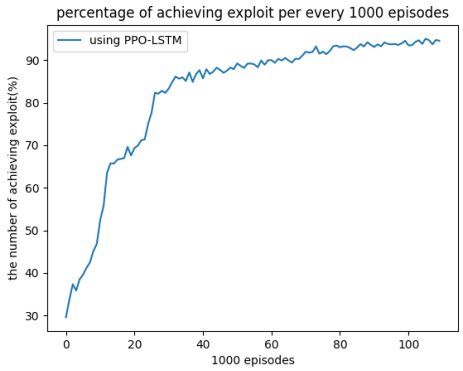


図 4.2: PPO-LSTM アルゴリズムによる学習の過程

4.3.3　モデルの評価

PPO アルゴリズムと同様の手法でモデルを評価した結果を表 4.8 に示す。

表 4.8: 実験 2 ・PPO-LSTM でのモデルの評価

action id	action name	vulnerable versions	way of getting password	success	failure	give up
3	WordPress 5.0.0 - Image Remote Code Execution	130	privilege_1_brute	92.9%	5.1%	2.1%
			privilege_1_database	96.0%	3.8%	0.2%
			privilege_1_password	95.9%	3.5%	0.6%
			privilege_1_password and database	97.9%	1.9%	0.2%
4	WordPress Admin Shell Upload	all	privilege_2_brute	99.2%	0.7%	0.2%
			privilege_2_database	97.1%	2.9%	0.0%
			privilege_2_password	96.2%	3.6%	0.2%
			privilege_2_password and database	97.6%	2.2%	0.2%
5	WordPress Simple File List Unauthenticated Remote Code Execution	all	sub	98.9%	0.6%	0.4%
7	Wordpress Drag and Drop Multi File Uploader RCE	all	sub	99.0%	0.6%	0.4%
10	Jenkins <1.650 - Java Deserialization	599	privilege_0	93.8%	4.1%	2.1%
11	Jenkins-CI Script-Console Java Execution	all	privilege_2_brute	97.9%	1.7%	0.5%
			privilege_2_password	95.2%	4.4%	0.4%
			privilege_2_password and database	97.8%	1.9%	0.3%
14	joomla_exploit	34	privilege_0	98.3%	1.4%	0.3%
16	joomla_admin_execution	all	privilege_2_brute	96.4%	3.0%	0.6%
			privilege_2_database	98.6%	1.1%	0.3%
			privilege_2_password	95.7%	3.8%	0.5%
			privilege_2_password and database	97.8%	2.1%	0.2%
19	Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 ' Drupalgeddon2 ' Remote Code Execution (PoC)	122	privilege_0	93.0%	5.2%	1.9%
20	Drupal < 8.6.9 - REST Module Remote Code Execution	32	privilege_0	93.3%	5.2%	1.5%
21	drupal_admin_execution	all	privilege_2_brute	95.7%	4.0%	0.3%
			privilege_2_database	96.9%	3.1%	0.0%
			privilege_2_password	95.8%	3.7%	0.5%
			privilege_2_password and database	97.8%	2.0%	0.2%
24	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	62	privilege_1_brute	95.0%	4.4%	0.6%
			privilege_1_database	96.6%	2.8%	0.6%
			privilege_1_password	95.0%	4.1%	0.9%
			privilege_1_password and database	95.2%	4.5%	0.3%
25	phpMyAdmin 4.8.1 - Local File Inclusion to Remote Code Execution	3	privilege_1_brute	95.6%	3.2%	1.2%
			privilege_1_database	96.0%	3.4%	0.6%
			privilege_1_password	95.0%	4.1%	0.9%
			privilege_1_password and database	93.3%	6.3%	0.4%
29	Webmin 1.910 - 'Package Updates' Remote Command Execution	46	privilege_1_brute	95.9%	3.2%	0.8%
			privilege_1_password	94.4%	4.5%	1.1%
			privilege_1_password and database	95.5%	4.2%	0.3%
30	webmin_admin_execution	all	privilege_2_brute	98.5%	0.9%	0.6%
			privilege_2_password	95.4%	4.0%	0.7%
			privilege_2_password and database	97.6%	2.2%	0.2%
33	SugarCRM 6.5.23 - REST PHP Object Injection (Metasploit)	30	privilege_0	94.4%	3.7%	1.9%
36	Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass / Remote Code Execution (2)	61	privilege_0	94.2%	4.7%	1.1%
37	Apache Tomcat Manager Application Deployer Authenticated Code Execution	all	privilege_2_brute	97.9%	1.6%	0.5%
			privilege_2_password	94.8%	4.5%	0.7%
			privilege_2_password and database	96.8%	3.1%	0.1%
38	action_giveup	all	giveup	0.0%	18.2%	81.8%

4.3.4　ケーススタディ

PPO-LSTM を用いて PPO と同様にケーススタディを行った結果を表 4.9 に示す。

4.3.5　考察

学習速度は PPO アルゴリズムを用いた時とエピソード数では大きな変化はなく、実験 1 に LSTM を適用した場合に比べ素早く学習を行うことができた。

モデル評価においては、PPO アルゴリズムと比べ give up の成功率が大きく伸びている。実験 1 と同様、実際に exploit を行うことにより判断ができていると推測される。

ケーススタディにおいては、PPO と比べて 6-1、6-2 での成功率が大幅に上がっている。これは、学習の過程で単一ではない経路を学習し、切り替えることができているからだと考えられる。一方で、複雑な認証の流れを要する 4-1 での成功率が PPO より下がっている。これは、一つの経路だけではない様々な action を行っているため、必ずしも reuse を選べるわけではないことが要因と考えられる。また、4-2 では PPO ほどではないが、wordpress の exploit に引き寄せられてしまっている。give up が必要な 3-1、5-1 においてもやはり成功はできていない。このことから、多数のアプリケーショ

表 4.9: 実験 2・PPO-LSTM のケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1-1	100%	0→13→9→2→18→1→12→14	8	各アプリケーションのスキャン、ログイン→joomla のスキャン→exploit
		0→13→9→2→18→1→15→15→1→1→1→12→14	1	
		0→13→9→2→18→1→12→10→14	1	
1-2	100%	18→32→19→9→10→35→36→33	5	各アプリケーションのスキャン、ログイン→sugarcrm の exploit
		18→32→19→9→35→33	1	
		18→32→17→9→10→35→36→8→33	1	
		18→32→19→9→10→35→36→33→36→33	1	
		18→32→19→9→10→35→17→20→17→31→33	1	
		18→32→19→9→10→33	1	
2-1	100%	0→9→35→2→4	9	各アプリケーションのスキャン、ログイン→wordpress のログイン→exploit
		0→9→35→2→31→4	1	
2-2	100%	18→19→28→30	6	各アプリケーションのスキャン、ログイン→webmin のログイン→exploit
		18→28→30	1	
		18→19→28→17→27→27→27→30	1	
		28→19→9→19→35→30	1	
		18→9→28→30	1	
3-1	0%	0→9→35→2→18→31→1→1→1→17→34→20→8	1	各アプリケーションのスキャン、ログイン試行を繰り返す
		0→9→35→2→18→1→19→36→20→17→36→36→36	1	
		0→9→35→2→18→1→19→36→10→17→31→1→1	1	
		0→9→35→2→1→31→17→1→18→1→1→10→20	1	
		0→9→35→2→18→1→19→36→36→17→10→10→1	1	
		0→9→35→2→18→1→19→36→10→34→19→10→19	1	
		0→9→35→2→18→1→19→36→10→20→10→8→19	1	
		0→9→35→2→18→1→19→36→17→1→1→8→36	1	
		0→9→35→2→18→1→19→36→31→1→1→1→1	1	
		0→9→35→2→18→1→19→36→36→1→36→1→17	1	
4-1	90%	18→28→12→14→13→23→26→39→40→29	4	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション→クラック→使いまわし
		18→28→12→14→14→13→23→26→39→40→29	2	
		18→28→12→14→13→23→22→26→20→39→40→29	1	
		18→28→12→14→14→14→13→23→22→26→39→40→29	1	
		18→28→12→14→14→13→23→22→26→39→40→29	1	
		18→28→12→14→18→14→18→13→23→22→26→39→40	1	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション→使いまわし
4-2	40%	0→28→32→2→1→23→26→39→3→22→33→33→33	2	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン →SQL インジェクション→クラック→sugarcrm・wordpress の exploit
		0→28→32→2→1→23→26→39→39→40→29	2	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン
		0→28→32→2→1→31→23→22→26→39→3→40→29	1	→SQL インジェクション→クラック→使いまわし→exploit
		0→28→31→32→2→1→23→31→22→26→39→40→29	1	
		0→28→32→2→1→23→26→39→3→33→3→3→22	1	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン
		0→28→32→2→1→23→26→22→39→3→3→33→33	1	→SQL インジェクション→クラック→sugarcrm・wordpress の exploit
		0→28→32→2→1→23→26→39→3→3→33→33→22	1	
		0→28→32→2→1→23→26→39→3→3→33→22→3	1	
5-1	0%	18→12→14→13→23→22→22→20→8→19→19→17→22	1	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション
		18→12→14→23→26→39→17→22→26→20→20→34→26	1	→クラック→他アプリケーションへのスキャン、ログイン、exploit
		18→12→14→13→23→22→26→39→40→20→21→21→21	1	
		18→12→14→13→23→22→26→39→39→20→20→34→20	1	
		18→12→14→14→14→13→23→26→39→40→21→21→22	1	
		18→12→14→13→23→26→39→17→19→21→20→22→40	1	
		18→12→14→15→23→26→39→40→22→21→21→21→21	1	
		12→18→13→23→22→26→39→40→21→21→21→22→21	1	
		18→12→14→13→23→26→39→40→21→21→22→26→21	1	
		18→12→14→13→23→26→39→36→5→20→22→26→26	1	
6-1	60%	18→12→14	3	drupal のログイン→joomla のスキャン→exploit
		18→12→18→14	2	
		18→12→13→11→14	1	
		18→12→13→23→26→39→36→20→17→22→20→22→20	1	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション
		18→12→19→13→23→26→39→40→22→22→13→35→8	1	→クラック→他アプリケーションへのスキャン、ログイン、exploit
		18→12→13→23→26→39→20→17→20→20→22→26→20	1	
		18→12→13→23→26→22→39→40→20→19→20→22→20	1	
6-2	80%	0→28→2→18→23→26→39→40→19	4	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション
		0→28→2→18→23→31→22→26→39→40→19	1	→使いまわし→exploit
		0→28→2→18→23→26→39→40→20→1→17→26→19	1	各アプリケーションのスキャン、ログイン→phpmyadmin のログイン→SQL インジェクション→exploit
		0→28→2→18→23→26→39→40→20→17→19	1	各アプリケーションのスキャン、ログイン→exploit
		0→28→2→18→2→23→26→39→40→14→20→19	1	
		0→28→2→18→23→26→39→40→20→34→20→20→34	1	各アプリケーションのスキャン、ログイン→他アプリケーションの exploit
		0→28→2→18→23→26→39→39→40→20→20→22→22	1	

ンが存在する環境での give up はいまだ難しいタスクであるといえる。また、PPO 使用時ほどではないが同じ action を 3-4 回ほど繰り返している状況が存在するため、やはりまだ無駄な行動が存在している。

4.4　一般性を考慮した観測状態の作成

4.4.1　前項までの問題点

前項までに提案した手法は、Web アプリケーションの数だけ観測状態を積み上げていたため、無駄となる状態空間が多くなってしまっていた。また、この形式ではアプリケーションを新たに追加する際に観測状態の長さが変化してしまうため、前のモデルを利用することができず拡張性に乏しい。

これを踏まえ、最大のアプリケーション数が 4 とした時のある程度一般的な観測状態の表し方を作成した。

4.4.2　観測状態

観測状態は以下のように構成される。

- アプリケーション ID
  アプリケーションは二進数で表す。8 個の各アプリケーションに対し、0-7 の数字を順番に割り振り、3 桁の 2 進数へと直した。
- バージョン
- 認証の有無
  これらは上記二つの実験と同じ表し方とする。
- サブ変数
  ３桁の数列を用いる。現状では wordpress のプラグインを表すための関数であることに変わりはないが、一般性を考慮して各アプリケーションに無意味な値として設定する。

この末尾にハッシュの数とパスワードを加えたものが観測状態となる。合計長は 54 次元となった。

4.4.3　学習アルゴリズム

本実験では、時間不足のため PPO のみを用いた学習を行うにとどまった。

4.4.4　学習結果

PPO および PPO-LSTM と同様、64 個のプロセスで合計 3 回の学習試行を行い、あるプロセスにおける 1000 回の学習中の action の成功回数について平均をとり、通常の状態空間の表し方で PPO を用いた手法と比較した結果を図 4.3 に示す。

4.4.5　モデルの評価

PPO、PPO-LSTM と同様の手法で PPO-LSTM を用いて PPO と同様にモデルの評価を行った結果を表 4.10 に示す。

4.4.6　ケーススタディ

PPO、PPO-LSTM と同様の環境でケーススタディを行った。表 4.11 にその結果を示す。

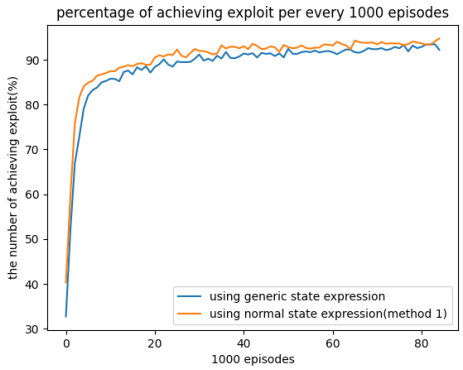


図 4.3: PPO アルゴリズムでの汎用的モデルの学習の過程

4.4.7　考察

学習は通常の PPO と比べてわずかに遅いものの、さほど過程に変化はなく、問題なく行えていることが読み取れた。

ケーススタディにおいてはおおむね通常の PPO と同じ結果であったが、joomla の exploit ではなく SQL インジェクションを繰り返してしまう部分だけは大きく異なった。モデル評価を確認する限り、この exploit を学習できていないわけではないと思われるが、状況に流されてしまい別の戦略に向けて動いてしまったものと考えられる。一方で、4-2 など wordpress に引き寄せられず正しい選択肢を選べる場合もあった。

とはいえ、PPO でも別の方策に流れることはあったように、現状ではこの手法固有の問題とは言い難い。今回は 8 種類のみであるが、より多くのアプリケーションに対する exploit を実装したときこそ、この手法の真価が試されると考えられる。

4.5　本章の結論

状態空間の遷移の学習においては、PPO、PPO-LSTM 双方ともに問題なく行えるものと分かった。

学習モデルの評価においても、exploit に関してはいずれも 85%を超えており、効率化スクリプトとしては実用に十分な学習ができていると考えられる。

一方で、PPO では SQL インジェクション→クラック→ reuse といった流れを強調した結果、一つの戦略へと傾倒してしまうため、ケーススタディで示されたように異なる戦略が必要であったり、途中で他の要素が挟まるとリカバリーが利かず、同じ action を繰り返し行ってしまうということが多くなってしまったが、PPO-LSTM ではこの点はやや緩和され、スキャン、login、exploit など多様な方向性を見据えて行動することが可能となっていた。

ただし、どちらのケーススタディにおいても、バージョンのスキャンを行わずにそのまま exploit を行うことがやや多くなっていた。認証の機構が追加された分、バージョンが軽視されやすくなってしまったと推測される。

表 4.10: 実験 2・PPO アルゴリズムでの汎用的モデルの評価

action id	action name	vulnerable versions	way of getting password	success	failure	give up
3	WordPress 5.0.0 - Image Remote Code Execution	130	privilege_1_brute	97.1%	2.8%	0.1%
			privilege_1_database	96.5%	3.5%	0.0%
			privilege_1_password	98.3%	1.7%	0.0%
			privilege_1_password and database	98.8%	1.2%	0.0%
4	WordPress Admin Shell Upload	all	privilege_2_brute	97.4%	2.6%	0.0%
			privilege_2_database	97.9%	2.1%	0.0%
			privilege_2_password	97.1%	2.9%	0.0%
			privilege_2_password and database	99.0%	1.0%	0.0%
5	WordPress Simple File List Unauthenticated Remote Code Execution	all	sub	99.4%	0.3%	0.2%
7	Wordpress Drag and Drop Multi File Uploader RCE	all	sub	99.3%	0.5%	0.2%
10	Jenkins <1.650 - Java Deserialization	599	privilege_0	99.3%	0.6%	0.1%
11	Jenkins-CI Script-Console Java Execution	all	privilege_2_brute	98.3%	1.7%	0.0%
			privilege_2_password	95.3%	4.0%	0.7%
			privilege_2_password and database	96.8%	2.4%	0.9%
14	Joomla! 3.4.6 - Remote Code Execution	34	privilege_0	96.5%	3.3%	0.1%
16	joomla_admin_execution	all	privilege_2_brute	91.9%	7.8%	0.3%
			privilege_2_database	91.3%	8.7%	0.0%
			privilege_2_password	93.2%	6.8%	0.1%
			privilege_2_password and database	96.5%	3.5%	0.0%
19	Drupal < 8.3.9 / < 8.4.6 / < 8.5.1 ' Drupalgeddon2 ' Remote Code Execution (PoC)	122	privilege_0	95.5%	3.8%	0.7%
20	Drupal < 8.6.9 - REST Module Remote Code Execution	32	privilege_0	93.3%	5.3%	1.4%
21	drupal_admin_execution	all	privilege_2_brute	92.6%	7.1%	0.3%
			privilege_2_database	93.2%	6.6%	0.2%
			privilege_2_password	95.1%	4.6%	0.2%
			privilege_2_password and database	96.9%	3.1%	0.0%
24	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	62	privilege_1_brute	94.7%	4.8%	0.4%
			privilege_1_database	85.8%	14.2%	0.0%
			privilege_1_password	96.3%	3.5%	0.2%
			privilege_1_password and database	95.6%	4.4%	0.1%
25	phpMyAdmin 4.6.2 - (Authenticated) Remote Code Execution	3	privilege_1_brute	97.8%	1.8%	0.3%
			privilege_1_database	89.9%	10.0%	0.1%
			privilege_1_password	95.6%	3.9%	0.5%
			privilege_1_password and database	93.2%	6.8%	0.0%
29	Webmin 1.910 - 'Package Updates' Remote Command Execution	46	privilege_1_brute	93.8%	6.1%	0.1%
			privilege_1_password	90.6%	9.1%	0.3%
			privilege_1_password and database	96.0%	3.8%	0.2%
30	webmin_admin_execution	all	privilege_2_brute	97.9%	2.1%	0.1%
			privilege_2_password	90.2%	9.6%	0.2%
			privilege_2_password and database	94.8%	5.2%	0.0%
33	SugarCRM 6.5.23 - REST PHP Object Injection (Metasploit)	30	privilege_0	93.2%	5.8%	1.1%
36	Apache Tomcat < 9.0.1 (Beta) / < 8.5.23 / < 8.0.47 / < 7.0.8 - JSP Upload Bypass / Remote Code Execution (2)	61	privilege_0	93.6%	6.1%	0.3%
37	Apache Tomcat Manager Application Deployer Authenticated Code Execution	all	privilege_2_brute	94.2%	5.8%	0.0%
			privilege_2_password	87.2%	12.8%	0.0%
			privilege_2_password and database	94.8%	5.2%	0.0%
38	action_giveup	all	giveup	0.0%	36.2%	63.8%

環境の生成方法は存在しうる攻撃への道筋を明確化させたものであり、攻撃の戦略を学ぶためには必要なものであるが、やや一般性を失ってしまったと考えられる。
実験 1 と同様、現状であっても利用した 8 種類のアプリケーションに対する攻撃の効率化ツールとしては十分な性能を有していると考えられるが、行程の自動化をより促進するのであればアプリケーション生成のアルゴリズムを考え直す、観測状態をより現実に即したものに変更するなどの工夫が必要になると思われる。

最後に汎用的な状態空間の表し方を定義したが、PPO とモデルのパフォーマンス・学習に必要な時間に大きな差はなかったことから、十分利用に耐えるものと考えられる。

表 4.11: 実験 2・一般的な状態空間でのケーススタディの結果

環境	成功率	観測された行動	行動の回数	行動の概要
1-1	0%	0→8→18→9→2→2→15→1→15→1→15→15→15	1	joomla 以外のアプリケーションのログイン→joomla の SQL を繰り返す
		0→8→2→9→15→15→18→15→1→15→15→15→15	1	
		0→8→9→2→9→18→1→15→15→15→15→15→15	1	
		0→8→2→9→18→33→15→1→15→15→15→15→15	1	
		0→8→1→9→2→18→15→15→15→15→15→15→1	1	
		0→8→2→9→18→15→15→1→15→15→15→15→15	1	
		0→8→9→2→15→18→15→1→15→15→15→15→15	1	
		0→8→9→2→18→15→15→1→15→33→15→15→15	1	
		0→8→9→1→2→15→18→15→15→15→15→15→15	1	
		0→8→2→1→9→18→15→15→15→15→15→15→15	1	
1-2	90%	8→17→1→35→18→9→36→34→32→34→36→36→1	1	tomcat の exploit を繰り返す
		1→17→8→35→18→9→33	1	各アプリケーションのスキャン、ログイン→sugarcrm の exploit
		17→8→35→1→18→9→34→32→33	1	
		17→1→8→9→18→35→33	1	
		17→8→35→18→9→31→33	1	
		17→8→35→1→18→9→33	1	
		17→8→1→9→35→18→31→33	1	
		17→8→9→35→18→33	1	
		17→8→9→35→1→18→33	1	
		17→8→35→18→9→33	1	
2-1	100%	0→8→35→18→2→1→4	1	各アプリケーションのスキャン、ログイン→wordpress のログイン→exploit
		0→8→35→1→2→4	1	
		0→8→1→35→9→2→4	1	
		0→8→9→1→34→35→18→2→4	1	
		0→8→9→35→2→4	1	
		0→8→35→1→18→2→4	1	
		0→1→8→2→4	1	
		0→8→2→4	1	
		0→8→35→9→2→4	1	
		0→8→35→18→2→4	1	
2-2	100%	17→8→35→18→9→28→30	3	各アプリケーションのスキャン、ログイン→webmin のログイン→exploit
		17→35→8→18→9→28→1→30	1	
		17→8→35→18→9→34→28→1→30	1	
		17→1→8→9→35→18→28→30	1	
		17→8→35→18→9→34→1→28→30	1	
		17→8→9→35→18→28→30	1	
		17→8→35→18→28→30	1	
		17→8→1→35→18→28→30	1	
3-1	0%	0→1→8→2→35→18→9→34→18→18→18→1→18	1	各アプリケーションのスキャン、ログイン試行を繰り返す
		0→8→35→9→1→18→34→2→18→19→18→18→18	1	
		0→8→35→18→2→9→1→17→20→20→36→20→34	1	
		0→8→35→9→2→1→34→18→18→18→36→20→18	1	
		0→1→8→35→2→18→9→17→36→36→36→1→19	1	
		0→8→35→18→1→18→34→2→9→18→36→20→36	1	
		0→8→35→18→2→1→9→17→20→19→36→34→36	1	
		0→8→35→2→1→9→34→18→18→18→19→18→20	1	
		0→8→18→9→34→18→18→1→2→35→19→20→20	1	
		0→8→35→1→18→1→2→9→1→34→18→18→36	1	
4-1	80%	23→26→39→40→29	4	phpmyadmin のログイン→SQL インジェクション→クラック→使いまわし→exploit
		23→1→26→39→40→29	3	
		23→26→39→1→40→29	1	
		17→23→26→39→40→40→40→1→40→40→40→40→40→40	1	phpmyadmin のログイン→SQL インジェクション→クラック→使いまわし→繰り返し
		17→23→26→39→40→40→40→40→1→40→40→40→40→40→14	1	
4-2	50%	0→1→23→26→39→40→29	1	phpmyadmin のログイン→SQL インジェクション→クラック→webmin の exploit
		0→23→26→39→40→29	1	
		23→0→26→39→3→40→29	1	
		23→0→26→39→40→1→29	1	
		0→23→26→39→3→40→1→29	1	
		0→23→26→39→3→3→22→1→3→26→26→26→26	1	phpmyadmin のログイン→SQL インジェクション→クラック
		0→1→23→26→39→22→26→26→26→26→26→1→11	1	SQL インジェクション繰り返し
		23→1→0→26→39→31→33→33→7→29→3→1→3	1	phpmyadmin のログイン→SQL インジェクション→クラック→
		0→23→26→39→31→29→7→1→3→29→3→33→26	1	wordpress などへの exploit
		23→0→26→39→22→26→26→3→1→3→11→26→26	1	
5-1	0%	17→23→26→39→40→40→40→1→40→40→40→40→40	1	phpmyadmin のログイン→SQL インジェクション→クラック
		23→26→39→40→1→40→40→40→40→40→40→40→40	1	→reuse 繰り返し
		23→26→39→40→1→17→40→40→40→40→40→40→40	1	
		23→26→39→40→19→40→17→40→1→40→40→40→37	1	
		17→1→23→26→39→40→40→40→40→40→40→1→40	1	
		23→26→39→40→40→40→17→1→40→40→37→40→40	1	
		23→26→39→1→40→40→40→40→40→40→17→40→40	1	
		23→26→39→40→40→40→40→1→40→40→17→40→40	1	
		23→26→39→40→40→40→40→1→40→19→40→40→40	1	
		17→23→26→1→39→37→40→40→40→40→22→40→40	1	
6-1	0%	23→26→39→1→40→17→40→40→40→40→40→40→40	2	phpmyadmin のログイン→SQL インジェクション→クラック
		23→26→39→1→17→40→40→40→40→40→40→40→40	1	→reuse 繰り返し
		23→1→26→39→40→40→40→40→40→40→40→1→40	1	
		23→26→39→17→1→40→40→40→40→40→40→40→40	1	
		17→23→26→39→40→1→40→40→40→40→40→40→40	1	
		23→26→1→39→40→40→40→40→40→40→40→40→1	1	
		23→26→39→40→40→40→40→1→40→40→40→40→40	1	
		23→1→26→39→40→40→40→17→40→40→40→1→40	1	
		23→26→39→40→40→40→1→40→40→17→40→40→40	1	
6-2	0%	0→23→1→26→39→40→40→40→40→40→40→40→1	3	phpmyadmin のログイン→SQL インジェクション→クラック
		0→23→26→39→1→40→40→40→40→40→40→40→40	3	→reuse 繰り返し
		0→23→26→39→40→40→1→40→40→40→40→40→40	2	
		0→23→26→39→40→1→40→40→40→40→40→40→40	1	
		0→23→26→39→40→40→40→1→40→40→40→40→40	1	

Chapter 5　結論

5.1　達成点

今回の実験で達成できたこととしては、以下のようになる。

5.1.1　現実に根差したデータセットを用いての、バージョンからの exploit の発見

現実に存在する exploit と多数のバージョンを持つアプリケーションをデータセットとしたうえで学習を行い、正しい行動を高い確率で見つけ出すことができたため、バージョンに対し深層強化学習を用いることで exploit を発見するタスクが十分に深層強化学習を用いて可能なレベルであると分かった。アルゴリズムが現状のままであっても、今後さらに exploit やバージョン数を増やしていくことで、ペネトレーションテストの補助として十分な役割を果たすことができると考えられる。

5.1.2　認証を含む状態空間での攻撃手法の発見

認証などが必要な条件下であってもその道筋を極めて高い確率で学習することができた。現状からさらに多くのアプリケーションや exploit を学習対象としていくことで、徐々にプログラム任せでも大部分の攻撃を補助することができるようになり、ペネトレーションテストの労力を大幅に減らしていくことができるようになると考えている。

既存のスクリプトを用いた自動化に比べたメリットとしては、状態による分岐を多数考慮に入れてスクリプトを組まずとも、データを使って学習を行うことである程度適した経路を見つけることができ、プログラマーの負担を減らすことができることであると考えている。

5.1.3　ペネトレーションテストの自動化において汎用的なフレームワークの考案

認証とアプリケーションのバージョンは Web アプリケーションのみならずクラウドアプリケーションやスタンドアロンアプリケーションであっても攻撃において重要となる普遍的なものである。そのため、今回の実験において考案したフレームワークもまたある程度の普遍性を持っており、Web アプリケーションに限らず、様々な系統のペネトレーションテストに応用していくことができると考えられる。

5.1.4　LSTM を用いた手法による有用性

モデルの評価およびケーススタディから、実験 1 と 2 のどちらの状況においても LSTM を導入することで、ある地点までの行動の成功・失敗を鑑みた行動を取りやすくなるということが明確になった。この手法を応用していくことで、ペネトレーションテスト過程でのより複雑な状況にも対応しやすくなると考えられる。ただし、学習に必要な時間が大幅に増加すること、多種類の行動を実行するためにかえって一部の状況では不安定化を招きやすいことには留意する必要がある。

5.2　問題点

現状の問題点としては、以下のようなものが考えられる。

5.2.1　拡張性の乏しさ

実際の exploit を用いて行った場合はもちろん、模擬環境での学習を行った場合であっても、実験結果が示すように十分な時間が必要となる。また、exploit をまとめたフレームワークである metasploit はオーバーヘッドが大きく、時間がかかる。既存の exploit を探す、もしくは実装することで時間を削減することは出来るが、その場合実装そのものに手間がかかる。また、exploit を検証するための環境に関しても、セットアップに時間がかかり、エラーに対処する手間もかかる。これら、所要時間が原因となり、拡張を行う際に時間がかかってしまう可能性が高い。

5.2.2　give up の正確性

実験 1、2 のいずれの場合でも、複数のアプリケーションに対し試行を打ち切る give up がうまくいかない場合が多かった。単なる省力化ツールとしてアプリケーション 1 つ 1 つに対して実行する場合であればあまり問題はないが、人の手をほぼ介在させずツール任せにするペネトレーションテストを実現することを最終目標と考えた場合、大きな陥穽となりうる。

5.3　今後の展望

5.3.1　パラメータの最適化

機械学習のモデルには多数のパラメータが存在する。深層強化学習であれば、モデルの層の深さ・ノードの数・学習率などが挙げられる。これらをどのように設定するかによって、同じアルゴリズムを用いても結果は大きく異なるものとなる。

また、状態変数においても論文中において説明したいくつかの表し方以外にもよりよいものがあると推測できる。これらに関しては、一概にどのようなパラメータの設定がタスクの精度を上げると判断できるものではないため、実験を積み重ねる必要が

ある。同じく、報酬関数に関しても様々な表し方が考えられる。今回の学習では学習の安定を優先して give up の制裁を大きくすることが多かったが、結果として give up の成功率を下げてしまった可能性が高く、見直しの必要がある。

5.3.2　より多くのターゲットアプリケーションへの適用

今回作成したアプリケーションは使用率の高いもの、exploit を含め存在している脆弱性の数が多いもので Web アプリケーションを主に選定した。しかし、ペネトレーションテストを行うのであれば、より多くのアプリケーションに対応していなければ総当たりを行うほうが良好となる可能性もある。

一方で、より多くのアプリケーションを状態空間へと含めることは、計算量を増加させ、学習にかかる時間・リソースを大幅に増加させることになるため、それに対する対策も平行して考えていく必要がある。

5.3.3　より多くのツールとの連携

本論文の範囲において作成されたアプリケーションでは、多くのツールは手動で簡易的な実装を行ったため、連携を行ったツールは限られている。より多くのアプリケーションと連携を行い、初期状態の設定やパスワードの取得と exploit との連携、スキャンの効率化などに生かせるよう、状態空間・行動空間などを設定していくことで、より効率よく状況の判断を出来るようにさせていき、自動化に寄与出来るようにすべきである。

5.3.4　より様々な exploit の種類への適応

Web アプリケーションに内在しうる脆弱性には今回の実験で使用したターゲットマシンで任意のコードを実行する RCE、SQL のクエリを挿入しデータベースを操作する SQL インジェクション以外にも javascript のペイロードを挿入する XSS(クロスサイトスクリプティング)、リスティングされていないディレクトリを読むことが出来るディレクトリトラバーサル、テンプレートエンジンへのレンダリングを行うことにより対応する言語でプログラムを実行することの出来る SSTI(サーバーサイドテンプレートインジェクション) など、様々な種類が存在している。これらに関しても環境へのインパクトを CVSS の score などから算出し、それに合致するよう報酬関数を設定しつつ、行動空間に組み込んでいけばより有用性を増すことが出来るだろう。

また、個別の脆弱性の他に、たとえばテンプレートインジェクションであれば TQLMAP など、汎用的に脆弱性を見つけ出すためのオープンソースアプリケーションも多く存在している。これらとも連携していくことで、現存するオープンソースのアプリケーションだけでなく、初見のアプリケーションに対して脆弱性を見つけるような学習も出来ると考えられる。

5.3.5　現実的な状況への最適化

今回は時間対効率を考え、exploit の情報を用いた模擬空間での学習を試みたが、実際の状況に適用する際には現実で起きる問題点を考慮しなければならない。現実におけるペネトレーションテストの問題点として最たるものは、第一章でも述べた通り、ペネトレーションテストそのものの危険性である。例えば Web アプリケーションのペネトレーションテストに際し、既存のパスワードを変更することで exploit を行うタイプを用いると、サービスを破壊してしまう危険性がある。このようなリスクに基づく評価基準も必要となると考えられる。

謝辞

本研究を進めるにあたり、多くの方々にご指導を賜りました。松浦教授からは研究の進め方、論文の書き方など経験の浅く未熟な自分に対しても懇切丁寧に指導してくださり、大変感謝しております。
卒業されました碓井様、宮里様、博士課程の先輩方でいらっしゃいます宮前様、林田様、Kittiphop 様、石井様、修士課程の同級生であります浅野様、林様、技術職員でいらっしゃいます細井様、協力研究員の島田様、田村様、角田様、特任教授の Miodrag Mihaljevic 様、後輩でいらっしゃいます澤田様には、ミーティングの度に様々な質問を頂き、自分一人では思いもよらなかったであろう角度から知見を広げていただきました。様々な手続きを行ってくださり、研究に邁進できるよう取り計らってくださった秘書の鶴山様にも、何度も助けていただきありがたく思います。
最後に、修士過程に至るまで経済的に困窮する中支えてくださった家族に深く感謝の念を表したいと思います。

参考文献

[1] H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2018, 4-4 May 2018 Farmingdale, NY, USA

[2] OWASP top 10, https://owasp.org/www-project-top-ten/ ,accessed 9 July 2021.

[3] Sugandh Shah, B.M. Mehtre, "An Automated Approach to Vulnerability Assessment and Penetration Testing using Net-Nirikshak 1.0", IEEE International Conference on Advanced Communication Control and Computing Technologies, 8-10 May 2014, Ramanathapuram, India

[4] Rajiv Pandey,Vutukuru Jyothindar, Umesh K Chopra, "Vulnerability Assessment and Penetration Testing: A portable solution Implementation", 2020 12th International Conference on Computational Intelligence and Communication Networks (CICN) , 25-26 Sept. 2020, Bhimtal, India

[5] G Jayasuryapal ,P. Meher Pranay ,Harpreet Kaur ,Swati,"A Survey on Network Penetration Testing", 2021 2nd International Conference on Intelligent Engineering and Management (ICIEM) , 28-30 April 2021, London, United Kingdom

[6] Kovačević and S. Groš "Red Teams - Pentesters, APTs, or Neither" 2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO) , 28 Sept.-2 Oct. 2020, Opatija, Croatia

[7] https://www.fsa.go.jp/common/about/research/20180516/TLPT.pdf , accessed 9 July 2021

[8] Jean-Paul A. Yaacoub, Hassan N. Noura, Ola Salman, Ali Chehab, " A Survey on Ethical Hacking: Issues and Challenges", Sun, 28 Mar 2021 arXiv:2103.15072

[9] https://www.lockheedmartin.com/en-us/capabilities/cyber/cyber-kill-chain.html

[10] https://attack.mitre.org/versions/v6/ , accessed July 9, 2021.

[11] https://github.com/nmap/nmap, accessed January 21, 2022

[12] https://nmap.org/ , accessed January 21, 2022

[13] https://www.shodan.io/, accessed January 21, 2022

[14] https://github.com/wpscanteam/wpscan , accessed January 21, 2022

[15] https://github.com/sqlmapproject/sqlmap , accessed January 21, 2022

[16] https://github.com/rapid7/metasploit-framework , accessed January 21,2022

[17] https://github.com/rapid7/metasploit-framework/wiki, accessed January 21, 2022

[18] https://www.powershellempire.com/ , accessed January 21,2022

[19] https://github.com/openwall/john , accessed January 21, 2022

[20] https://github.com/facebookresearch/hydra , accessed January 21,2022

[21] Sven Turpe,Jorn Eichler,“Testing Production Systems Safely: Common Precautions in Penetration Testing”, 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, 4-6 Sept. 2009 , Windsor, UK

[22] Jajodia, Sushil Noel, Steven. (2008). Topological Vulnerability Analysis: A Powerful New Approach For Network Attack Prevention, Detection, and Response. Algorithms, Architectures and Information Systems Security (Indian Statistical Institute Platinum Jubilee Series). 10.1142/9789812836243_0013.

[23] Pike, Ronald E. (2013) "The “Ethics” of Teaching Ethical Hacking," Journal of International Technology and Information Management: Vol. 22 : Iss. 4 , Article 4. Available at: https://scholarworks.lib.csusb.edu/jitim/vol22/iss4/4

[24] Changwei Liu, Anoop Singhal,Duminda Wijesekera, ”Using Attack Graphs in Forensic Examinations”, 2012 Seventh International Conference on Availability, Reliability and Security, 2012 20-24 Aug. 2012, Prague, Czech Republic

[25] "GitHub - risksense/mulval: A logic based enterprise network security analyzer", https://github.com/risksense/mulval ,accessed December 20, 2020

[26] Mehdi Yousefi, Nhamo Mtetwa, Yan Zhang; Huaglory Tianfield,"A Reinforcement Learning Approach for Attack Graph Analysis” , 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 1-3 Aug. 2018, New York, NY, USA

[27] T. Grance, M. Stevens, and M. Myers, “Guide to selecting information technology security products.” National Institute of Standards and Technology, 2003.

[28] Zhenguo Hu, Razvan Beuran, Yasuo Tan,"Automated Penetration Testing Using Deep Reinforcement Learning”, 2020 IEEE European Symposium on Security and Privacy Workshops(EuroSPW) ,22 October 2020, Genoa, Italy,

[29] Konstantin Pozdniakov,Eduardo Alonso,Vladimir Stankovic, Kimberly Tam,Kevin JonesSmart Security Audit ”Reinforcement Learning with a Deep Neural Network Approximator” 2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment(CyberSA),Dublin, Ireland, Ireland,15-19 June 2020

[30] ”13o-bbr-bbq/machine_learning_security”,https://github.com/13o-bbr-bbq/machine_learning_security/tree/master/DeepExploit

[31] Artem Tetskyi, Vyacheslav Kharchenko, Dmytro Uzun, ” Neural networks based choice of tools for penetration testing of web applications”, 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT) ,24-27 May 2018, Kyiv, UKraine

[32] https://www.hackthebox.com/ accessed January 21, 2022

[33] Ovidiu Valea and Ciprian Opris, ”Towards Pentesting Automation Using the Metasploit Framework”,2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP),3-5 Sept. 2020 Cluj-Napoca, Romania

[34] 前田龍星, 三村守, ”深層強化学習による Post-Exploitation の自動化”, 研究報告コンピュータセキュリティ（CSEC）,2020-03-05

[35] https://www.exploit-db.com/ , acccessed November 12, 2021

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov ”Proximal Policy Optimization Algorithms”, arXiv:1707.06347

[37] https://github.com/fofapro/vulfocus , acccessed November 12, 2021

[38] https://github.com/vulhub/vulhub , acccessed January 7, 2022

[39] Tejaswi Kakarla, Aakif Mairaj, Ahmad Y. Javaid ,”A Real-world Password Cracking Demonstration Using Open Source Tools for Instructional Use” 2018 IEEE International Conference on Electro/Information Technology (EIT), 3-5 May 2018, Rochester, MI, USA

[40] Indira Mannuela, Jessy Putri,Michael, Maria Susan Anggreainy ”Level of Password Vulnerability” 2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI) 28-28 Oct. 2021 Jakarta, Indonesia

[41] Rick Wash,Emilee Rader, Ruthie Berman,Zac Wellmer ”Understanding Password Choices: How Frequently Entered Passwords are Re-used Across Websites” SOAPS 2016, JUNE 22-24 2016, denver

発表文献

国際会議

Hajime Kuno and Kanta Matsuura, Towards Automation of Penetration Testing for Web Applications by Deep Reinforcement Learning,Annual Computer Security Applications Conference (ACSAC 2021), Poster Session, 2021.

国内会議

久野　朔, 松浦　幹太. 深層強化学習による Web アプリケーションのペネトレーションテストの自動化に向けて. 暗号と情報セキュリティシンポジウム (SCIS2022) 論文集，2022.