L-020

動的プランニングを用いたサイバー攻撃経路の自動生成
Automatic generation of attack routes with dynamic planning

ブロンモー・ショウロブ‡　　八槇　博史‡
Saurav Brahma　　Hirofumi Yamaki

1. はじめに

サイバー攻撃が複雑になる中，将来的には人工知能による攻撃手法の自動化が行われると考えられる．現在，人工知能は生活におけるあらゆる面で浸透しつつある．例えば，人をサポートするための知能を持たせた人工知能から，携帯端末などにより必要な情報を，対話によって機械的に得るだけでなく，情緒的な会話が可能であることや，株式の自動売買や，がんの早期発見・早期治療への寄与などである．将来的には，ほぼ全ての分野に人工知能が関わって，自動化されるようになると考えられる．これはサイバー攻撃を行う者達にも当てはまるだろう．彼らも人工知能技術を用いて攻撃パターンを増やし，あるいは既存の方法を最適化し，攻撃の規模を拡大していくと考えられる．これと同様に，マルウェアによる攻撃手法や感染手法も進化していくだろう．

本研究ではこのような予想をもとに，特にマルウェアが複雑な攻撃手順を自動生成するシナリオを想定し，そこで用いられるプランニング技法について検討する．

2. 自律型マルウェア

現在の標的型攻撃に用いられる一般的なマルウェアは，まず，ソーシャルエンジニアリングなどを活用しながら巧妙に攻撃対象についての情報を集める．次に，ゼロデイの脆弱性や対策されていない脆弱性などを利用して内部のサーバへ感染し，この感染したサーバから C&C（Command and Control）サーバへ通信を行う．最後に，外部の攻撃者から指令を受け取りつつ LAN 内部へ感染を広げていき，目的とする攻撃行為を達成する．これまではマルウェアが必ず外部の攻撃者と通信を行う必要があったので，C&C サーバを特定・確保することが主要な対策となり得た[1].

しかし，マルウェアが従来のように C&C サーバから指令を逐一受ける必要がなくなった時，上記のような対策は困難になる可能性が高い．何故なら，現在はマルウェアが外部からの指令を必要とすることからこのような対策が可能だが，マルウェアが自律的に動作し，外部からの指令や支援を受けなくても目的を達成することができるのであれば，外部からの異常な通信を検知することができなくなる可能性が高いからである．こうした自律的な動作ができ，攻撃者からの指令を必要としないマルウェアのことを自律型マルウェアと呼ぶ[2]. この自律型マルウェアの基本機能を以下に示す．

- 環境を認識する機能
- 認識した情報から取るべき行動を決定する機能
- 決定した行動を実行する機能

まずマルウェアは感染活動を行うために，対象となる環境を認識する機能がある．次に，この認識した情報から対象とする環境に対してマルウェア自身が取るべき行動を決定する．行動を決定すると，マルウェアは決定された行動を実行する．

本研究では，攻撃者が自律型マルウェアを社内サーバに侵入させた場合を仮定して，目的の攻撃行為を達成するまでの「認識した情報から取るべき行動を決定する基本機能」にあたる「攻撃プラン」を，人工知能の一分野であるプランニング技法を用いて自動生成することを試みる．プランニング技法とは，望ましい状態に至る行為列を見つけることによって行うべきことを決める問題解決エージェントが存在する時に，そのエージェントに与えられた目標を達成するための一連の行為を自動的に生成する技法のことである．これにより，実際に想定される攻撃経路から自律型マルウェアによる攻撃の危険性について考察することが本研究の目標である．

本研究では 3 章で既存方式によるプランニングを「行動決定」に適用した際のマルウェアの挙動について説明し，4 章で既存方式についての問題点を述べる．5 章でその問題点を解決することができると考えられる方式について提案し，その有効性を検証する．6 章ではこれまでのことを踏まえた上で，こうした自律型マルウェアが将来的に登場した時，どのような対策が有効になりうるか考える．なお，本研究における静的なプランニングとはあらかじめ攻撃対象のシステム，およびネットワークの構成がわかっている状態でのプランニングであり，動的なプランニングとはこうしたネットワークの構成がわかっていない状態でのプランニングである．

3. 静的プランニング

既存研究として静的なプランニングである，ダイクストラ法[3]を用いた自動経路生成が存在している[4]. 以下にダイクストラ法を用いて，プランニングを行うアルゴリズムを示す．

3.1 ダイクストラ法

攻撃対象となる情報システムの，各種サーバ，端末，ネットワーク装置をグラフ理論におけるノードとし，それらが回線により接続されたグラフとしてモデル化する．この時，ダイクストラ法に基づく攻撃対象ノードの決定手順は以下のようになる．

1. スタートノード S とゴールノード G を設定し，スタートノード S を優先度付きのキューの OPEN リストに格納する．OPEN リストは未攻撃ノードを格納しておくためのリストである．また，攻撃済ノードを格納しておくためのリストとして，CLOSE リストを空に初期化して用意する．

2. OPEN リストに格納されているノード n を取り出し，現在展開中のノードとする．この時 OPEN リストが空でありノード n を取り出せなければスタートノード S からゴールノード G にたどり着くような経路が存在しなかったとして終了する．ノード n にノード G が含まれているのであれば，そのような経路が存在し，たどり着くことができたとして探索を終了する．経路が存在し，探索も終了した場合は 5. に遷移する．

3. ノード n と隣接しているノードを m とし，現在のノードから隣接しているノードに対するコストである評価関数 $f(n)$ を計算する．この時，$f(n) = g(n) + step(n,m)$ である．
$step(n,m)$とは，ノード n から m へ移動するときのコストである．この「評価関数」については 3.3 節で詳細を示す．

4. 計算後 CLOSE リストにノード n を追加し，OPEN リストにノード m を追加し，OPEN リストにあるノードで最もコストが小さいノード n を選択し 2. に遷移する．

5. 探索終了後，G から S までのノードを順にたどっていくとスタートノード S からゴールノード G までの最短経路が得られる．

以上がダイクストラ法を用いたプランニングアルゴリズムである．後述する実際の実験ではこのアルゴリズムと評価関数を用いて実験を行っている．

3.2 システム構成

この節では，上記のアルゴリズムによる攻撃生成手法の挙動を評価するため構成したシステムの詳細について述べる．実際に使用する全体の環境は，VMware vSphere などの完全仮想化を行うハイパーバイザーを使用する．本研究のホスト OS には Ubuntu 14.04 を導入している．この Ubuntu にはペネトレーションテストなどで使われることも多い Metasploit Framework[5]を導入している．

ローカルネットワーク上では実際に使用されているマルウェアを使うことは難しいので，攻撃と感染の判定は，この Metasploit Framework を使用して行われる．また，Metasploit Framework からの攻撃を受ける，被攻撃サーバには，Metasploitable2 が用いられている．

これらを使用して仮想環境内のネットワークに対して，攻撃と感染を行う．また，Metasploit Framework では攻撃先の IP アドレスや，使用するエクスプロイト，ポートやペイロードの指定と設定を行う必要がある．本研究ではその自動化のため Metasploit Framework のプラグインである autopwn を使用する．この時，Nmap[6]を使用してポートスキャンを行い，IP 情報やポート情報，OS の種類などの情報を獲得し，それらに基づいて autopwn が自動的にエクスプロイトを選択する．そして，その選択したエクスプロイトに必要な設定やペイロードなども自動で設定を行い，当てはまりそうなエクスプロイトを総当たりで自動実行するプラグインである．これらを利用して，攻撃の自動化を行う．

Ubuntu には Metasploit Framework で使用するデータベースとして PostgreSQL[7]が導入されている．また，実際には攻撃が成功するとセッションが立ち上がるが，本研究ではセッションが確立されると攻撃が完了したとして感染の拡大を再現していく．ただし，その感染したマルウェアが Metasploit Framework のモジュールなどを用いて，攻撃が成功したサーバに対してさらに深く被害を与えていくようなことはしない．何故なら，本研究はあくまでもマルウェアによる人工知能を用いた攻撃の自動化が可能であるかどうかに主眼を置いているので，実際に本格的な攻撃を行う必要がないためである．また，攻撃が目的ではないことから，本研究で使用する脆弱性は php に関する脆弱性である，「exploit/multi/http/php_cgi_arg_injection」の 1 つのみとする．

本研究では，2 節で述べた「環境の認識」のために Nmap を使用している．また，Nmap から得られた情報をもとに，マルウェアの「行動を決定する」ためにプランニングを使用している．そして，マルウェアが取るべき行動が決定すると，実際に攻撃を行うための「行動」に移る．この「行動」は本研究においては目的とするサーバに対する攻撃であり，この攻撃に Metasploit Framework を使用している．

3.3 評価関数

この節では 3.1 節で述べた評価関数についての詳細を示す．評価関数とは，現在の状態がどれだけ目的とする状態に近いと考えることができるか，定量的に表現することができる関数のことである．本研究では，攻撃対象となるサーバがどれだけ脆弱であるかを指標としている．

表 1：エクスプロイトの件数(Platform)

Platform	件数
php	17,731
Windows	8,005
linux	2,391
multiple	2,015
asp	1,509
hardware	1,124
cgi	692
unix	305
osx	291
lin_x86	231

表 2：エクスプロイトの件数(Port)

Port	件数
80	963
21	145
8080	89
443	49
143	47
25	35
69	22
8000	21
22	18
445	16

表 1 は Metasploit Framework に登録されている Platform の脆弱性の件数である．また，表 2 は Metasploit Framework に登録されている Port の脆弱性の件数である．実験で使用している評価関数 $f(n) = g(n) + step(n,\ m)$ について，詳細を以下に示す．

$g(n)$は実コストとする．表 1 から php は 17,731 件のエクスプロイトがあり，最も登録件数が多いのでコストが最も低いとし，platform の値を 0 とする．次に Windows の 8,005 件が最も多く platform の値を 1 とし，次に多いのが linux の 2,391 件であるので platform の値を 2 とする．同様に multiple，asp，hardware，cgi，unix，osx，lin_x86 の順にコストが 1 つずつ高くなっていく．ポートにおいても同様に 80 が最も登録件数が多いので port の値を 0 とし，次にポート 21 の 145 件が多いので，port の値を 1 とする．次に 8080 は三番目に件数が多く，89 件だったので port の値を 2 とする．同様に 443，143，25，69，8000，22，445 の順にコストが 1 つずつ高くなっていく．また，platform や port にそれぞれ値が二つ以上存在していた場合は，platform では最も件数の多い php を 10 とし，platform の元々の値から 2 つ目以降を引く．例えば，1 つ目が php で 2 つ目が Windows であれば，0－(10 - 1)となり，コストは -9 となる．port も同様の計算を行い，コストを計算する．サーバ n の platform と port のそれぞれの計算後に $g(n) = platform + port$ として計算を行い，実コストを求める．また，表 1 に存在しない port や platform があった場合は，コストを 11 として計算を行う．

次に，$Step(n,\ m)$ について，サーバ n とは現在のサーバの位置を指しており，サーバ m とは，隣接しているサーバのことを指す．$Step(n,\ m)$ とは，サーバ n からサーバ m まで移動したコストをステップ値として，+1 をする．しかし，スタートサーバ S から隣接しているサーバへのステップ値は 0 とする．

計算終了後にサーバ n を CLOSE リストに，サーバ m を OPEN リストにそれぞれ追加し，OPEN リストの中で最もコストが低いものから選択し，攻撃を行う．また，全体のコストが一緒であった場合，どちらか一方をランダムで選択する．最後に探索終了後に目標を達成できていれば，ゴールサーバ G からスタートサーバ S までを順に追っていくことでスタートサーバ S からゴールサーバ G までの最短経路を求めることができる．

3.4 実験

この節では前節までの条件を適用するために構築した実験環境を示す．以下の図 1 はダイクストラ法を用いてプランニングを行った時の実際のネットワークの構成である．表 3 は図 1 に示されている脆弱性を持っているサーバごとの環境である．

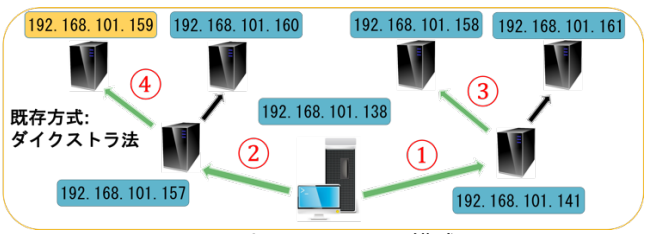

図１：ネットワークの構成

表３：脆弱性を持つサーバごとの環境

サーバ名	Platform	Ports
192.168.101.141	Ubuntu	80,21,25,22,445
192.168.101.157	Ubuntu	80,21,25,22
192.168.101.158	Ubuntu	80,21,25,445
192.168.101.159	Ubuntu	80,21,25,22,445
192.168.101.160	Ubuntu	21,25,22,445
192.168.101.161	Ubuntu	80,25,22,445

表 3 に関する説明と図 1 に示されている実際の攻撃ステージについての説明を以下に示す．

前述の通り，図 1 には実験で使用された実際のネットワークの構成が示されている．この中のそれぞれのサーバに関して，Metasploitable2 は Ubuntu がもととなっているので，IP「192.168.101.138」以外も表 3 の Platform は Ubuntu としている．また，これらの Metasploitable2 には事前に表 3 に示されているそれぞれのポートが開いている状態である．以下は図 1 の攻撃ステージについての説明である．

① まず，マルウェアが IP「192.168.101.138」のサーバへの侵入に成功したことを前提としてプランニングが始まる．プログラムの中に記述されている IP のリストから最初のターゲットとして IP「192.168.101.141」と IP「192.168.101.157」のサーバに対してどちらに攻撃を先に行うかコストの計算を行う．その結果，空いているポートは IP「192.168.101.141」のサーバの方が多いので，攻撃するコストが低いものとして判断する．従って，IP「192.168.101.141」を先に攻撃対象として設定して，接続することができることを確認した後，攻撃を開始する．今回使用している環境は Metapsloitable2 であるので，攻撃は成功する．

② 攻撃の成功を確認すると，経路としてリストに登録をする．これは攻撃が成功したサーバの情報を登録することと，最短経路を導出するためのリストである．次に，攻撃を行う対象として IP を取得する．今回は IP「192.168.101.141」に対して攻撃が成功したので，取得した IP は「192.168.101.158」と「192.168.101.161」である．次に，これまで取得した IP を持つサーバの中でコストを再計算し，OPEN リストを並び替える．その結果，一番小さいコストをもつサーバの IP は「192.168.101.157」であるので次に IP「192.168.101.157」を攻撃する．これも先程と同様に Metasploitable2 であるので攻撃が成功し，経路としてリストに登録をする．

③ その後，先程と同様の手順でプログラム中から IP「192.168.101.159」と IP「192.168.101.160」を取得する．取得後，再び OPEN リストをコスト順に計算して並び替える．その結果一番小さいコストをもつサーバの IP は「192.168.101.158」であるので次に IP「192.168.101.158」を攻撃する．この攻撃も成功するが，ゴールではないので感染活動はまだ続くことになる．

④ 次に攻撃するサーバとして計算された IP は「192.168.101.159」である．これも Metasploitable2 であるので攻撃は成功する．すると，ゴールサーバの情報として今回指定されているのはこのサーバの IP であるのでゴールであることが，攻撃が成功したこの時点でわかることになる．ここでゴールサーバまで攻撃が成功したので，リストにあらかじめ登録しておいた情報を逆にたどると，最短経路を導出することができる．今回の実験で得ることができた最短経路は「192.168.101.138」→「192.168.101.157」→「192.168.101.159」となる．

このことから，ダイクストラ法による攻撃経路の静的生成が可能であることがわかる．

3.5 静的プランニングの課題

静的プランニングによる手法には大きく分けて二つの問題点がある．

一つ目は使われているアルゴリズムが静的なダイクストラ法を元にしているため，プランニングをするにあたってあらかじめ攻撃対象となるネットワークの構成を把握しておかなければならないという事である．例えば図 1 に示した例だと，IP「192.168.101.141」が攻撃済リストに入っており，IP「192.168.101.158」と IP「192.168.101.161」が攻撃済リストに入っておらず，IP「192.168.101.158」の情報が Nmap によりスキャンされていない場合，IP「192.168.101.158」と IP「192.168.101.161」を未攻撃 IP として OPEN リストに追加する命令が直接記述されている．もし図 1 のネットワークの IP アドレスが書き換えられてしまった場合や，何らかの理由で攻撃を行う前にネットワークの構成が変更されてしまった場合，静的プランニングでは対応をすることが難しい．また，ネットワークの構成を入力するのはこの時点では人間なので，たとえ十分攻撃行為を遂行することができるマルウェアであっても，人間の IP アドレスを指定する際の設定ミスによって，失敗することも考えられる．さらに，実際のサイバー攻撃においてはネットワークの構成をあらかじめ把握して，外部から攻撃を行うことは困難である場合が多い．ネットワークの構成を把握している内部の人間によっての攻撃など，非常に限定されたシナリオでの動作になってしまうことが考えられる．よって，実際にマルウェアが内部ネットワークに侵入して C&C サーバと通信せずに目的を達成するためには，マルウェア自身が内部のネットワーク状況を把握していきながら感染を拡大していく必要がある．

二つ目の問題点は，攻撃失敗時の挙動である．既存方式では，攻撃が失敗した場合が想定されていない．しかし，プランニングはあくまでも推定である．推定上は正しくとも，実際の攻撃時には推定通りに攻撃が成功せず，失敗することもあり得る．例えば，図 1 のネットワークでは Metasploitable2 が使用されているので，攻撃は必ず成功するようになっていることは 3.4 節で述べた通りである．しかし，現実のシステムでは全ての環境が脆弱である可能性は極めて低く，そういったシステムに対してはどういう挙動を取るべきなのかを考慮する必要がある．図 1 では IP「192.168.101.138」から，IP「192.168.101.141」と IP「192.168.101.157」の二つのサーバのどちらかに対して攻撃を行うが，もし片方のサーバに対して攻撃が失敗した場合，そのサーバからこれ以上の感染や情報摂取などは困難だろう．しかし，攻撃が失敗した理由が，マルウェアが特定されてしまって完全に活動が遮断されてしまったなど，マルウェア自体に問題があるのではなく，例えば，ただ単に故障による電源断などであれば，まだマルウェア自体は活動を継続する事ができる．もし，攻撃が失敗したサーバからのみ目的とするサーバにたどり着くプランがないのだとすれば，そのサーバに対する攻撃を成功させる必要が出てくる．その場合は，本研究で検討するようなプランニングの手法ではなく，この手法を適用するマルウェアの完成度に依拠する．しかし，攻撃が失敗しても，他のプランを適用することで目的とするサーバにたどり着くことができるのであれば，マルウェア自体に問題があり，攻撃のみならずプランを形成することさえ困難になるまで攻撃を続け，感染を拡大するべきである．ただし，これはその時マルウェアが達成するべき目標や置かれた状況による．従って，攻撃が失敗した時，異常終了をするのではなく，攻撃を再開させる必要がある．

4. 動的プランニング

こうした上記のそれぞれの問題は，従来の静的プランニングではなく，動的プランニングによる自動攻撃を導入することで解決することができると考えられる．動的プランニングを導入することで，静的プランニングでは解決することが困難なネットワークの構成を把握していない状態での攻撃経路の生成や，攻撃が成功しなかった場合の動作の選択など，より複雑な行動の選択が可能になると考えられる．これらの詳細はそれぞれ 4.1.1 節と 4.1.2 節で述べる．

そこで，RBFS（Recursive Best-First Search）法[8]を用いたプランニングアルゴリズムを構築した．RBFS は，最良優先探索の一種であり，通常の最良優先探索のオペレーションを模倣しながら線形の空間だけを使う再帰的アルゴリズムである．このことから，再帰的最良優先探索と呼ばれる．

4.1 Recursive Best-First Search

この節では RBFS についてのアルゴリズムとその動作を，以下に示した図 2 を用いて述べる．図 2 に示されているゴールノード G までの最短経路を求めるステージについての説明を以下に示す．ここで，今回のスタートノードを S，ゴールノードを g とする．

① スタートノード S を設定する．S の隣接ノードに遷移して展開するコストをそれぞれ算出する．各ノードの右側に示されている数値がそのコストである．S の隣接ノード A, B, C の中で最小コストを持っているノード A に遷移する．この時，葉ノード（今回の場合 S の隣接ノード）の中で，A を除いて一番小さいコストである，第二最小コストを持っているノードとそのコストを記憶しておく．次に，A の隣接ノードである D, E, F の中で最小コストを持っているノード F に遷移する．この時，先と同様にこの時点にお

いて展開した葉ノードの中で第二最小コストを持っているノードとそのコストを記憶しておく．

② 本来 F の次に遷移するノードとしては，展開された H, I, J の中で最小コストを持っている I であるが，F の第二最小コストを見ると，I より少ないコストで遷移する事ができるノードが I の他に存在している事がわかる．そこで，再帰呼び出しを後戻りして，忘れられていた部分木の最良の葉ノードのコストが F に保存される．その後，F の第二最小コストを持っているノードとして保存されていた D に遷移して展開されるが，葉ノードの中で最小コストを持っているノードである L が先ほど展開された I より大きいコストを持っており，なおかつ，ゴールノード g ではなかった事が明らかになる．

③ そこで，先ほどと同様に再帰呼び出しを後戻りして，忘れられていた部分木の最良の葉ノードのコストが D に保存され，現時点で最小コストを持っている F が再び展開され，I に遷移する．I を展開し，最小コストを持っているノードに遷移するとゴールノード g に到達する事ができる．

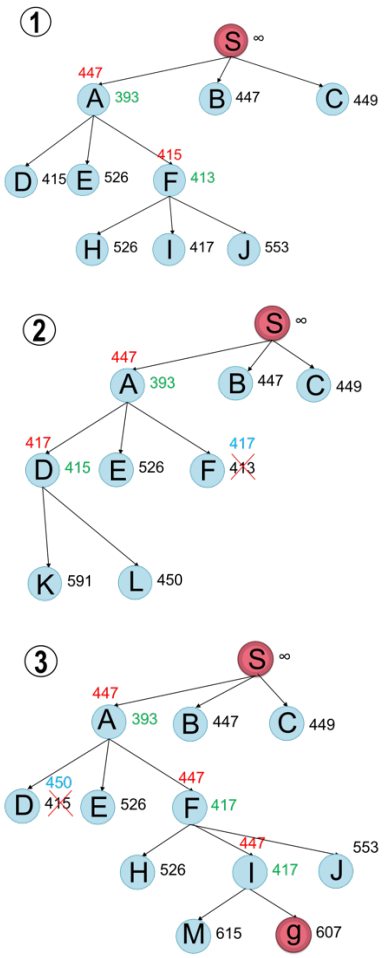

図２：Recursive Best-First Search

また，以下に今回用いた RBFS についての具体的なアルゴリズムを示す．

1. スタートノード S とゴールノード G を設定し，スタートノード S を優先度付きのキューの OPEN リストに格納する．OPEN リストは未攻撃ノードを格納しておくためのリストである．また，攻撃済ノードを格納しておくためのリストとして，CLOSE リストを空に初期化して用意する．

2. OPEN リストに格納されているノード n を取り出し，現在展開中のノードとする．この時 OPEN リストが空でありノード n を取り出せなければスタートノード S からゴールノード G にたどり着くような経路が存在しなかったとして終了する．ノード n にノード G が含まれているのであれば，そのような経路が存在し，たどり着くことができたとして探索を終了する．経路が存在し，探索も終了した場合は 7. に遷移する．

3. ノード n と隣接しているノードを m とし，現在のノードから隣接しているノードに対するコストである評価関数 $f(n)$ を計算する．この時，$f(n) = g(n) + step(n,m)$ である．
$step(n,m)$ とは，ノード n からノード m へ移動するときのコストであり，評価関数の概要は 3.3 節で述べたものと同様である．この時，第二最小コストを計算し，そのコストを持つノード k を記憶しておく．

4. 計算後 CLOSE リストにノード n を追加し OPEN リストにノード m を追加し，OPEN リストにあるノードで最もコストが小さいものを選択する．これをノード t とする．

5. 4.で選択されたノード t のコストがノード k のコストより大きかった場合，現在のノード n を閉じて最小コストをノード k の持っているコストとして更新する．

6. ノード k を OPEN リストに追加して，再び第二最小コストを更新し，2. に遷移する．

7. 探索終了後，ゴールノード G からスタートノード S までのノードを順にたどっていくとスタートノード S からゴールノード G までの最短経路が得られる．

4.1.1 システム構成が未知の場合の解決法

まず，それぞれのサーバが隣接しているサーバの IP アドレスを内部に記憶していると仮定した．サーバに対しての攻撃が成功すると，この情報を参照する事ができる．マルウェアは攻撃を行いながら，IP アドレスを収集していく事で同時にネットワークの構成を把握していく．こうする事で人間が外から指示を与えなくても自律的にネットワークの構成を把握し，攻撃を実行する事ができる．

4.1.2 攻撃失敗時の解決法

静的プランニング時は，まずマルウェアがネットワークの中でプランを立ててから攻撃を実行していた．つまり，プランニングと攻撃のフェーズが分けられており，攻撃が失敗すると，その時点でプロセス自体が終了していた．そこで，プランニングフェーズと攻撃フェーズを合同にした．こうする事で，これまでプランを立ててから攻撃を実行していたものから，攻撃をしながらプランも同時に立てるようにする事ができた．また，こうする事で，攻撃が失敗した時にそこで終了するのではなく，一つ前に戻ってそこからプランニングを再開させる事ができるようになった．この時攻撃が失敗したノードは攻撃済ノードとして記憶しておき，二度と攻撃しないようにした．

これらの要件を入れてプランニングアルゴリズムを実装した．

4.2 実装と環境

基本的な構成は 3.4 節で構築した環境に沿って実装した．上記の要件を満たしているかを確認するために以下の変更を施した．

1. 今回，プログラムの中に直接 IP は記述されておらず，それぞれのサーバの中に次の遷移先のサーバの IP を記憶していることにする．このためにサーバの中にはそれぞれ次の遷移先の IP アドレスを保有するテーブルが存在している．

2. 攻撃が失敗した場合の挙動を確認する必要がある．そこで，IP アドレス「192.168.101.141」を持っているサーバに関しては攻撃コストを算出するための環境情報である，「Platform」と「Ports」は変更しないが，使用した Metasploit Framework に登録されている脆弱性である，「exploit/multi/http/php_cgi_arg_injection」に対して対策を施し，攻撃が失敗するようにした．具体的には，Metasploitable2 ではなく，Ubuntu を使用した仮想環境を新しく用意してポート情報を設定した．

これに基づいて，RBFS を用いた自動攻撃が想定通りに可能であるかの検証を行った．

4.3 実験・評価

4.2 に従って作成した環境を以下の図 3 に示す．

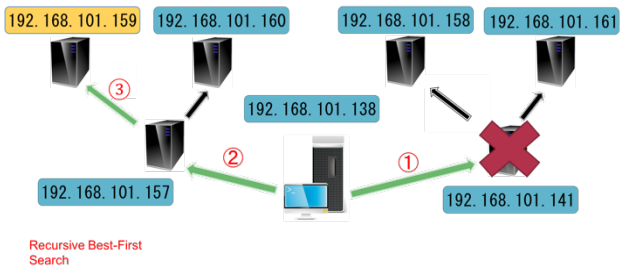

図３：ネットワークの構成

上記の図 3 に従ってネットワークを構築した後，実際に従来のダイクストラ法を用いたプランニングが失敗することを確認するために，従来のプログラムを動作させて実験を行った．すると，想定通り IP「192.168.101.141」と IP「192.168.101.157」のサーバに対してコスト計算を行った後，攻撃を IP「192.168.101.141」に対して行ったことを確認した．またその後，IP「192.168.101.141」に対して攻撃を行っていたが，前述の通りこのサーバは今回攻撃のために指定した脆弱性に対して対策を施したサーバであったので攻撃が失敗して異常終了した．図 3 の IP「192.168.101.141」に書かれている大きなばつ印はこのサーバに対する攻撃が失敗したことを示している．このことから想定していた「プランを立てた後でも攻撃が失敗して異常終了をしてしまう可能性がある」ことを確認することができた．

想定した通りのネットワークを構築することが確認できたので，次に，RBFS を用いたプランニングを適用して実験を行った．想定は以下の通りである．実験時はこの通りに正しく動作するか確認した．

① まず，IP「192.168.101.138」に登録されているサーバである，IP「192.168.101.141」と IP「192.168.101.157」のサーバを未攻撃サーバとしてリストに登録した後，コストの計算を行う．この結果，IP「192.168.101.141」の方が，コストが小さかったので先に攻撃をする．しかし，これは脆弱性対策済みのサーバであったので攻撃は失敗する．

② ここで，マルウェアは攻撃が失敗したものの，まだ活動を続けることができるものとして，今回攻撃が失敗したサーバの IP を二度と攻撃しないように，攻撃済みサーバとしてリストに登録をする．そして攻撃が成功していた IP「192.168.101.138」から再び活動を再開する．IP「192.168.101.138」から見ることができるサーバは IP「192.168.101.141」と IP「192.168.101.157」のサーバであるのでもう一度コスト計算を行う．しかしこの時，IP「192.168.101.141」は攻撃済みサーバであるため，攻撃対象からは除外する．従って，残ったサーバである IP「192.168.101.157」のサーバに対して次の攻撃を行う．こちらは対策を施していない Metasploitable2 であるので攻撃が成功する．

③ 攻撃が成功したのでマルウェアはここからさらに感染を拡大する活動を行う．IP「192.168.101.157」に登録されているサーバは IP「192.168.101.159」と IP「192.168.101.160」である．この情報を取得して，コスト計算を行う．この結果 IP「192.168.101.159」のコストがより小さいことがわかったので次にこのサーバを攻撃する．このサーバも脆弱性の対策を行っていないので攻撃は成功する．今回はこの IP をもつサーバに対して攻撃を成功させることが成功条件であったので，これでプランニングは終了する．

以上が実験前の動作の想定であるが，実際の実験でもこの想定通りにプランニングすることができた．また，実験を開始した時から終了するまでにかかった時間は 6 分 54 秒で

あった. 今回のような規模のネットワークだと, これは時間がかかりすぎていると考えられる. この時間のほとんどは, 総当たりの Nmap によるポートスキャンであった.

4.4 考察

この実験結果からローカルネットワーク内でのネットワークの構成が未知である場合においても人工知能を用いた自動攻撃が可能であることがわかった. このことから攻撃者が社内ネットワークに侵入を一度成功すると, そこからは攻撃者が直接指示を与えることがなくても, 自律的に攻撃を行っていくことが考えられる. さらに, 攻撃者は標的型攻撃において, 対象とする組織の情報のもと, 終了条件など一定のパラメータを変更する必要があるが, 組織内への侵入を果たした後は直接マルウェアと関わる必要がなくなる可能性が高い.

しかし, 本研究で構築したネットワークは提示した条件を満たす最低限のものとして構築したものであり, 非常にシンプルなネットワークである. 実務的に利用されているネットワークは当然今回のものよりさらに複雑で巨大なネットワークであるはずなので, 今回の実験で未知のネットワークに対して完璧にプランニングを行うことが可能であったと結論付けるのはまだ難しいと考えられる. また, 今回は一例としてサーバが隣接しているそれぞれのサーバの IP を保有しており, マルウェアは攻撃を成功させることでそれらのサーバの IP を獲得できるとして設定したが, これはネットワーク管理者が DHCP などのプロトコルを使わずにネットワークを静的に管理しており, これらの情報をサーバに登録しているという前提であるので, 比較的小規模なネットワークでの実装になっている. しかし, 目的の達成までの間, IP の再取得などが行われない限り, 今回示したような動的プランニングは動的なネットワークでも対応をすることができる. また, 今回のネットワークは小規模であったが, 大規模なネットワークになると探索にかかる時間も比例して大きくなっていく. 前述の通り今回のネットワークでは 6 分 54 秒であった. このかかった時間のほとんどは Nmap によるポートスキャンにある. 今回は総当たりにてポートスキャンを行っているが, 使用する脆弱性についてある程度決定することができるのであれば, 必ずしも総当たりを行う必要はない. 例えば, ネットワークを把握することはできなかったが, 前情報を収集する段階で, 使われていないことが予想できるシステム, あるいは使われていることが予想できるシステムなどがあれば, それらに関係ないポートについてはスキャンをかける必要がないので, スキャンの対象から外すなどといったことができる. また今回の場合だとスキャンをしても使用するポート情報はウェルノウンポートのみであったので, スキャンに関してはさらに時間を縮小することができると考えられる.

これらのことから, 動的プランニングを用いた未知ネットワークに対しての自動攻撃を行うことは可能であると考えられる.

5. 攻撃への対策

この節では本研究で紹介してきたような, 人工知能による自動攻撃が起きたときに取ることができると考えられる対策案について示す.

5.1 サイバーデセプション

本研究で使われたプランニング方式は, コスト計算によって, 攻撃が通る可能性が高く, 脆弱であることがある程度推定できるサーバに対して攻撃を優先的に行っている. 従って, ハニーポットなどの不正アクセスを誘導するようなシステムを導入してマルウェアを誘い込んで特定・確保することが可能になる場合が考えられる. だが, ハニーポットはグローバルのアドレス空間で動作するサーバに置かれ, 攻撃者の攻撃内容を観察・分析することが主な目的である. これに対して, ハニーポットの応用概念のサイバーデセプション(Cyber Deception) [9]が存在している. これは攻撃者が行う識別プロセスや不正アクセスの妨害・除外・偽装・策略と自動化ツールの中断・活動の遅延・妨害などを行うテクノロジである. 従来のハニーポットと違い, これは標的型攻撃による内部侵入が成功してしまった場合に, 攻撃者が目的とする攻撃行為を達成させないように惑わしたり阻害したりすることができる. サイバーデセプションは, 一種の囮サーバをプライベートのアドレス空間で動作するサーバに置かれ, 攻撃者からの攻撃を防御または遅延させ, 検知することを主な目的としている. また, ハニーポットは攻撃を観察・分析することを主な目的としているので, 攻撃者を取り締まることが目的ではない. このことから特に時間的な制約は存在しない. しかし, サイバーデセプションが機能する時は, すでに組織内部に侵入されているというクリティカルな状況下にある場合がほとんどなので, 情報漏洩の危険性があることから, 検知してからのアクションに時間の制約が存在する.

5.2 SIEM

また, サイバーデセプションの他に, SIEM(Security Information and Event Management)[10]の運用が対策として挙げられる. SIEM はネットワーク内に存在する様々なセキュリティ, ネットワーク機器のログを収集して一元管理し, 集めてきたログの内容を横断的に分析することで異なるログの間をまたがった「相関分析」を自動的に行ってくれるシステムである. 今回示したプランニングはコスト計算を行うために, Nmap を利用したポートスキャンを行っている. このことからポートスキャン時のトラフィックを観測することで通信を遮断することも可能であると考えらえる. しかし, 4.4 節で示した通り, 使用する攻撃モジュールなどの制限ができる場合, 同様にポートスキャンにおいても総当たりでスキャンをする必要がないこともあるだろう. このとき, しっかり観測することができない可能性が挙げられる. また, 相関分析を行うためのパラメータが複雑化してしまうことから安定的に稼働するまでのコストも相対的に高くなる.

以上の二つの方法を組み合わせることで, 対策案の構築を行うことができると考えられる. 人工知能を用いた攻撃

の自動化は人間が同様のものを行う場合と違って，タイピングミスや判断による攻撃の遅延などのヒューマンエラーが存在しないので，対策が難しいものの１つだろう．何故なら，ヒューマンエラーが起きた場合はそれ自体がヒントになり攻撃を特定することが可能になる場合があるが，人工知能を用いることで攻撃をより素早く，より最適化した手法で攻撃が行われると考えられるからである．

6. おわりに

これまでに示した通り，機械学習などのデータ量を利用した攻撃以外に人工知能を利用した攻撃の自動化手法がこれから出てくることが考えられる．この時，従来の対策のみでこれらの攻撃を全て防ぐことは難しい可能性が高い．このことから，今の内にこうした攻撃手法への対策を考えていく必要がある．また，この手法をさらに応用することで，例えば，長期的に組織内のネットワークに対して複数のマルウェアを潜伏させておき，ある一定の条件を満たした時にそれぞれのマルウェアが一斉に特定のネットワークに対して攻撃をすることもできる．

また，このシステムは攻撃法のみではなく，防御側に適用することもできる．例えば，ペネトレーションテストの補助的なシステムとして運用する手段がある．これまでは人間がネットワークの弱い部分を診断し，レポートにしていたが，今回のプランニングを応用することで，人工知能が代わりにネットワークにおいて弱い部分を診断することができるようになる可能性がある．

今後の本研究の目標としてはより複雑なネットワークを構築し，スキャン時の調整や必要な情報の収集法などの攻撃手法を再現して，より具体的な対策に関して検討していく．

参考文献

[1] 独立行政法人　情報処理機構セキュリティセンター “IPA テクニカルウォッチ 『新しいタイプの攻撃』に関するレポート”, 2010
[2] 八槇 博史 “人工知能技術を用いた標的型サイバー攻撃に関する一考察”, 電子情報通信学会, 2016
[3] Dijkstra，Edsger W．“A note on two problems in connexion with graphs.” Numerische mathematik 1.1 269-271．(1959)
[4] 齊藤 悠希 “自動プランニングを用いたサイバー攻撃手順の生成”，2017
[5] “Metasploit”．www.metasploit.com，(参照 2018-05-21)
[6] “Nmap リファレンスガイド”．https://nmap.org/man/jp，(参照 2018-05-21)
[7] “PostgreSQL 日本語マニュアル”，(参照 2018-05-21)
[8] Korf，R.E．“Linear-space best-first search.” Artificial Intelligence．62(1):41–78．(1993)
[9] Rowe，Neil C．“Deception in defense of computer systems from cyber-attack” The NPS Institutional Archive DSpace Repository．(2007)
[10] D. Miller，S. Harris，A. Harper，S. VanDyke & C. Blask．“Security Information and Event Management(SIEM) Implementation.” McGraw-Hill Education. (2011)