情報処理学会論文誌　コンピューティングシステム　Vol.7　No.1　1–13 (Mar. 2014)

サポートベクタマシンを用いた WAFへの異常検知機能の実装と評価

伊波 靖[1,a)]　高良 富夫[2]

受付日 2013年7月25日, 採録日 2014年1月6日

概要：インターネットの急速な発展にともない様々な分野においてWebアプリケーションの利用が増える一方で，Webアプリケーションを対象としたCross-Site Scripting攻撃やSQLインジェクション攻撃による情報漏えいなどの深刻な被害が報告されている．Webアプリケーションをそれらの攻撃から守る方法の1つに，WAFの使用があるが，WAFは，ルールの増大による処理能力の低下と未知の攻撃に対して対応が難しいという問題をかかえている．そこで本論文では，WAFの入力値検査にサポートベクタマシン（SVM）を利用することにより，正常なデータを誤検知する割合であるFalse Positiveを低減させながらもXSSおよびSQLインジェクションの特徴を持つ未知の攻撃を検知できる手法をWebサーバとして最も普及しているApacheのモジュールとして実装する．性能を評価するためにApacheの標準的なWAFであるModSecurityとの比較実験を行い，その結果から有効性を議論する．比較実験の結果から，ModSecurityを上回る認識性能と処理性能で検知を行えることが明らかになり，ModSecurityの代替モジュールとしての可能性を示した．

キーワード：Web Application Firewall（WAF），異常検知，Support Vector Machine（SVM），N-gram

Implementation and Evaluation of Anomaly Detection Using Support Vector Machine for WAF

Yasushi Iha[1,a)]　Tomio Takara[2]

Received: July 25, 2013, Accepted: January 6, 2014

Abstract: Coupled with the rapid development of the internet, use of Web applications has increased in various fields. Meanwhile, serious damage such as XSS attacks and SQL injection attacks targeted at Web applications has been reported. One way to protect Web applications from such attacks is to use WAF. But the problem with WAF is that the processing power is reduced because of the increase in rules and that it is difficult to deal with unknown attacks. Therefore, in this article, by using SVM for inspection of input value by WAF, we propose the method by which False Positive, the rate of misdetection of normal data, is declined and unknown attacks can be detected at the same time. And then we show its effectiveness by an experiment. Besides, the method proposed by us is implemented as a module of Apache. And then, to evaluate its performance, we conduct a comparative experiment with ModSecurity, which is the standard WAF of Apache. And from its result we discuss its effectiveness. It has been found from the result of the comparative experiment that it can carry out detection with recognition performance and processing performance exceeding those of ModSecurity.

Keywords: Web Application Firewall (WAF), anomaly detection, Support Vector Machine (SVM), N-gram

[1] 沖縄工業高等専門学校メディア情報工学科
Department of Media Information Engineering, Okinawa National College of Technology, Nago, Okinawa 905–2192, Japan
[2] 琉球大学工学部情報工学科
Department of Information Engineering, University of the Ryukyus, Nishihara, Okinawa 903–0213, Japan
[a)] yasuc@okinawa-ct.ac.jp

1. はじめに

近年，インターネットの急速な発展にともないWWW（World Wide Web）が普及したことにより，企業や官公庁などにおいて多くのWebアプリケーションが利用されるようになった．しかし，利用が増える一方で，Webアプリ

ケーションを狙った XSS（Cross-Site Scripting）攻撃 [1] や SQL インジェクション攻撃 [2], [3], [4] も後を絶たず，Web サイトの改ざんや非公開情報の漏えいなどの深刻な被害が報告されている [5]．Web アプリケーションを XSS 攻撃や SQL インジェクション攻撃から防御する方法の 1 つとして，WAF（Web Application Firewall）の使用がある [6]．WAF とは，Web アプリケーションを含む Web サイトと利用者の間で交わされる HTTP 通信（WAF によっては HTTPS 通信を含む）を検査し，攻撃などの不正な通信を自動的に遮断するソフトウェア，もしくはハードウェアである．WAF は，入力値をホワイトリストおよびブラックリストとのシグネチャマッチングで識別し攻撃を遮断する．しかし，正常なリクエストを不正なリクエストとして誤検知する False Positive の問題や未知の攻撃に対して検知漏れを起こす可能性がある．

この問題を解決するための 1 つの方法が，機械学習によるアノマリ検知である．

Bolzoni らは，N-gram 法と SVM を組み合わせた WAF として Panacea を提案し，機械学習アルゴリズム（RIPPER）に比較して SVM を用いた場合の有効性について検証を行っている [7]．さらに，SQL インジェクションの検知に SVM を用いる方法がいくつか提案されている [8], [9], [10], [11]．Komiya らは，XSS と SQL インジェクションの検知に SVM を用いる方法を提案し，評価実験により有効性を示した [12]．

これらの既存研究から N-gram 法と SVM を用いた検知手法の有効性は示されてきたが，Web サーバとしての利用実績が高い Apache Web サーバのモジュールとして実装された例はこれまで存在しなかった．Apache Web サーバの WAF モジュールとしては，ModSecurity [13] が広く利用されているが，ModSecurity は大量のシグネチャパターンと正規表現による検知手法により Web サーバのパフォーマンスに影響を与えることが知られている [14]．また，False Positive の発生を抑えるためにルールセットを適切に設定することが難しく，運用上の問題となっている．

そこで，本論文では，ModSecurity の Web サーバのパフォーマンスに与える影響を実際の攻撃パターンを含む大量のデータセットを用いた評価実験を通して確認するとともに，ModSecurity の代替として Apache のモジュールとして使用できる XSS および SQL インジェクション攻撃用の WAF の実装を目的とする．実装には，既存研究において有効性が確認されている Web サイトへのリクエスト中のクエリ文字列を文字に着目して N-gram 法により特徴ベクトルを生成し，生成した特徴ベクトルを SVM で認識させる手法を用いた．実装したモジュールを用いて ModSecurity との性能比較実験を行い，検知率や誤検知率および処理時間を測定した [15], [16]．

本論文で行った ModSecurity の性能評価実験は，筆者が調査した範囲ではこれまで行われておらず，また，Apache のモジュールとして N-gram 法と SVM を組み合わせた検知手法を実装した例も存在しない．本論文での比較実験の結果において，False Positive を低減させながらも ModSecurity を上回る認識性能と処理性能で検知を行えることを明らかにし，XSS および SQL インジェクション対策用 WAF として ModSecurity の代替モジュールとして使用できる可能性を示した．

本論文の構成は以下のとおりである．2 章で関連技術として Web アプリケーションの脆弱性および WAF の解説を行い，検知手法として使用する SVM の概要を解説する．3 章で SVM を WAF の異常検知に利用する手法を解説し，予備実験について解説する．また，検知手法を Apache のモジュールとして実装する方法について解説する．4 章では実装したモジュールの性能評価実験について解説し，5 章では ModSecurity と本研究での実装における処理性能に関する考察および評価実験の結果から類似研究との考察を行う．最後に 6 章でまとめを行う．

2. 関連技術

本論文で対象とする Web アプリケーションの脆弱性について Web アプリケーションの概要と，脆弱性としての XSS および SQL インジェクションについて解説する．また，検知手法で使用する SVM の概要について解説する．

2.1 Web アプリケーション

Web アプリケーションとは，Web アプリケーションサーバで処理を実行し，Web ブラウザで処理結果を表示するクライアントサーバ型のシステムである．また，Web サーバと Web ブラウザとの間に HTTP（Hyper Text Transfer Protocol）通信を使って，サーバとクライアント間のデータ送受信を行っている．**図 1** に Web アプリケーションの基本的な仕組みを示す．

Web アプリケーションの特徴は，Web サーバと Web ブラウザとの間で HTTP 通信プロトコルを使用していることである．ステートレスなプロトコルである HTTP は，HTTP のみでセッション管理を行うことができない．多くの Web アプリケーションは，Cookie などを用いてサーバ

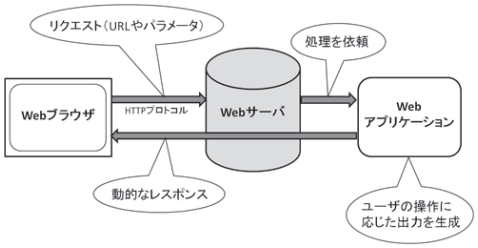

図 1　Web アプリケーションの概要
Fig. 1　Overview of Web application.

とクライアント間でセッション ID の受け渡しを行い，セッションの管理を行っている．Web アプリケーションは，インターネットを介して不特定多数の人間がアクセスできる特徴ゆえに，XSS や SQL インジェクションによる攻撃にさらされるため，それらの脆弱性を狙った攻撃から守る必要がある．

2.2 XSS（Cross-Site Scripting）

Web アプリケーションにおいては，利用者が Web ブラウザから入力した内容を処理し，その出力結果を Web ページとして出力することが一般的である．たとえば，掲示板における投稿内容や検索エンジンにおける検索キーワードの表示などである．しかし，利用者が入力したデータの中には，いわゆる「汚染されたデータ」と呼ばれるデータが存在し，Web アプリケーションにおいて適切な処理を行わずに出力を行った場合，その Web ページに悪意を持った JavaScript のコードなどの攻撃用スクリプトを埋め込まれてしまう可能性がある．この問題を「XSS（Cross-Site Scripting）の脆弱性」と呼び，この問題を悪用した攻撃手法を「XSS 攻撃」と呼ぶ．

XSS 攻撃により，発生しうる脅威には，本物サイト上に偽のページが表示される，ブラウザが保存している Cookie を取得される，任意の Cookie をブラウザに保存させられるなどがある．

2.2.1 XSS 攻撃

XSS の脆弱性を使った攻撃は，HTML コンテンツ内にスクリプトを埋め込む操作によって行われる．これは，プログラマが用意したサーバ側のプログラムが，ブラウザから入力された値をそのままブラウザへの出力に使用してしまうことを悪用するためである．XSS 攻撃の手口と攻撃文字列には多くのバリエーションがありうる．以下に例を示す．

(1) テキスト部分に直接タグを挿入

```
<script>document.cookie='sid=ROOT'</script>
```

(2) 引用符と > を用いタグ属性値から脱出して <script> タグを挿入

```
'''><script>document.cookie='sid=ROOT'</script>
```

(3) コメント終了記号 −− > を用い HTML コメントから脱出して <script> タグを挿入

```
−−><script>document.cookie='sid=ROOT'</script>
```

2.2.2 XSS 対策

XSS 対策の基本は，攻撃者によって送り込まれた文字列が出力において HTML コンテンツの一部となったときに，ブラウザによってスクリプトとして解釈されないように処理を行うことである．たとえば，HTML コンテンツに出力する際に，特殊文字が HTML のタグとして解釈されないように実体参照を用いて置換を行う．あるいは，入出力データ内に HTML データを扱わずに済む場合は，出力の際にすべてのタグを排除する．しかし，XSS の脆弱性を Web アプリケーションから根絶することは難しい．なぜならばその Web アプリケーションプログラムが，入力データをどのような形のデータとして扱うかによって様々な対策方法が考えられるうえ，多くの入力データを扱う中で，容易に対策漏れが起こるからである．

2.3 SQL インジェクション

データベースと連携した Web アプリケーションでは，利用者から入力された情報に基づいてデータベースへアクセスするための SQL 命令文を生成している．ここで生成する命令文に汚染されたデータが含まれることにより問題がある場合，データベースを不正に利用される可能性がある．このような問題を「SQL インジェクションの脆弱性」と呼び，問題を悪用した攻撃を「SQL インジェクション攻撃」と呼ぶ．

SQL インジェクション攻撃により発生しうる脅威には，データベースに蓄積された非公開情報の閲覧，データベースに蓄積された情報の改ざん，消去，認証回避による不正ログイン，ストアドプロシージャなどを利用した OS コマンドの実行などがある．

2.3.1 SQL インジェクション攻撃

たとえば，利用者が Web アプリケーションにログインする際，Web アプリケーションでは，データベースに登録された利用者情報に基づいて認証を行う．その際，以下のような SQL 文が使用される．

```
SELECT uid FROM account_table WHERE uid='ユーザ ID' AND pw='パスワード'
```

この SQL 文は，利用者が入力したユーザ ID とパスワードを利用してデータベースに対して問合せを行っている．もし，該当するレコードがあった場合は，そのユーザの ID を返し，これ以降の処理において ID が返された場合にログインを許可する．

ここで，ユーザ ID に「' OR 1=1−−」という文字列が与えられた場合には，次のような SQL 文が組み立てられてしまう．

```
SELECT uid FROM account_table WHERE uid='' OR 1=1−−' AND pw='任意の文字列'
```

ここで与えられた文字列「' OR 1=1−−」には，次のような意味がある．

' ：1 つ前の「'」と対となり，文字列定数を終わらせる．
OR 1=1：uid の値に関係なく，検索条件を，真とさせる．
−− ：それ以降の内容をコメントとして無視させる．

このため，この文字列をパラメータとして与えられた場合は，ユーザ ID がつねに返ってくるため，本来ログインを許可されていないユーザもログインが可能となってしまう．

2.3.2　SQL インジェクション対策

SQL インジェクションの対策としては，データベースにアクセスする SQL 文にユーザが入力したデータを使用する場合，文字列データであれば SQL エスケープを行い，数値データであれば数値として適切であるかどうかを確認した後で SQL 文を生成する方法もある．また，あらかじめ SQL 文をコンパイルしてテンプレートとして使うプリペアドステートメントの使用や文脈に応じた特殊記号対策などを行う方法もある．SQL インジェクションも XSS と同様に対策が難しい．

2.4　WAF（Web Application Firewall）

Web アプリケーションから XSS や SQL インジェクションの脆弱性を根絶することは難しい．よって，Web ブラウザと Web サーバとの間にあって双方でやりとりされるデータをチェックし，その脆弱性を Web アプリケーションの外側から防いでくれる機能が必要となる．Web アプリケーションの XSS や SQL インジェクションの脆弱性に対する攻撃を防ぐものとして，WAF（Web Application Firewall）と呼ばれるものがある．

2.4.1　WAF の機能

WAF は以下の 3 つの機能によって，外部からの攻撃を防ぐ．

(1) 入力値検査
　パラメータに対するホワイトリストあるいはブラックリストによる検査を行う．

(2) 画面遷移のチェック
　外部からの入り口となるページのチェック，Web アプリケーション内のすべての遷移のチェックを行う．

(3) hidden フィールド操作の防止
　hidden フィールドや Cookie，ラジオボタンなどの選択肢の値がクライアント側で改変されてないかチェックし，改変されていたらエラーとする．

WAF の基本的な機能は，ホワイトリストおよびブラックリストによる入力値の検査である．通常，入力値検査は正規表現などの手法を用いてパターンマッチングにより行われ，検知した攻撃を遮断する．しかし，すべての想定される入力値に対してホワイトリストを定義，設定することは難しく，正常なリクエストを不正なリクエストとして誤検知する False Positive の問題が発生する．またブラックリストを用いた場合，すべての攻撃をブラックリストとして設定できたとしても，未知の攻撃に対しては検知漏れを起こす可能性がある．さらに，増加する攻撃手法に対してブラックリストが増大しパターンマッチングにかかる処理時間が Web アプリケーションのパフォーマンスに影響を与えるようになってきており，新たな検知手法の研究がさかんに行われている．

なお，本論文では WAF の機能の中でも，XSS や SQL インジェクションの脆弱性など，インジェクション系の脆弱性対策の基本となる入力値検査の問題について検討する．

2.4.2　ModSecurity

ModSecurity は Apache のモジュールとしてオープンソースで開発されている WAF である．ModSecurity は Web サーバと Web ブラウザの間のあらゆるリクエストおよびレスポンスに対してフィルタリングを行うことができる．また，hidden，Cookie などのパラメータや値に対してフィルタリングを行うことができる．さらに，詳細な監査ログを取得することも可能である．

2.5　サポートベクタマシン（SVM）

SVM は，Vapnik によって提案された，統計的学習理論に基づく新しい 2 クラスのパターン認識手法である．ニューラルネットワークなどの従来法と比較して汎化能力が高い点と最適解が求まる点に特徴があり，学習に用いていないデータに対しても高い認識率を示す．SVM が優れている理由に，クラス分類を行う識別面を一意に決定するために「マージン最大化」という明確な基準が設けられている点と，線形分離することが不適切な場合「カーネル法」により非線形の判別問題への拡張ができることがあげられる．また，SVM は，その学習に認識誤りと汎化性能の両面から最適化が行われ，これが 2 次の凸計画問題として定式化されているため最適解を求めることができる [17].

SVM の概要を以下に示す．与えられた学習用データ集合 S を，

$$(\mathrm{x}_1, y_1), (\mathrm{x}_2, y_2), \cdots, (\mathrm{x}_n, y_n) \quad (1)$$
$$\forall i, \quad \mathrm{x}_i \in \mathbb{R}^N, y_i \in \{-1, 1\}$$

とする．ここで x_i はデータ i の特徴ベクトルであり，y_i はデータ i のクラスラベルで，+1（正例）か −1（負例）を表す．

SVM は図 **2** に示すように，データ集合 S を正しく分離するために式 (2) で与えられる超平面のうち，マージン

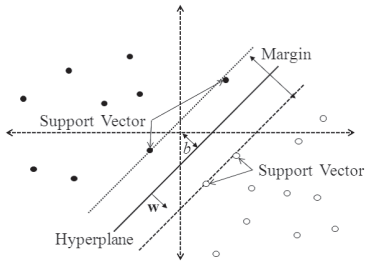

図 2　線形 SVM
Fig. 2　Linear SVM.

（分離超平面とベクトルの距離）が最大になるような分離超平面が最も汎化能力が高いものと判断する．

$$f(x) = \mathrm{w} \cdot \mathrm{x} + b \qquad (2)$$
$$\mathrm{w} \in \mathbb{R}^N,\ b \in \mathbb{R}$$

ここで，w は n 次元の法線ベクトル，$(\mathrm{w} \cdot \mathrm{x})$ は w と x の内積を表す．

SVM では，学習事例が線形分離不可能な場合には，カーネル法を組み合わせることでアルゴリズムを容易に非線形に拡張することができる．よく使用されるカーネル関数としては，線形カーネル $(\mathrm{x}_i \cdot \mathrm{x}_i)$，多項式カーネル $(\mathrm{x}_i \cdot \mathrm{x}_j + 1)^d$，RBF（Radial Basis Function）カーネル $\exp\{-|\mathrm{x}_i - \mathrm{x}_j|^2/2\sigma^2\}$ などがある．

SVM は，カーネル関数を用いることにより，素性ベクトルを高次元の素性空間に写像し，素性空間において線形分離を行う．この写像によって SVM は線形分類器でありながら，非線形分離が可能となっている．

3. SVM を用いた WAF への異常検知機能

前述したとおり WAF には「未知の攻撃パターンへの対応」と「ブラックリストの増大による処理時間の増加」の 2 つの問題をかかえている．この問題を解決するための 1 つの方法が，機械学習によるアノマリ検知である．本論文では，Web アプリケーションへのリクエスト中のクエリ文字列から N-gram を用いて特徴ベクトルを生成し，機械学習として SVM を用いてアノマリ検知を行う手法を採用する．

3.1 検知手法の概要

検知手法の概要を**図 3** に示す．あらかじめ用意した正常なリクエスト（ホワイトリスト）と XSS および SQL インジェクション攻撃を含むリクエスト（ブラックリスト）を N-gram を用いて特徴ベクトルに変換し，2 クラスのパターン識別器を構成するために SVM を用いて学習を行いモデルを生成しておく．次に検知対象リクエストからクエリ文字列を取得し，N-gram を用いて特徴ベクトルに変換し，学習により生成したモデルを用いて SVM で識別を行う．SVM は汎化性能が高く，学習に用いたデータ以外についても高い認識性能を発揮するため，ホワイトリストに入力可能なすべてのパラメータを設定することができない場合や設定漏れの可能性があっても誤検知を低減できることが期待できる．また，ブラックリストとのシグネチャマッチングだけでは検知漏れの可能性があった未知の攻撃についても，検知率の向上が期待できる．

3.2 N-gram による特徴ベクトルの生成

SVM により識別を行うためには，ホワイトリストとブラックリストが持つ特徴を素性ベクトルとして表現する必要がある．本論文における検知手法では，言語モデルとして広く採用されている N-gram を用いた．ホワイトリストとブラックリストに記述された各行の文字列に対して N-gram モデルを用いて得られた N-gram と N-gram の共起頻度によって特徴ベクトルを生成した．

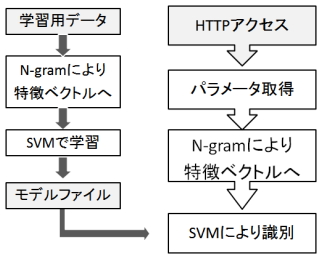

図 3　SVM を用いた WAF の異常検知機能
Fig. 3　Anomaly detection using SVM for WAF.

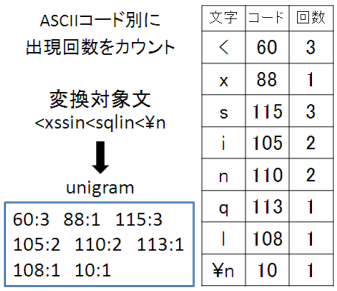

図 4　unigram への変換例
Fig. 4　Examples of converting to unigram.

表 1　XSS および SQL インジェクションデータ
Table 1　Data of XSS and SQL injection.

カテゴリ	入手先
XSS （600 個）	C. Snake: XSS (Cross Site Scripting) Cheat Sheet など http://ha.ckers.org/xss.html
SQL （700 個）	Unixwiz: SQL Injection Attacks by Example など http://unixwiz.net/techtips/sql-injection.html
XSS & SQL （1,100 個）	HTTP DATASET CSIC 2010 http://iec.csic.es/dataset/

入力文字列の文字コードは ASCII と UNICODE を対象としており，日本語の入力文字列については UNICODE に変換を行った．N-gram の生成は入力文字列から文字単位ではなくバイト単位で切り出すことで，可変長の文字コードに対応できるようにした．

具体的には，SVM の「素性番号」として，N-gram を対応させ，「素性の値」に N-gram の共起頻度を対応させる．N-gram では，N が 1 のとき unigram，N が 2 のとき bi-gram，N が 3 のとき trigram と呼ばれる．**図 4** に unigram の場合の特徴ベクトルへの変換例を示す．

3.3 実験用データセット

表 1 にブラックリストに用いるデータを示す．ブラックリストのデータは重複のないデータを総計で 2,400 個を用意した．内訳は XSS が 1,400 個，SQL インジェクションが 1,000 個である．なお，ブラックリストのデータは目視

表 2　正常なデータ
Table 2　Data of normal message.

カテゴリ	入手先
日本語の名前	同姓同名事典，全国同姓同名ランキング（1～200 位）
（200 個）	http://www.douseidoumei.net/00/dou01.html
英語の名前	欧羅巴人名録など
（200 個）	http://www.worldsys.org/europe/search/
日本語の住所	市区町村役場住所一覧
（400 個）	http://www.geocities.jp/takevv/adrs.html
英語の住所	Global NAVITIME，New York 公共機関・施設
（200 個）	http://global.navitime.co.jp/area/us/
日本語と英語の	twitter パブリックタイムライン
メッセージ	http://twitter.jp
（1400 個）	
電話番号	日本のお役所全国の都道府県庁一覧など
（100 個）	http://www.towninf.co.jp/p/52/52100/100.htm
パスワード	The Top 500 Worst Passwords of All Time（1～200 位）
（100 個）	http://www.whatsmypass.com/
	/the-top-500-worst-passwords-of-all-time
その他	HTTP DATASET CSIC 2010
（4,400 個）	http://iec.csic.es/dataset/

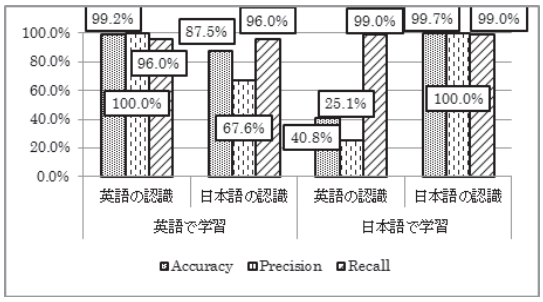

図 5　学習データによる認識結果の違い
Fig. 5　Difference in the recognition result from training data.

表 3　実験の環境
Table 3　Environment of experimental.

OS	Fedora 14
CPU	Intel® Core™ i7 2.80 GHz
RAM	4 GB
Web サーバ	Apache 2.2.17
SVM	Lib SVM 3.11
WAF	ModSecurity 2.5.12

による確認と ModSecurity により攻撃パターンと認識できたものだけを用意した．

なお，ModSecurity による確認を行った理由について述べる．学習データおよび評価データとして用いる場合，それぞれのデータを表す教師情報が必要となる．そのため，ブラックリストのデータに正常なデータが含まれていると教師情報に誤りが生じ，SVM による学習および認識実験を行う際の結果に悪影響を与えるため，ホワイトリストとブラックリストを明確に分ける必要がある．ホワイトリストについては，入手先に表記したように正常なデータであることが明確なものを用いているが，ブラックリストについては目視のみでは十分ではないと考え，ModSecurity により異常と認識することを確認したうえでデータとして用いた．

次に，**表 2** にホワイトリストに用いるデータを示す．ホワイトリストのデータは重複のないデータを総計で 7,000 個を用意した．ホワイトリストに用いるデータを準備するにあたり，Web アプリケーションに入力されると考えられるデータを「名前」，「住所」，「メッセージ」，「電話番号」，「パスワード」の 5 つのカテゴリに分類してデータを準備した．「名前」，「住所」，「メッセージ」については日本語と英語の両方を揃えた．

異常なデータとしては英語のみのデータを用いた．また，正常なデータとして日本語と英語の両方のデータを用いたのは，我々が N-gram と SVM を組み合わせた異常検知手法の有効性について確認した予備実験において，正常なデータとして日本語を認識する際に日本語の学習が効果があることを確認したためである [15]．予備実験で行った実験の結果を**図 5** に示す．図から日本語のデータを用いて学習することで，日本語の認識データの評価率が向上することが分かる．このことから学習データに日本語と英語を用意することで日本語の正常なデータを誤検知する割合を減らせる可能性がある．

なお，侵入検知システムの性能を評価するベンチマークとしては，MIT が作成した DARPA IDS Evaluation Data Sets 1999 や UCI が作成した KDD Cup 1999 Data などが存在するが，WAF の性能を評価するベンチマークには標準的なものが存在しないため，先行研究において使用されていた HTTP DATASET CSIC 2010 [18] を用いた．このデータには，学習用および認識用の正常なアクセスデータと認識用の異常なアクセスデータが用意されおり，ブラックリストとホワイトリストとして用いた．

3.4　予備実験

本論文における検知手法の有効性を確認するために予備実験を行った．**表 3** に実験に用いた環境を示す．

予備実験では表 1 のブラックリストおよび表 2 のホワイトリストのデータから各 200 個を用意し，それぞれ 100 個を学習用，残りの 100 個を評価用とした．各データは N-gram を用いて特徴ベクトルを生成するプログラムにより LibSVM のデータフォーマットに変換した．あらかじめ学習用のデータを LibSVM の svm-train コマンドを用いて学習を行い，識別に用いるモデルを生成し，svm-predict コマンドを用いて評価を行った．N-gram による識別性能の変化を調査するため，N の値を 1～3 まで変化させた．実験結果は，識別の Accuracy（精度）の高さと F 値（$F_{\beta=1}$）で評価する．Accuracy は次式で与えられる．

Accuracy = 正解したデータの総数 ÷ データの総数

また，F 値とは，情報検索の検索性能を評価するための

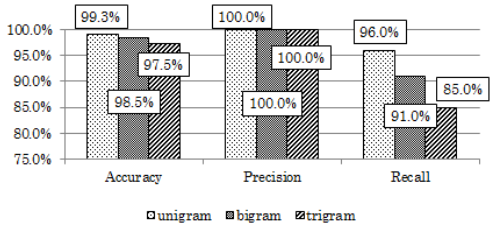

図 6　予備実験の結果
Fig. 6　Results of preliminary experiment.

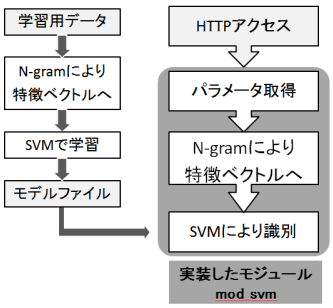

図 7　実装モジュールの概要
Fig. 7　Overview of the implementation module.

値であり，0〜1 の間で求められる．1 に近いほど性能が高いことを意味する．F 値（$F_{\beta=1}$）は，次の式 (3) で求められる．

$$F_{(\beta=1)} = \frac{2 \cdot Recall \cdot Precision}{Recall + Precision} \tag{3}$$

また，式 (3) において，Recall は再現率，Precision は適合率を示し次式で与えられる．

Recall = 正解した正例の数
　　÷ データセット中の正例の数

Precison = 正解した正例の数 ÷ 正例と見なした数

図 6 に予備実験の結果を示す．C-SVM での実験の結果，F 値は，unigram で 0.98，bigram で 0.95，trigram で 0.92 となった．unigram は，Accuracy も 99.3%と最も高い値を出している．また，unigram の False Positive は 0%，False Negative は 4%であった．よって，Accuracy と F 値から SVM の有効性を確認した．

3.5　Apache への実装方法

図 7 に本論文における検知手法を Apache に実装したモジュール mod_SVM の概要を示す．Apache への実装にあたっては，Apache に用意されている hook 関数を用いた．hook 関数を用いることにより HTTP のリクエストを受けてからレスポンスを返すまでの任意のタイミングに，モジュールとして関数による処理を挿入することができる．本研究の実装においては，すべてのリクエストに対してレスポンスの生成前に hook を行う「ap_hook_access_checker」を用いた．この hook 関数を用いることで，すべてのリクエストに対して SVM による識別処理を行うことができる．

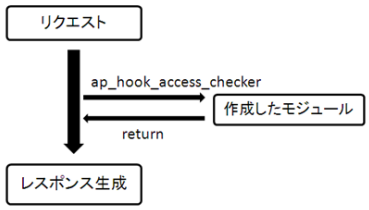

図 8　Apache への hook 関数による実装方法
Fig. 8　Implementation using a hook function to Apache.

図 8 に hook 関数の概要を示す．

実装したモジュールの処理内容について解説する．あらかじめ学習データを用いて SVM が使用するモデルファイルを作成しておく．このモデルファイルは Apache の起動時に読み込まれる．Apache への HTTP アクセスが発生すると，ap_hook_access_checker により Apache からモジュールが呼び出される．モジュールでは HTTP アクセスのパラメータを reqest_rec 構造体より取得する．request_rec 構造体は，リクエストごとに作成されるもので，現在のリクエスト情報とレスポンス情報を保持するためのものである．パラメータの取得方法を以下に示す．

- GET パラメータの場合
  Apache モジュールに引数として渡される request_rec 構造体のメンバである args に一連の文字列として GET パラメータが渡されるため，args をデリミタ ‘&’ でトークンに分割して，各パラメータごとに取得する．
- POST パラメータの場合
  Apache2 系では，POST パラメータは bucket brigade と呼ばれる抽象化されたデータ構造の形で取得することができる．bucket brigade は，入力ストリームを抽象化したデータ構造 bucket がリング状に連なった構造であり，各 bucket がデータを保持している．request_rec の input_filters から取得した bucket brigade の各 bucket からデータを読み込み，EOS bucket が出現するまで繰り返すことで，POST パラメータを取得することができる．取得したパラメータをデリミタ ‘&’ でトークンに分割して，各パラメータごとに取得する．

ここで，取得したすべてのパラメータの値が空であった場合，モジュールはそれ以降の処理を行わずに動作を終了し，Apache へと処理を返す．

次に，取得した各パラメータを SVM で識別するために，モデルファイルを作成した際と同様の処理で N-gram を用いて特徴ベクトルを生成する．生成した特徴ベクトルを，学習によって作成してあったモデルを用いて SVM で識別を行う．識別の結果，不正な値であると判定されたパラメータが存在した場合は，Web ブラウザに対して「403 Forbidden」の応答を返してアクセスを拒否する．すべてのパラメータが正常な値であると判定された場合は，モ

ジュールの動作を終了し Apache へと処理を返す．

実装に使用した SVM は，LibSVM [19] であり，カーネルには線形カーネルを用い，Solver Type は C-SVM を用いた．また，SVM のパラメータについてはデフォルト値を用いた．なお，一般的に RBF カーネルのほうが性能が高いことが知られているが，あらかじめ行った実験において他のカーネルに比べて線形カーネルの性能が高かったため実装には線形カーネルを用いた．SVM において特徴次元数が大きい場合，高次元に写像する必要がないため，線形カーネルのほうが性能が高くなることが知られている．

4. 実装したモジュールの性能評価実験

実装したモジュールの有効性を識別性能と処理性能から検証するために行った実験について解説する．実験には，表 1 のブラックリストのデータ 2,400 個のうち 1,200 個を学習データとして，残りの 1,200 個を評価データとして使用した．また，表 2 のホワイトリストのデータ 7,000 個のうち 3,500 個を学習データとして，残りの 3,500 個を評価データとした．学習および評価データセットについては，それぞれのデータセットがユニークになるように元のデータセットの各データ単位でランダムにシャッフルを行い，できるだけ偏りや恣意的な分け方が発生しないように工夫を行った．

なお，ModSecurity は検知に関する設定は標準のままにし，詳細な監査ログを出力しないように設定して使用した．ModSecurity の具体的な設定は，標準で提供されている XSS および SQL インジェクション攻撃検知用のシグネチャーパターンを中心として設定を行った．

4.1 識別性能

mod_SVM の識別性能を ModSecurity と比較するために，Apache に mod_SVM を設定した状態と ModSecurity を設定した状態でそれぞれ実験を行った．実験は，Perl 言語で作成した HTTP によるアクセスプログラムを用いて学習データおよび評価データをパラメータとするリクエストでアクセスして検知状況を調査した．なお，予備実験で N-gram において N = 1 と N = 2 のときに Accuracy が高かったため，mod_SVM は N = 1 と N = 2 について実験を行った．**表 4** に学習データを用いたときの実験（Closed Test）結果を，**表 5** に評価データを用いたときの実験（Open Test）結果を示す．

実験の結果から，mod_SVM は，N = 2 のときに Closed Test においてホワイトリストおよびブラックリストのデータを 100%の精度で識別していることが分かった．これは，ModSecurity の精度 99.85%を上回っている．また，Open Test においても N = 2 のときに 99.98%の精度で識別しており，F 値も 0.9996 と高い値となっている．これは，ModSecurity の精度 99.81%と F 値 0.9963 を上回っている．N = 2 のときの mod_SVM は，学習と評価データを合わせた 9400 個中，誤検知したのは 1 個だけであった．なお，ブラックリストのデータは，ModSecurity を用いて XSS または SQL インジェクションと識別されたものだけを用いているため，ModSecurity の実験では，ブラックリストについては 100%の精度となったが，ホワイトリストにおいて誤検知が発生していた．また，mod_SVM において，各リクエストにおいて正常なデータと異常なデータを混在した形で実験を行ったが，検知率に違いは見られなかった．これは，リクエスト中の各パラメータについて，個別に識別を行っているためである．

表 4　Closed Test の結果
Table 4　Result of Closed Test.

	ModSecurity	svm（N = 1）	svm（N = 2）
Accuracy	99.85%	99.89%	100.00%
Precision	99.42%	100.00%	100.00%
Recall	100.00%	99.58%	100.00%
F-measure	0.9971	0.9979	1.0000

表 5　Open Test の結果
Table 5　Result of Open Test.

	ModSecurity	svm（N = 1）	svm（N = 2）
Accuracy	99.81%	99.60%	99.98%
Precision	99.26%	99.33%	100.00%
Recall	100.00%	99.08%	99.92%
F-measure	0.9963	0.9921	0.9996

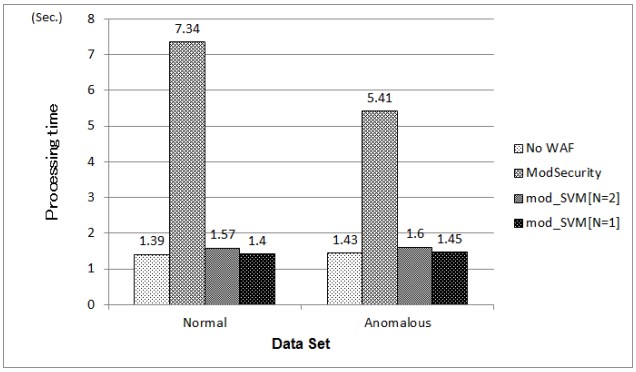

図 9　処理時間計測結果
Fig. 9　Result of the processing time.

4.2 処理性能

mod_SVM の処理性能を ModSecurity と比較するために，Apache に両方のモジュールを組み込まない状態（No WAF）と Apache に mod_SVM を設定した状態と ModSecurity を設定した状態で実験を行った．実験は，Perl 言語で作成した HTTP によるアクセスプログラムを用いてホワイトリストとブラックリストの評価データから 1,000 個ずつをパラメータとするリクエストでアクセスして，OS に付属の time コマンドを用いて 10 回ずつ処理時間を測定し，平均値を求めた．**図 9** に実験の結果を示す．

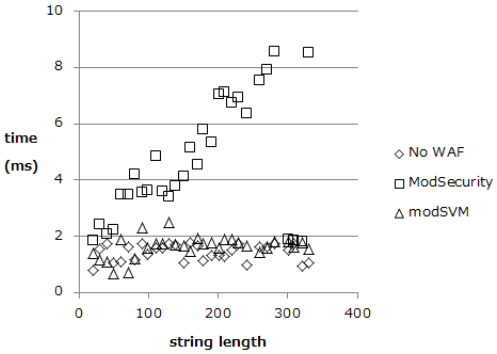

図 10　正常なアクセス 1 件の処理時間
Fig. 10　Processing time of 1 normal access.

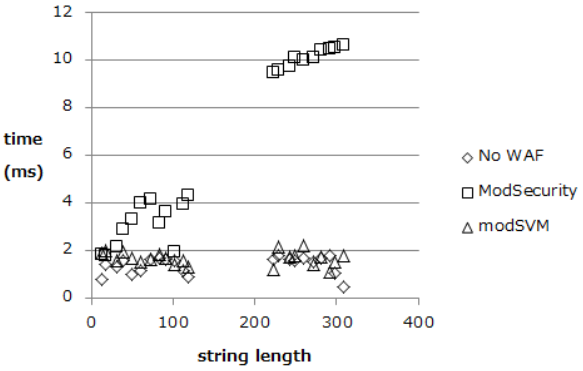

図 11　異常なアクセス 1 件の処理時間
Fig. 11　Processing time of 1 malicious access.

実験の結果から正常なデータを処理する場合において ModSecurity は mod_SVM [N = 2] の 4.67 倍の処理時間がかかっており，XSS や SQL インジェクションなどの異常なデータを処理する場合において 3.38 倍の処理時間がかかっていることが分かった．また，mod_SVM [N = 2] の処理時間は，WAF を使用しない場合に比べて正常なデータを処理する場合において 1.13 倍，異常なデータを処理する場合において 1.12 倍の増加に抑えられていることが分かった．一方，ModSecurity の処理時間は，WAF を使用しない場合に比べて正常なデータを処理する場合において 5.28 倍，異常なデータを処理する場合において 3.78 倍と大幅に増加することが分かった．ModSecurity が正常なデータを処理する場合において処理に時間がかかっているのは，ブラックリストとのマッチングを行う際に正常なデータではすべてのブラックリストとのマッチングが行われるためだと思われる．なお，mod_SVM において N = 2 の場合，N = 1 に比べて処理時間の増加は正常なデータを処理する場合において 1.12 倍，異常なデータを処理する場合において 1.10 倍となっていることが分かった．

次に，リクエスト文字列の長さと処理時間の関係を調査するために，Apache に付属するベンチマークツールである ab（Apache Bench）コマンドを用いて，文字列の長さを変えて処理時間を計測した．文字列の長さの違う 100 個のデータそれぞれについて 1,000 回のアクセスを行い，1 回あたりの処理時間を計測した．**図 10** に正常なアクセス 1 件の処理時間を，**図 11** に異常なアクセス 1 件の処理時間を示す．それぞれのグラフにおいて横軸はアクセスした文字列の長さを表し，縦軸は処理時間を表す．グラフから mod_SVM は，処理時間が文字列の長さにかかわらずほぼ一定となっており，ModSecurity は文字列の長さに比例して処理時間がかかっていることが分かった．

4.3　WAF 機能の確認

WAF としての機能を確認するために，XSS 脆弱性を持った CGI プログラムを Perl 言語で作成し，mod_SVM が入力された XSS 文字列を検知してアクセスを拒否できるか確認した．**図 12** に XSS を持ったページを示す．mod_SVM を使用しない状態でテキスト入力エリアに <script>alert(”hoge”);</script> と入力することで，**図 13** に示すアラートウィンドウが表示される．mod_SVM を使用することで**図 14** に示すように，アクセスが「403 Forbidden」として拒否できることが確認できる．

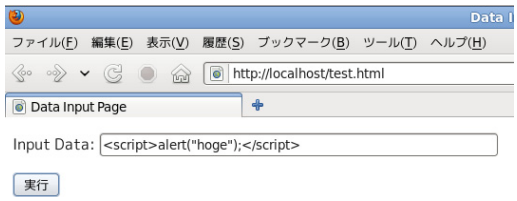

図 12　XSS を持ったページ
Fig. 12　Page with XSS.

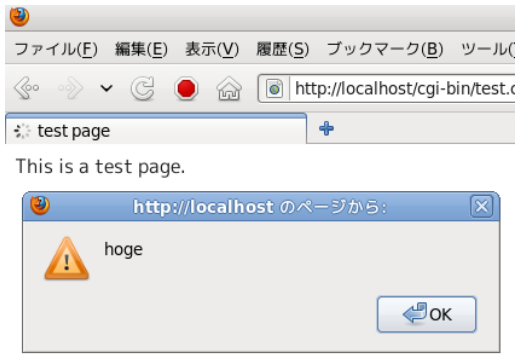

図 13　XSS が成功した場合
Fig. 13　XSS success.

図 14　XSS が失敗した場合
Fig. 14　XSS failure.

表 6　SecRule の引数
Table 6　Argument of SecRule.

引数	必須/任意	意味
VARIABLES	必須	検査対象とする HTTP 通信の特定データを格納した変数を指定する
OPERATOR	必須	検査項目を指定する
ACTIONS	任意	OPERATOR の条件が真となった場合に実行する処理を指定する

5. 考察

ここでは，ModSecurity における XSS 攻撃および SQL インジェクション攻撃に関する異常検知実装法について解説し，本研究における実装上の工夫と処理性能に関する優位性について処理性能評価実験に基づいて考察する．また，評価実験の結果から類似研究との考察を行う．

5.1 ModSecurity の処理フェーズ

ModSecurity はルールに従って異常検知を行う．ルールは以下の 5 つのフェーズで処理される．

- Phase:1 Request headers（REQUEST_HEADERS）
- Phase:2 Request body（REQUEST_BODY）
- Phase:3 Response headers（RESPONSE_HEADERS）
- Phase:4 Response body（RESPONSE_BODY）
- Phase:5 Logging（LOGGING）

また，上記の 5 つのフェーズは以下の Apache における HTTP リクエスト処理の流れの中で行われる．

- post-read-request（Phase:1）
- URI translation
- Header parsing
- access control
- authentication
- authorization
- MIME type checking
- fixups（Phase:2）
- RESPONSE（Phase:3, Phase:4）
- logging（Phase:5）

XSS 攻撃および SQL インジェクション攻撃は主に Phase:1 において検知ルールが定義された設定ファイルに基づいて検査が行われる．設定ファイルには SecRule と呼ばれる検知ルールが記載されており，SecRule は**表 6** に示す 3 つの引数で構成されている．

このうち OPERATOR の部分に正規表現を用いた攻撃パターンなどが記述されており，OPERATOR の条件が真になった場合に ACTIONS に記述された処理が実行されることで異常検知が行われる．

5.2 XSS 攻撃および SQL インジェクション攻撃に関する検知ルールの構成

ここでは，ModSecurity の実装上の工夫と問題点として，検知ルールの記述法について述べる．XSS 攻撃関する検知ルールは ModSecurity Core Rule Set（CRS）の base_rules にある modsecurity_crs_41_xss_attacks.conf において定義されている．主な検知ルールのカテゴリとカテゴリごとの SecRule の数（ルール数）は以下のとおりである．検知ルールは全部で 113 用意されている．

- XSS Filters - Category 1（ルール数：1）
- XSS Filters - Category 2（ルール数：1）
- XSS Filters - Category 3（ルール数：1）
- XSS（ルール数：69）
- Detect tags（ルール数：3）
- Detect event handler names（ルール数：1）
- Detect usage of common URI attributes（ルール数：3）
- JavaScript fragments（ルール数：1）
- CSS attack fragments（ルール数：2）
- Misc（ルール数：5）
- XSS Filters from IE（ルール数：26）

また，SQL インジェクション攻撃に関する検知ルールは modsecurity_crs_41_sql_injection_attacks.conf において定義されている．主な検知ルールのカテゴリとカテゴリごとの SecRule の数（ルール数）は以下のとおりである．検知ルールは全部で 55 用意されている．

- Detect SQL Comment Sequences（ルール数：1）
- SQL Hex Evasion Methods（ルール数：1）
- String Termination/Statement Ending Injection Testing（ルール数：1）
- SQL Operators（ルール数：1）
- SQL Tautologies（ルール数：1）
- Detect DB Names（ルール数：1）
- SQL Keyword Anomaly Scoring（ルール数：18）
- Blind SQL injection（ルール数：1）
- SQL injection（ルール数：6）
- SQL Injection Character Anomaly Usage（ルール数：2）
- PHPIDS - Converted SQLI Filters（ルール数：22）

SecRule の OPERATOR における条件が複雑な正規表現による検査の場合は，処理に時間がかかるため ModSecurity の処理性能上の問題点となる．そのため，ルールは正規表現が単純で検知しやすいルールから記述することで，複雑な正規表現による検査を最小限に行うような工夫が行われている．しかし，正常なリクエストの場合は，上記のルールをすべて通過した上で正常と判断されるため，ModSecurity を導入した場合のレスポンス低下の影響が大きくなっている．

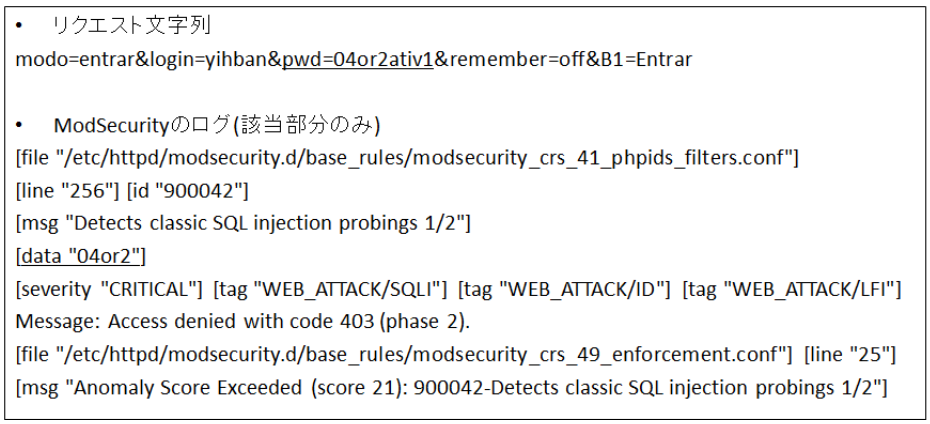

図 15　アクセス文字列と ModSecurity のログ
Fig. 15　Access string and log of ModSecurity.

5.3　本研究における実装上の工夫

先行研究においては，入力データの構文解析などを行って N-gram の生成を行っているが，本研究では，XSS 攻撃および SQL インジェクション攻撃を検知するために，Web サーバへ送られるリクエストを文字単位で N-gram を用いて特徴ベクトルを生成している．文字単位で行うことで構文解析などを行う必要がなく N-gram の生成と頻度の計算についても N-gram をインデックスとした配列において頻度計算をする際に，N-gram のインデックスを N = 1 から N = 4 までについて char 型の配列と long 型の共用体を用いることで処理コストが低くなるように実装している．そのため，リクエスト文字列の長さにあまり影響を受けずに特徴ベクトルを生成することが可能となっている．また，特徴ベクトルを SVM によって識別する際の処理コストについても生成した特徴ベクトルによらず，一定の速度で処理することが可能となっている．そのため，ModSecurity で実装している正規表現を用いたルールベースのマッチングに比べて処理コストが低くなっている．処理性能評価実験の結果からも，本論文における実装が ModSecurity の運用上の問題となっていた処理性能のボトルネック解消に有効であることが分かった．

5.4　他の研究との比較

性能評価実験の結果から本論文において採用した SVM を用いた WAF への異常検知機能と実装したモジュールの性能について，同じデータを用いていないので厳密な比較はできないが，XSS と SQL インジェクションを対象として SVM を用いた類似研究と「特徴ベクトルの生成法」と「識別性能」の観点から考察を行う．

文献 [9] の研究では，アクセスされた文字列から字句解析を行いトークンの並びに変換し，トークン単位で N-gram を用いて特徴ベクトルを生成している．そのため，XSS と SQL インジェクションとで異なる字句解析の基準を必要とする．また，識別性能については Precision = 98.04%, TPR = 0.985, FPR = 0.015 となっている．ここで TPR = TP/(TP + FN)，FPR = FP/(FP + TN) であり，TP はブラックリストのデータを正確に検知できた数，FN はブラックリストのデータをホワイトリストのデータとして検知した数，FP はホワイトリストのデータをブラックリストのデータとして検知した数，TN はホワイトリストのデータを正確に検知できた数である．

また，文献 [12] の研究では，アクセスされた文字列から字句解析を行いトークンの並びに変換し，トークン単位の頻度を用いて特徴ベクトルを生成している．そのため，XSS と SQL インジェクションとで異なる字句解析の基準を必要とする．また，識別性能については SQL インジェクションにおいて Accuracy = 99.16%，Precision = 0.986，Recall = 1.00，XSS において Accuracy = 98.95%，Precision = 0.989，Recall = 0.994 となっている．

これに対して我々が採用した検知手法では，アクセスされた文字列の文字の情報のみに着目して N-gram を用いて特徴ベクトルを生成しているため，XSS および SQL インジェクションの両方を同時に扱うことができる．また，認識性能については Accuracy = 99.98%，Precision = 100.0%，Recall = 99.92%，TPR = 0.9992，FPR = 0 となっている．

ModSecurity が採用している正規表現を用いたパターンマッチングによる異常検知では，XSS や SQL インジェクションの特徴を持つ新たな攻撃パターンが考案されたときに検知漏れが発生する可能性があるが，本論文における検知手法では，学習データ内の攻撃パターンを N-gram により使用されている文字の分布や出現頻度に基づいて構築されたモデル表現するため新たな攻撃パターンについても検知できる可能性が高い．また，正規表現を用いたパターンマッチングでは正常なデータを誤検知するケースが発生する．一例として本論文における検知手法においては正常なリクエスト文字列と判断されたが，ModSecurity において異常なデータと誤検知されたリクエスト文字列について，**図 15** に示す ModSecurity のログから誤検知した理由につ

いて考察する．

リクエスト文字列はログイン ID とパスワードとその他の情報から構成されている．この中で ModSecurity は正規表現によるパターンマッチングにより「pwd=」にセットされた「04or2ativ1」というデータについて，「04or2」の部分を数値と SQL のキーワード “or” の組合せと判断して，最終的には「classic SQL injection」と誤検知している．しかし，このアクセス文字列は正常なデータであることが分かっており，パスワードに数値と “or” の組合せを用いたところ，異常なデータとして検知されている．このことから，ブラックリスト方式のパターンマッチングによる検知では，アクセス文字列の中に一部異常なデータと見なされた部分が含まれている場合，正常なデータを異常なデータとしてしまう False Positive の問題が発生しやすくなることが分かる．本論文で提案した SVM を用いた検知方式では，学習データセット中の正常なデータと異常なデータに基づき N-gram と頻度によりアクセス文字列の特徴を文字列全体で表現することができると考えられ，このことが False Positive 低減にも効果があると考える．今後，より多くのデータと検知結果の考察を行うことで，より精度の高い異常検知システムを構築できる可能性がある．

また，検知性能を評価する場合は，学習データとは異なる評価データを用いた Open Test の結果を用いることが一般的であり，本研究で実装したシステムについては，SVM でモデルを作成するのに用いた学習データと評価実験に用いた評価データでは重複がないデータとなっており，システムにとって未知のデータと考えることができる．一方，ブラックリストのデータを目視および ModSecurity による確認を行っているため，ModSecurity の場合はブラックリストのデータについては 100%の検知となっている．そのため，単純に検知結果のみで判断することは難しいが，ModSecurity と同等もしくは，それ以上の性能を発揮する可能性がある．このことから，Apache の標準的な WAF である ModSecurity と比較した場合，パターンマッチングでの検知結果と同等またはそれ以上の性能を発揮し，処理時間についても大幅に低減させることができたことから，パターンマッチング方式の WAF が直面している 2 つの問題を解決することができるものと考える．

6. まとめ

Web アプリケーションへの攻撃を防ぐ WAF への実装を目的として Web サイトへのリクエスト中のクエリ文字列から文字に着目して N-gram 言語モデルにより特徴ベクトルを生成し，生成した特徴ベクトルを SVM で認識させる異常検知手法を Apache のモジュールとして実装した．実装したモジュールの性能を評価するために Apache の標準的な WAF である ModSecurity との性能比較実験を行った．性能比較実験の結果から，Accuracy = 99.98%，Precision = 100.0%，Recall = 99.92%，F 値 = 0.9996 が得られ，False Positive を低減させながらも ModSecurity を上回る認識性能と処理性能で検知を行えることを明らかにし，実装したモジュールが ModSecurity の代替モジュールとして有効であることを示した．今後の課題としては，学習データにおける XSS および SQL インジェクションのパターンをさらに増やすことと，実運用環境における性能評価があげられる．

謝辞　本研究は科研費 23500106 の助成を受けたものである．

参考文献

[1] OWASP: Cross-site Scripting (XSS) – OWASP, available from ⟨https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)⟩ (2011).
[2] Anley, C.: Advanced SQL injection in SQL Server applications, available from ⟨http://www.nextgenss.com/papers/advanced_sql_injection.pdf⟩ (2002).
[3] Anley, C.: (more) advanced SQL injection, available from ⟨http://www.nextgenss.com/papers/more_advanced_sql_injection.pdf⟩ (2002).
[4] OWASP: SQL Injection - OWASP, available from ⟨https://www.owasp.org/index.php/SQL_Injection⟩ (2012).
[5] 情報処理推進機構：ソフトウェア等の脆弱性関連情報に関する届出状況，入手先 ⟨http://www.ipa.go.jp/security/vuln/report/vuln2012q3.html⟩ (2012).
[6] 情報処理推進機構：安全なウェブサイトの作り方改訂第 4 版，入手先 ⟨http://www.ipa.go.jp/security/vuln/websecurity.html⟩ (2012).
[7] Bolzoni, D. and Etalle, S. and Hartel, P.H.: Panacea: Automating Attack Classification for Anomaly-based Network Intrusion Detection Systems, *Recent Advances in Intrusion Detection* (*RAID*), pp.1–20 (2009).
[8] Pinzon, C., Herrero, A., de Paz, J.F., Corchado, E. and Bajo, J.: CBRid4SQL: A CBR Intrusion Detector for SQL Injection Attacks, *Hybrid Artificial Intelligence Systems, 5th International Conference*, Vol.6077, pp.510–519 (2010).
[9] Choi, J., Kim, H., Choi, C. and Kim, P.: Efficient Malicious Code Detection Using N-Gram Analysis and SVM, *Proc. 2011 14th International Conference on Network-Based Information Systems*, *NBIS '11*, pp.618–621, IEEE Computer Society (online), (2011).
[10] Rawat, R., Zodape, M.: URLAD (URL attack detection) - using SVM, *International Journal of Advanced Research in Computer Science and Software Engineering*, Vol.2, No.1 (2012).
[11] Rawat, R. and Shrivastav, S.: SQL injection attack Detection using SVM, *International Journal of Computer Applications*, Vol.42, No.13, pp.1–4 (2012).
[12] Komiya, R., Paik, I. and Hisada, M: Classification of malicious web code by machine learning, *Awareness Science and Technology* (*iCAST*), *2011 3rd International Conference on* pp.406–411 (2011).
[13] ModSecurity: Open Source Web Application Firewall, available from ⟨http://www.modsecurity.org/⟩.
[14] Ways to improve performance of your server in ModSecurity 2.5, available from ⟨http://www.packtpub.com/article/ways-improve-performance-server-in-modsecurity2.5⟩.

[15] 伊波　靖，高良富夫：SVM を用いた WAF の検知機能の提案，情報処理学会第 73 回全国大会講演論文集（大会優秀賞受賞），pp.445–447 (2011).
[16] 伊波　靖，高良富夫：SVM を用いた WAF への異常検知機能の実装と評価，情報処理学会第 74 回全国大会講演論文集，Vol.3, pp.445–447 (2012).
[17] Cristianini, N. and Shawe-Taylor, J.：サポートベクターマシン入門，共立出版 (2005).
[18] Spanish Scientific Research Council (C.S.I.C.): HTTP DATASET CSIC 2010, available from ⟨http://iec.csic.es/dataset/⟩ (2010).
[19] Chang, C.-C. and Lin, C.-J.: LIBSVM: A library for support vector machines, *ACM Trans. Intelligent Systems and Technology*, 2:27:1–27:27 (2011), available from ⟨http://www.csie.ntu.edu.tw/˜cjlin/libsvm⟩.

伊波 靖　（正会員）

1961 年生．1984 年琉球大学工学部電子・情報工学科卒業．同年沖縄県立八重山商工高等学校電気科教諭．1998 年琉球大学大学院工学研究科修士課程電気・情報工学専攻修了，修士（工学）．2004 年沖縄工業高等専門学校メディア情報工学科講師，2007 年同准教授，2009 年同教授，2009 年情報処理センター長．情報セキュリティとコンピュータフォレンジックスに関する研究に従事．

高良 富夫　（正会員）

1952 年生．1976 年鹿児島大学理学部物理学科卒業．1979 年東京工業大学大学院総合理工学研究科博士前期課程，1983 年後期課程物理情報工学専攻修了，工学博士．1991～1992 年米国カーネギー・メロン大学客員研究員．1995 年琉球大学工学部情報工学科教授，2002 年総合情報処理センター長，2006 年学長補佐，2009 年大学教育センター長，2011 年工学部長．日本音響学会，電子情報通信学会，IEEE 各正員．2006 年日本音響学会九州支部長．1990 年沖縄研究奨励賞．共著：「やわらかい南の学と思想」（2008 年），「沖縄の先端技術」（1988 年）．音声の分析，合成，認識，言語獲得のモデル，琉球語とアジアの言語の音声分析と合成の研究に従事．