# Exploit に対する WAF シグネチャ自動生成の研究

越智　勇貴[†1]　安部剛[†1]

**概要**：公開サーバで使われるソフトウェアの脆弱性を狙った攻撃に対して Web Application Firewall（WAF）は有効な対策の一つである．近年，WAF の検知ルールであるシグネチャを自動的に生成する技術が求められている．本研究では形態素解析，Longest Common Subsequence（LCS）を利用し，脆弱性を突く文字列（exploit）の特徴を抽出しシグネチャを生成する．結果，シグネチャ作成に使用した攻撃コード以外の，同じ脆弱性をつく別の攻撃コードも検知可能なシグネチャ自動生成を可能にした．また，本方式を応用し Web 空間から同じ脆弱性を突く別の攻撃コードを自動抽出することを可能にした．

**キーワード**：脆弱性，攻撃コード，exploit，WAF，シグネチャ

# A Study for WAF signature generation for exploit

Yuki Ochi[†1]　Tsuyoshi Abe[†1]

***Abstract***: Web Application Firewall (WAF) is effective for attacks on vulnerabilities in web server software. In recent years, there is a need for a system of automatic signature generation. In this study, we use the morphological analysis and Longest Common Subsequence (LCS) against attack codes to extract the features of the string that exploit vulnerabilities. As a result, we success to generate signatures which can detect another attack codes that exploit the same vulnerability other than the attack codes that were used to create the signatures. Furthermore, we can show that it is possible to automatically extract another attack codes that exploit same vulnerabilities from the Web space by applying the present method.

***Keywords***: vulnerability, attack code, exploit, WAF, signature

## 1. 背景

### 1.1 Web サーバで使用されるソフトウェアの脆弱性を悪用した攻撃の脅威

近年公開サーバに対するサイバー攻撃が大きな脅威となっている．特に Struts や PHP 等，一般的に広く使用されるソフトウェアの脆弱性を悪用する例が問題となっている．例えば Apache Struts の脆弱性（CVE-2014-0094[a]）は値の適切なチェックが行われないことに起因する脆弱性であり，悪用されると任意のコード実行につながる．また Bash の脆弱性（CVE-2014-6271）は環境変数の解析方法に起因する脆弱性であり，悪用されると任意の Bash コマンド実行につながる．このようなリモートから Web サーバにコード実行が可能になる脆弱性が悪用されることで情報漏えい，Web 改ざん等様々なインシデントにつながる．Web サーバの脆弱性を突く攻撃は The Open Web Application Security Project（OWASP）の Top 10 Most Critical Web Application Security Risk に該当する非常に大きな脅威である[1].

### 1.2 Web サーバで使用されるソフトウェアの脆弱性を悪用した攻撃の脅威

Web サーバで広く使用されているソフトウェアの脆弱性が発見されると，多くの場合攻撃コードがインターネット上に公開される．まず 1 つ目の特徴として攻撃コードが公開されるとすぐに攻撃が急増する．例えば Bash の脆弱性（CVE-2014-6271）においては攻撃コードが公開されると数時間で攻撃が観測され始め，その後攻撃が急増した[2]. 次に同じ脆弱性を突く攻撃コードの亜種についても同様にインターネット上に次々と公開されるという特徴がある．

### 1.3 Web Application Firewall (WAF)

WAF は Web アプリケーションの脆弱性に対する攻撃に有効な防御システムである．また Web サーバ（Apache や IIS 等），ミドルウェア（Tomcat や Struts 等），プログラミング言語処理系（PHP 等）の脆弱性に対しても防御能力がある．検知方法にはアノマリ型とシグネチャ型がある．アノマリ型はプロトコル仕様などと比較して統計的に異常なトラヒック等を分析し，攻撃として検知するものである．この検知方法は未知の攻撃を防ぐ可能性もあり，有効な検知方法である．一方，シグネチャ型はあらかじめ攻撃文字列をシグネチャとして定義し，シグネチャと一致する内容の通信をブロックする検知方法である．この検知方法は既知の攻撃コードを確実に検知し，同じ脆弱性を突く攻撃コードの亜種も検知できる可能性がある．任意のコード実行

* †1 NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, 3-9-11, Midori-cho, Musashino-shi, Tokyo 180-8585, Japan
a)CVE-ID:共通脆弱性識別子であり，脆弱性に対して固有の識別番号を付与されたものである．

につながるような重大な脆弱性の対策を実施する場合，既知の攻撃パターンは確実に防ぐ必要がある．そのため本研究ではシグネチャ型 WAF に着目する．

### 1.4 シグネチャ作成の現状と課題

シグネチャ作成者は，様々な脆弱性情報や攻撃方法に関する情報を収集し，攻撃に特徴的な文字列のパターンを抽出する．その際，正常な通信は検知せず，攻撃のみを漏れなく検知するようなパターンを選択する必要がある．そのようなパターンをシグネチャとして記述する際には正規表現を用いることが一般的である．正規表現は文字列の集合を一つの文字列で表現する方法である．シグネチャ作成側が事前に想定したパターン以外の攻撃コードが確認された場合には，パターンを見直してシグネチャを更新する必要がある．新たな攻撃方法が次々と発見される一方でシグネチャを作成できる専門家は限られており，対策が求められている．

## 2. 研究概要

本テーマではシグネチャ自動生成手法を提案する．以下の 2 つの自動化を検討する．

(1) **攻撃コードの自動収集器**

脆弱性に関する記事や通常のニュース記事など様々なカテゴリの記事群から同じ脆弱性をつく攻撃コードのみを自動的に収集する攻撃コード収集器

(2) **シグネチャの自動生成器**

攻撃コードから自動的に正規表現を用いたシグネチャを作成するシグネチャ生成器

攻撃コードの自動収集器については 4 章でシグネチャの自動生成器については 3 章で述べる．

## 3. シグネチャの自動生成器

### 3.1 従来手法

Ralf S.はある時点で既知の攻撃コードを入力値として，オープンソースの WAF である ModSecurity のシグネチャを出力する手法を提案している[3]．この手法によって生成されるシグネチャの正規表現部分は入力した複数の攻撃コードに連続して共通する部分を固定し，それ以外の文字列を OR でつないだものである．例えば，“HELLO”, “HELKK”, “HEKK”, “HELMO”, “HERIK” を入力すると “(?xism:HE(?:L(?:[LM]O|KK)|(?:RI|K)K))” を出力する．このため，入力に使用した既知の攻撃コードを検知することは可能であるが，同じ脆弱性をつく別の攻撃コードを検知することはできない．

### 3.2 提案手法

攻撃コードは脆弱性をつく部分である exploit，脆弱性をついた後に任意コード実行を行う部分である payload, それ以外の部分に大きく分類できる．同じ脆弱性をつく攻撃コードの亜種同士を比較した際，特に exploit 部分は類似した特徴を持つ．本提案手法では，このような特徴を利用して入力した複数の既知攻撃コードに対して以下のような処理を行う．まず，どの攻撃コードにも共通して現れる文字列パターンを抽出することで exploit に特徴的なパターンを抽出する．次に exploit に特徴的なパターンを固定しそれ以外の要素を正規化することでシグネチャを作成する．このようなシグネチャ作成を実現するために攻撃コードの形態素解析，複数攻撃コードの LCS（Longest Common Subsequence）算出， LCS 以外の要素の正規化を行う．

#### 3.2.1 攻撃コードの形態素解析

形態素解析とは文を形態素（言語で意味を持つ最小単位）の列に分割しそれぞれの形態素の品詞等を判別する作業であり，自然言語処理で主要分野の一つであり，様々な手法が提案されている．形態素解析エンジンは，英語では POS Tagger[4]，日本語では MeCab[5]などが代表的である．

ここでは攻撃コードを要素単位に分割するために攻撃コードの形態素解析を行う．例えば Bash の脆弱性（CVE-2014-6271）では以下のような攻撃コードがある．

User-Agent:() { v_root; }; bash -i >& /dev/tcp/･･･

これに対して POS Tagger を用いると以下のように分かち書きされる．

User-Agent|:|()| |{| |v|_|root|;| }|;| |bash| |-|i| |>|&| |/|dev|/|tcp|/|･･･

しかし，シグネチャ作成の前処理としては，以下のように分かち書きされることが望ましい．

User-Agent|:|()| |{| |v_root|;| }|;| |bash| |-i| |>|&| |/dev/tcp/･･･

攻撃コードは自然言語ではないため，自然言語処理用の形態素解析器をそのまま適用しても望ましい結果が得られない．そこで攻撃コード専用の形態素解析器を作成することとした．形態素解析は HMM や CRF などの学習器を使用すると精度が良いことが古くから知られている[6,7]．しかし，攻撃コードにおける文字列は対象とする脆弱性によって意味合いが変わる．そのため機械学習をさせる場合，脆弱性毎に学習器，教師データを選定することが必要となる．脆弱性が発表されたばかりの時点では攻撃コードの種類も 1～2 個程度のため，その時点である脆弱性に特化した形態素解析器を作成することは困難である．そこで，様々な脆弱性をつく攻撃コードに共通した特徴（英数字アンダースコアはひとまとまりの単語を構成する，記号やスペースは一文字で意味を持つ等）を元に正規表現辞書を作成し，攻撃コードを分かち書きする形態素解析器を作成した．

#### 3.2.2 複数攻撃コードの LCS 算出

3.2.1 において形態素に分かち書きした攻撃コードの LCS を算出する．LCS とは 2 つの系列の共通部分列の内，最長の系列のことである．例えば<B, D, C, B, A>と<A, B, C, B, D, A>の LCS は<B, C, B, A>となる．本研究では 2 つの系列の長さが M と N の時，時間計算量 O(MN)で LCS を算出するアルゴリズム[8]を用いた．分かち書きした攻撃コードの形態素を 1 要素単位とした複数の攻撃コード系列の LCS を算出することで exploit に特徴的なパターンを抽出することができる．例えば Struts の脆弱性（CVE-2014-0094）をついた以下の 2 つの攻撃コード系列

<“obj”，“.”，“class”，“.”，“ClassLoader”，･･･>
<“&”，“Class”，“[”，“”“，“classLoader”，･･･>

の LCS は次のように抽出される．

<“[cC]lass”，“[cC]lassLoader”，･･･>

英数字アンダースコアで構成される単語において大文字と小文字は同じ要素とみなしている．

#### 3.2.3 LCS 以外の要素の正規化

3.2.2 において LCS によって攻撃コードの exploit に特徴的なパターンを抽出した．LCS 以外の要素は，payload やそれ以外の部分であるため，可変な部分である可能性が高い．つまり可変な部分が変化した攻撃コードの亜種が生まれる可能性が高い．そこで攻撃コードの亜種を検知可能にするため LCS 以外の要素を正規化した．正規化を行うと検知可能な攻撃コードが増えるが同時に正常通信を過検知する可能性が高くなる．そのため適切な手法で正規化を行う必要がある．そこで正規化の手法を段階的に変更できるようにするため独自に正規化レベルを定義した（表 1）．“.”は全ての文字を，“¥w”は英数字アンダースコアを，“¥W”は英数字アンダースコア以外を，“¥s”は空白文字を表すメタ文字で，“*”は直前の文字の 0 回以上，“+”は 1 回以上の繰り返しを意味する．次に，LCS 以外の要素の正規化レベルを決定する．まず，最も正規化レベルの高いレベル 5 で正規化を行い正常通信データセット[9]を用いて過検知の評価を行う．次に過検知があった場合正規化レベルを 1 段階下げて正規化を行う．これらを繰り返すことで過検知がゼロになった時点の正規化レベルを適用する．

表 1　正規化レベル

| 正規化レベル | 正規化ルール | |
|---|---|---|
| | 正規化したい文字 | 正規化 |
| 5 | 全ての文字列 | .* |
| 4 | 全ての文字列 | .+ |
| 3 | 英数字と_のみ | ¥w* |
| | 英数字と_以外の文字のみ | ¥W* |
| | 空白文字のみ | ¥s* |
| | 上記以外 | .* |
| 2 | 英数字と_のみ | ¥w+ |
| | 英数字と_以外の文字のみ | ¥W+ |
| | 空白文字のみ | ¥s+ |
| | 上記以外 | .+ |
| 1 | 全ての文字列 | A\|B |

## 4. 攻撃コードの自動収集器

多くの記事群から特定の脆弱性の攻撃コードが書かれた記事のみを自動抽出し，抽出した記事から攻撃コード部分を自動抽出する手法を提案する．

### 4.1 攻撃コードが書かれた記事の自動抽出

大量の記事の中からあるカテゴリに属する記事のみを収集する技術はテキスト分類の研究分野に属する．現在までに多数のテキスト分類手法が提案されており，その有効性が示されている[10]．本研究では，その中でも分類精度が高いニューラルネットワークを用いた手法により，記事を脆弱性に関する記事のうち攻撃コードが書かれたもの，脆弱性に関する記事のうち攻撃コードが書かれていないもの，脆弱性に関連しない記事の 3 クラスに分類する．本研究では入力層，中間層，出力層の 3 層からなるニューラルネットワークを使用し学習方法には誤差逆伝播法[11]を用いた．学習させるためには文書からベクトルを作成する必要がある．今回は文書を形態素解析し文章の n-gram（ここでは uni-gram，bi-gram）を要素とする 2 値ベクトルと正解の分類結果を教師データとし，3 層のニューラルネットワークを用いて学習をさせた．入力層のユニット数はユニークな uni-gram と bi-gram の数になる．n-gram とは隣接する n 文字を一つの単位として区切る方法である．学習データセットには脆弱性に関する記事として 2014 年の Fulldisclosure の記事 [12], 一般記事として Reuters21578[13] を用いた．

### 4.2 特定の脆弱性に対する攻撃コードの自動抽出

4.1 により対象となる記事群から脆弱性に関する記事のうち攻撃コードの書かれたものを分類することができた．次に分類された記事から特定の脆弱性に関する記事を抽出し，抽出された記事から攻撃コード部分を抽出する．例えば同じ製品名と同じ CVE-ID が存在するという条件で特定

の脆弱性に関する記事を抽出することが考えられる．しかし，攻撃コードが公開された時点ではまだ CVE-ID が割り振られていない段階のものやそもそも記事文に CVE-ID が書かれていないことは多い．記事中に存在する攻撃コードには以下の特徴がある．

- 攻撃コードは記事文の一行以内に収まっていることが多い．
- 攻撃コードの亜種同士を比較すると，特に exploit 部分は非常に類似したパターンとなる．

このような特徴を利用して，特定の脆弱性に関する記事の抽出ルールおよび攻撃コードの抽出アルゴリズムを以下のように設計した．

(1) **CVE-ID 有り**

“同じ製品名＆同じ CVE-ID”が存在した場合，同じ脆弱性の攻撃コードが書かれた記事とする．得られた記事文から既知の攻撃コードとの類似度が最大な行から攻撃コードを抽出する．

(2) **CVE-ID 無し**

“同じ製品名＆既知の攻撃コードとの類似度が閾値を超える行”が存在した場合，同じ脆弱性に対する攻撃コードが書かれた記事とする．攻撃コードの抽出は CVE-ID 有りの時と同様.

文字列の類似度判定には n-gram，Jaccard 係数，LCS を用いたものなどがあげられる．n-gram を用いたものは，n-gram の出現頻度から類似度を算出する．攻撃コードはすぐ隣の文字との関係が重要な場合もあるが，そうでない場合も多くある．次に Jaccard 係数は 2 系列の共通要素の割合を用いて判定するが共通要素の順番は考慮しない．攻撃コードはプログラムであるため要素の順序が重要であることも多い．そのため n-gram や Jaccard 係数による類似度判定を本技術に適用することは困難である．そのためここでは順列を考慮し要素間に何が挿入されてもよい LCS を用いて類似度 S を以下のように定義した．

$$S = \frac{lcsの形態素数}{既知攻撃コードの形態素数}$$

## 5. 実験

本提案手法の有効性の検証および課題の抽出を行うために，以下の 2 つの実験を行った．

(1) **シグネチャ生成器の検知率・過検知率の評価**

過去の 3 つの脆弱性に対するシグネチャを本研究の提案手法で作成し，従来手法で生成されるシグネチャ，ベンダによりリリースされたシグネチャと検知率・過検知率の比較を行った．

(2) **攻撃コード収集器の精度の評価**

特定の脆弱性に対する攻撃コードを含む記事，それ以外の記事のデータセットから交差検定を行い攻撃コード収集器の精度を評価した．

### 5.1 シグネチャ生成器の検知率・過検知率の評価

#### 5.1.1 実験方法

本実験では，Struts の脆弱性（CVE-2014-0094），Bash の脆弱性（CVE-2014-6271），HTTP.sys の脆弱性（CVE2015-1635）に対するシグネチャを用意して評価を行った．比較対象は 3.1 で示した従来手法で生成したシグネチャ，オープンソースの IPS である Snort 用に Talos がリリースしたシグネチャ，オープンソースの WAF である ModSecurity 用に Trustwave がリリースしたシグネチャ，某ベンダが正式版リリース前に早期に顧客に提供したシグネチャとした．

まず，ある日付時点で既知であった攻撃コードを用いてシグネチャを作成した．次に同じ脆弱性をつく現時点で既知の攻撃コード全パターンのうちどれだけを検知するかを検知率とし，正常通信データセット[9]をどれだけ過検知するかを過検知率とした．なお評価用のデータセットは 3.2.3 でシグネチャ作成に使用したデータセットとは別の期間のデータセットを使用している．

#### 5.1.2 実験結果

図 1 に CVE-2014-0094 のシグネチャ評価の結果を示す．本研究は従来手法で生成されるシグネチャ，ベンダがリリースするシグネチャよりも早い時点で高い検知率のシグネチャを作成できることが分かる．また今回プロットした時点の全てのシグネチャにおいて過検知はなかった．他のサンプルにおいても全て以下の特徴が言えた．

- どの日付時点においても本研究は従来手法，ベンダ以上の検知率のシグネチャを生成できる．
- プロットした時点のシグネチャでは全て過検知はなかった．
- 本原稿執筆時点で最新版のシグネチャは本研究，従来手法，ベンダ全て検知率 1 である．

#### 5.1.3 分析と考察

5.1.2 の結果より，本手法，従来手法，セキュリティベンダにおいて同じ条件下（ある時点において同じ攻撃コードが既知である）で生成されるシグネチャは，本手法が最も多くの同じ脆弱性をつく別の攻撃に対して有効であることが示された．すなわち，本手法は従来手法や比較対象としたセキュリティベンダと比較して，同じ検知率のシグネチャを出力するために必要な既知の攻撃コードが少なくてよいことを示している．今回過検知評価に使用したデータセットに対しては過検知が無かったが，このデータセットは 1999 年のものである．そのため，過検知評価としては十分でないと考えられる．より現状に近い過検知評価用のデータセットを作成し，評価を行うことは今後の課題である．

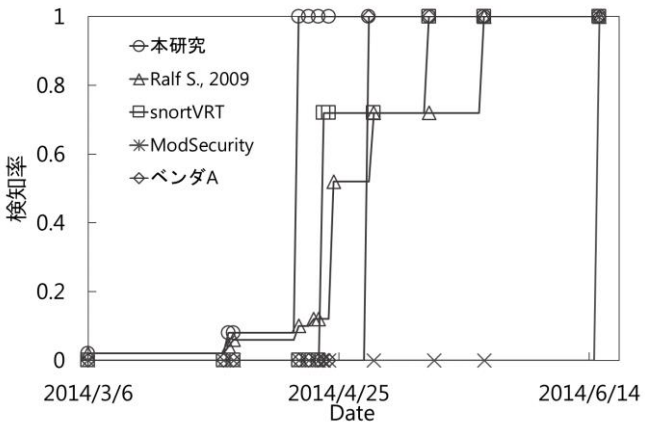


図 1　CVE-2014-0094 のシグネチャ検知率比較

### 5.2 攻撃コード収集器の精度の評価

#### 5.2.1 実験方法

本実験では，データセットとして脆弱性に関する記事には 2014 年の Fulldisclosure の記事を，一般記事には Reuters21578 の記事を用いて，Struts の脆弱性（CVE-2014-0094），Bash の脆弱性（CVE-2014-6271）に関するある既知の攻撃コードを入力値として用い，既知の他の攻撃コードの抽出精度を交差検定により評価した．既知の攻撃コードのパターン全てを入力値として実験し，平均値を評価した．

#### 5.2.2 実験結果

本収集器において類似度 S の閾値をパラメータとした Precision-Recall curve を示す（図 2）．類似度 S の閾値は 0.1 から 1 まで 0.1 刻みに変化させ 10 点プロットした. Precision とは正と予想したデータのうち実際に正であるものの割合，Recall とは実際に正であるもののうち正と予想したものの割合である．収集器の F 値が最大になる類似度 S の閾値は 0.8 となった．F 値とは Precision と Recall の調和平均である．

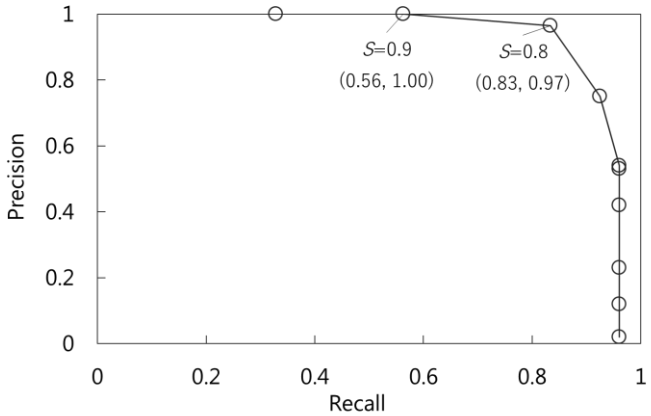


図 2　攻撃コード収集器の Precision-Recall curve

## 6. まとめと今後の予定

### 6.1 まとめ

本研究では攻撃コードの収集とシグネチャの生成を自動化する手法を検討した．

**(1) 攻撃コードの自動収集器**

様々なカテゴリの記事から攻撃コードの書かれた記事を自動収集し，出された記事から特定の脆弱性をつく攻撃コードを自動的に抽出する手法を検討した．ニューラルネットワークと LCS を応用した類似度関数を用いた手法を提案し，プロトタイプを実装した．この技術を用いることでインターネット上から特定の脆弱性をつく攻撃コードを効率的に抽出することができる．ただし実運用を前提とした場合には Precision および Recall をさらに向上させる必要がある．

**(2) シグネチャの自動生成器**

収集された攻撃コードをもとに同じ脆弱性をつく別の攻撃コードも検知可能なシグネチャを自動生成する手法を検討した. 形態素分解及び LCS 抽出を用いた手法を提案し，プロトタイプを実装した．本提案手法により，従来手法で生成するシグネチャやセキュリティベンダのリリースするシグネチャと比較し，検知率の高いシグネチャを生成することができた．

攻撃コードの自動収集器で自動的に収集した攻撃コードをシグネチャの自動生成器に入力することでシグネチャの自動生成システムとなる．本研究で提案したこれらの手法は例えば，脆弱性情報を監視し，シグネチャ作成対応をしているオペレーションの現場に導入することが考えられる．任意コード実行につながる脆弱性が公開される等，緊急性の高い状況下において本技術は有用であると考えられる．

### 6.2 今後の予定

**(1) 攻撃コードの自動収集器**

今後は様々なドメイン配下の記事を用いたデータセットの追加および Precision，Recall の更なる向上を図る．まず, 攻撃コードが書かれた記事は製品名などの固有表現や，プログラムコード断片などが多く存在する．これらに対して CRF を用いた固有表現抽出や word2vec[14]等を用いて，文書ベクトルのノイズを削減することで，4.1 で述べた記事の抽出アルゴリズムを改良することが考えられる．次に 4.2 で述べた類似度は LCS 以外の要素に関する情報を考慮していない．これに対して形態素解析で付与されたラベルの出現順序等を考慮する等して，アルゴリズムを改良することが考えられる．

### (2) 攻撃コードの自動収集器

今後はサンプル数を増やし，そこで得られた知見からシグネチャ生成器のアルゴリズムを改良する．更なる検知率向上・過検知率低下の観点から以下のような改良を検討する．

現在，LCS において同じとみなす系列を判定する際に完全一致（大文字と小文字は区別しない）を判定基準としているが，重み付きレーベンシュタイン距離[15]などの類似度を判定基準とすることで exploit 部分の抽出精度を高める．今回使用した 1999 年の DARPA の正常通信データセットは現在でも WAF や IPS の評価に広く用いられている．しかしこのデータセットは近年のサーバに対する正常通信を疑似するという点では課題が残る．今後は実運用されているサーバのログを活用する等して，より現状に即したデータセットを作成することを検討する．

**謝辞**　本研究を進めるにあたり丁寧なご指導をいただいた NTT セキュリティ株式会社の岸川雄輝様, 東日本電信電話株式会社の折目吉範様には大変感謝しております．また本研究を進める上で攻撃コードやシグネチャに関する貴重な情報を提供していただいた NTT-CERT の橋本淳様，鈴木崇文様をはじめとする皆様には非常にお世話になりました。深く感謝いたします。

## 参考文献

[1] “The Ten Most Critical Web Application Security Risks, OWASP Top 10-2013” https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, (参照 2016-08-01)

[2] “ Imperva Cyber Security Blog” https://www.incapsula.com/blog/shellshock-bash-vulnerability-aftermath.html, (参照 2016-08-01)

[3] Ralf Spenneberg., Webserver-Sicherheit mit mod_security, Datenschutz und Datensicherhelt, May, 2009

[4] “The Stanford Natural Language Processing Group , Stanford Log-linear Part-Of-Speech Tagger” http://nlp.stanford.edu/software/tagger.shtml, (参照 2016-08-01)

[5] “MeCab: Yet Another Part-of-Speech and Morphological Analyzer”, http://taku910.github.io/mecab/, (参照 2016-08-01)

[6] C. D. Manning and H. Schutze. Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, MA, 1999

[7] E. F. Tjong and K. Sang. Text chunking by system combination. In Proceedings of Conference on Computational Natural Language Learning, 2000.

[8] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. J.ACM, Vol. 21, pp. 168–173, January 1974

[9] Lincol Laboratory Massachusetts Institute of Technology, “1999 DARPA Intrusion Detection Evaluation Data Set”

[10] Charu C. Aggarwal , ChengXiang Zhai. A Survey of Text Classification Algorithms. Mining Text Data, pp163-222, January 2012

[11] Rumelhart, David E. Hinton, Geoffrey E., Williams, Ronald J. Learning representations by back-propagating errors. Nature 323 (6088): 533–536., October 1986

[12] “Full Disclosure Mailing List.” http://seclists.org/fulldisclosure/, （参照 2016-08-01）

[13] “Reuters-21578.” http://www.daviddlewis.com/resources/testcollections/reuters21578/, （参照 2016-08-01）

[14] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR, 2013

[15] Marzal, A. & Vidal, E. Computation of Normalized Edit Distance and Applications. IEEE Transactions on Pattern Analysis and Machine Intelligence 15, 926-932, 1993