Vol.2013-IOT-20 No.11
2013/3/14

ベイジアンフィルタを利用した Web Application Firewallの開発

菱沼 猛[1,a)]　吉浦 紀晃[1,b)]

概要：近年，インターネット上では様々なWebアプリケーションによるサービスが提供されているが，その運用ではセキュリティ対策が大きな課題となる．Web Application Firewall（以下WAF）は，事前に用意された特定の入力のパターンに基づくシグネチャベースの入力検査を行うが，悪意のあるコードやスクリプト，バグ等による個々のWebアプリケーション固有の脆弱性に対して，入力検査のための適切な入力のパターンが常に用意されているとは限らない．そこで本論文では，Webアプリケーションの管理者が提示する拒否すべき入力や許可すべき入力からベイジアンフィルタによる学習を行うWAFを開発した．入力検査では統計情報に基づくベイジアンフィルタとアノマリ検出器を併用し，統計情報にはN-gram統計を利用する．スクリプト言語Pythonを用いて本WAFのプロトタイプを実装し，ランダムに作成したHTTPリクエストによる実験を行った．実験の結果，ベイジアンフィルタによる学習を繰り返すことで入力検査における誤検知および検知漏れが減少していくことが確認できた．

キーワード：Web Application Firewall，ベイジアンフィルタ，アノマリ検出，N-gram

Development of Web Application Firewall by Using Bayesian Filter

Hishinuma Takeshi[1,a)]　Yoshiura Noriaki[1,b)]

Abstract: Recently, many web applications which provide many kinds of services on the Internet and security is important in their operations. Web Application Firewall(WAF) is used for security of web applications. Many of WAFs consists of signature-based filters which have prepared particular input patterns, but the filters do not always have proper patterns for malicious codes, malicious scripts and vulnerability due to bugs of individual web applications. This paper develops a WAF which learns good or bad input patterns by Bayesian filter when the administrators of web applications suggest input which should be rejected or accepted. Input inspection uses a Bayesian filter and an anomaly detector. The anomaly detector is based on statistics, which are data of N-gram of inputs for web applications. This paper implements a prototype of our WAF by script language Python and experiments with randomly-generated HTTP requests. The results of the experiment found that iteration of learning by using Bayesian filter decreases false positives and false negatives.

Keywords: Web Application Firewall, Bayesian Filter, Anomaly Detection, N-gram

1. はじめに

近年，インターネットを通して多様なサービスを提供するWebアプリケーションが様々な場面で利用されている．一方でWebアプリケーションでは不特定多数の利用者か

[1] 埼玉大学大学院理工学研究科
Graduate School of Science and Engineering, Saitama University
[a)] hishinuma@fmx.ics.saitama-u.ac.jp
[b)] yoshiura@fmx.ics.saitama-u.ac.jp

らの入力を受け付けることが多く，セキュリティ対策が大きな課題となる．悪意のある利用者から攻撃を受けた場合や Web アプリケーション自体にバグ等による脆弱性が潜んでいた場合，Web アプリケーションを利用したサービスの提供に悪影響を与えてしまう可能性がある．

Web Application Firewall（以下 WAF）は，上記のような問題への技術的な対策として利用されている．WAF ではあらかじめ拒否すべき入力や許可すべき入力等の特定の入力のパターンを用意しておき，用意された特定の入力のパターンに基づくシグネチャベースの入力検査を行う．しかしこのような WAF は，あらかじめ用意されていないパターンのコードやスクリプト，またバグ等による個々の Web アプリケーション固有の脆弱性に対しては対応することができない．

WAF に関する研究は数多く行われており，近年では統計情報に基づくアノマリ検出器の開発が幾つかの文献で紹介されている [1], [2], [3], [4]．アノマリ検出器とは，過去の入力のログから得られる統計情報を用意し，新たな入力とこの統計情報を比較して新たな入力が異常な値であるかを判断するシステムである．しかし管理者にとって拒否したい入力や許可したい入力があった場合に，新たな入力検査の基準を設定することはできない．

一方で，ベイジアンフィルタはスパムメールフィルタとして既に広く利用されている [5], [6]．フィルタの利用者がスパムメールとスパムでないメールをフィルタに提示することで，それぞれのメールの内容から統計情報を作成して新たに受信したメールを検査し，スパムメールであるかどうかを判断する．

そこで本論文ではアノマリ検出器による入力検査を行いつつ，ベイジアンフィルタを利用して管理者が提示する拒否すべき入力や許可すべき入力の例から学習を行い新たな入力検査の基準を設定する方法を提案する．本論文で紹介する WAF では，Web アプリケーションへの入力として GET メソッドを用いた HTTP リクエストを扱い，HTTP リクエストに含まれる各要素の入力文字列における N-gram 統計を統計情報として記録する．入力検査ではベイジアンフィルタとアノマリ検出器を併用し，それぞれが別々に記録した N-gram 統計を用いて入力検査を行う．さらに管理者から提示された拒否すべき入力または許可すべき入力によって，ベイジアンフィルタを用いた学習を行う．

本論文では第 2 節において開発した WAF の概要について述べ，第 3 節で実験結果を示す．第 4 節では今後の課題を述べる．

2. 開発した WAF

開発した WAF（以下本 WAF）では Web アプリケーションへの入力に対する入力検査と管理者から提示された入力からの学習が可能である．ここでは本 WAF の概要について述べる．

```
GET /opac/opac_suggest.cgi?q=opac HTTP/1.1
Host:
User-Agent: Mozilla/5.0 (Windows NT 6.0; rv:18.0) Gecko/20100101 Firefox/18.0
Accept: */*
Accept-Language: ja,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
X-Requested-With: XMLHttpRequest
Referer:
Connection: keep-alive
```

図 1 HTTP リクエストの例
Fig. 1 Example of HTTP request

2.1 入力

本 WAF が扱う入力は GET メソッドを用いた HTTP リクエストとする．GET メソッドを用いた HTTP リクエストは図 1 のように複数行から構成されており，先頭行のリクエストラインとそれ以外の行のリクエストヘッダフィールドが含まれる．リクエストラインでは URL のパスや CGI への引数が，リクエストヘッダフィールドではリクエストヘッダとして送信される値が記述されており，それぞれの要素に対して Web アプリケーションの利用者からの入力がある．このように複数の要素に対しそれぞれ入力があるが，Web アプリケーションに対する攻撃やバグによる不具合を引き起こす原因となる入力はどの要素に対して入力されるかわからない．そのため文献 [1], [4], [7] では，HTTP リクエストに含まれる各要素に着目し，各要素ごとにその入力の内容に基づく統計情報の作成と入力検査を行っている．本 WAF ではこれらの文献と同様に，HTTP リクエストに含まれる各要素に対する入力からベイジアンフィルタおよびアノマリ検出器で用いる統計情報を各要素ごとに区別して作成する．本 WAF で扱う HTTP リクエストの要素を次のように定義する．

定義 1 HTTP リクエストの要素とは以下のものを指す．

- リクエストラインに含まれる URL のパス
- リクエストラインに含まれる CGI に対する各引数
- リクエストヘッダフィールドに含まれる各リクエストヘッダ

パスに対する入力ではパスに指定される文字列のみで統計情報を作成し，CGI のある引数に対する入力ではその引数に対する入力（引数名と「=」で結ばれた右辺の文字列）のみで統計情報を作成する．リクエストヘッダに関しても同様にあるリクエストヘッダに対しての入力（リクエストヘッダと「:」で結ばれた右辺の文字列）のみで統計情報を作成する．その後の入力検査も各要素ごとに行う．

2.2 N-gram 統計

前述のようにベイジアンフィルタとアノマリ検出器では統計情報を利用する．本 WAF では統計情報として N-gram 統計を用いる．N-gram 統計では，文字列中の N 文字の部

分文字列についての頻度情報を記録する．各要素の入力文字列についてではなくその部分文字列についての頻度情報を記録することで，既知の入力文字列だけでなく部分的に変化した未知の入力文字列であっても対応できる．アノマリ検出器の開発においては，文献 [1], [2], [8] 等多くの研究で N-gram 統計が利用されている．またスパムメールフィルタとしてのベイジアンフィルタにおいても，統計情報として文字単位の N-gram 統計を利用する研究が行われている [9]．本 WAF においては 2.1 で述べたように，各要素に対する入力文字列から各要素ごとの N-gram 統計を記録する．なお本 WAF では N=2 とし，入力文字列に含まれる 2 文字の部分文字列についての頻度情報を記録する．

本 WAF において，各要素に対する入力文字列から各要素ごとにそれぞれ作成する N-gram 統計 N に含める情報を次のように定義する．

定義 2　本 WAF における N-gram 統計 N に含める情報は以下のものとする．

(1) 要素名
(2) 入力文字列 w から抽出した全ての 2 文字の部分文字列 c_ic_j の出現回数 $f(c_ic_j)$
(3) 入力文字列 w から抽出した全ての 2 文字の部分文字列 c_ic_j の遷移頻度 $p(c_ic_j)$

(1) は，定義 1 で定義した HTTP リクエストの要素のうち，どの要素に対する入力文字列における N-gram 統計かを区別するための情報である．(2) では，(1) の要素に対する入力文字列 w から全ての 2 文字の部分文字列 c_ic_j を抽出し，その出現回数 $f(c_ic_j)$ を記録する．ここで c_ic_j は，入力文字列 w において先頭から i 番目と j 番目の連続する 2 文字の文字列を表す．なお文字 c_i には入力文字列の先頭であることを示す仮想的な先端文字を含め，文字 c_j には入力文字列の終端であることを示す仮想的な終端文字を含める．先端文字および終端文字を含めることで，要素への入力文字列 w の長さが 0 である場合にも先端文字と終端文字の 2 文字の入力文字列として対応することができる．(3) では，(2) で記録した文字列 c_ic_j の出現回数 $f(c_ic_j)$ から，文字列 c_ic_j において文字 c_i の後に文字 c_j に遷移する頻度を文字列 c_ic_j の遷移頻度 $p(c_ic_j)$ とし，以下の計算で求める．

$$p(c_ic_j) = \frac{f(c_ic_j)+1}{\sum_{c_k \in C} f(c_ic_k) + 96} \tag{1}$$

ここで C は，文字 c_i の後に続く文字がとりうる全ての文字の集合である．なおここではラプラススムージング [10] と呼ばれる N-gram 統計における頻度情報の調整を行う．式 (1) の計算により，全ての 2 文字の文字列に対して最低 1 回の出現回数を与えた場合と同様の結果を得る．本 WAF で扱う文字は半角英数字（英字においては大文字と小文字を区別する）および半角記号であり，前述の仮想的な終端文字も含めると，文字列 c_ic_j において文字 c_j がとり得る文字は 96 通りである．

記録した N-gram 統計からは，各要素に対する入力文字列 w の生成頻度を計算することができる．長さ l の入力文字列 w の生成頻度 $P(w)$ を以下の式で求める．なおこの計算でも前述の仮想的な先端文字と終端文字を考慮している．

$$P(w) = \prod_{i=1}^{l+1} p(c_ic_{i+1}) \tag{2}$$

このように入力文字列 w の生成頻度は文字列の遷移頻度の積で表されるため，入力文字列 w に含まれる部分文字列のうち遷移頻度が 0 となるものが 1 つでもあった場合，$P(w)$ の値が 0 になってしまう．そのため式 (1) の計算の際には，前述のラプラススムージングによって文字列 c_ic_j の遷移頻度 $p(c_ic_j)$ の値が 0 になることを防いでいる．本 WAF における入力検査および学習は，$P(w)$ の値に基づいて行われる．

2.3　入力検査

入力検査は，ベイジアンフィルタとアノマリ検出器を用いて各要素ごとに行う．ここでは，アノマリ検出器，ベイジアンフィルタの順でそれぞれの入力検査の手順を説明し，その後 2 つの入力検査を組み合わせた利用方法について示す．

2.3.1　アノマリ検出器

本 WAF におけるアノマリ検出器は，文献 [1], [4] で紹介されているアノマリ検出器を参考にした．まず Web アプリケーションのログに含まれる過去の入力から，定義 2 で定義した各要素の入力文字列における N-gram 統計 N_{log} を作成する．その後，作成した N_{log} における文字列の遷移頻度を用いて計算される各要素の入力文字列 w の生成頻度 $P_{log}(w)$ の値によって入力検査を行う．ただし式 (2) で計算される入力文字列の生成頻度は文字列の遷移頻度の積であり，結果の値が入力文字列 w の長さ l に依存する．そのため入力文字列 w の長さ l と仮想的な先端文字と終端文字を考慮した上で $l+1$ による幾何平均をとり，その値をアノマリ検出器の出力 $A(w)$ とする．

$$A(w) = \sqrt[l+1]{P_{log}(w)} \tag{3}$$

アノマリ検出器では，$A(w)$ の値が 0 に近づくほど異常な入力であると判断する．$A(w)$ の値に対して閾値 t_A を設定し，入力検査を行う．

2.3.2　ベイジアンフィルタ

本 WAF におけるベイジアンフィルタでは，拒否すべき入力の統計情報を記録する N-gram 統計 N_{bad} と，許可すべき入力の統計情報を記録する N-gram 統計 N_{good} を作成し，この 2 つの N-gram 統計に基づいて入力検査を行う．N_{bad} と N_{good} は 2.3.1 で作成した N_{log} と同様の方法で作成し，その初期状態は N_{log} と同じものとする．しかし管理者が提示する拒否すべき入力または許可すべき入力による

学習を繰り返すことで N_{bad} と N_{good} は次第に変化していく．提示された入力による学習については，2.4 で述べる．

N_{bad} と N_{good} それぞれにおける文字列の遷移頻度を用いて，各要素の入力文字列 w の生成頻度 $P_{bad}(w)$ と $P_{good}(w)$ を求める．その後入力文字列 w を拒否すべきかどうかを以下の式で求め，ベイジアンフィルタの出力 $B(w)$ とする．

$$B(w) = \frac{P_{bad}(w)}{P_{bad}(w) + P_{good}(w)} \tag{4}$$

ベイジアンフィルタでは，$B(w)$ の値が 1 に近づくほど拒否すべき入力文字列であると判断し，逆に 0 に近づくほど許可すべき入力文字列であると判断する．$B(w)$ の値に対して閾値 t_B を設定し，入力検査を行う．

なお，ベイジアンフィルタによる入力検査で用いる式 (4) の $P_{bad}(w)$ と $P_{good}(w)$ における入力文字列 w は同一の文字列であり，$B(w)$ の値に文字列 w の長さは影響しない．そのためベイジアンフィルタによる入力検査では幾何平均はとらない．

2.3.3 2 つの入力検査を組み合わせた利用方法

前述のように，本 WAF における入力検査はベイジアンフィルタとアノマリ検出器を用いて各要素ごとに行う．本 WAF では，1 つの HTTP リクエストに含まれる複数の要素のうち，下記の入力検査で拒否される入力文字列を受け取る要素が 1 つ以上あった場合，その要素が含まれる HTTP リクエスト全体の入力を拒否する．また本 WAF における入力検査では，学習した入力検査の基準を優先するためにベイジアンフィルタによる入力検査を優先する．各要素の入力文字列 w に対する入力検査の手順を図 2 に示す．

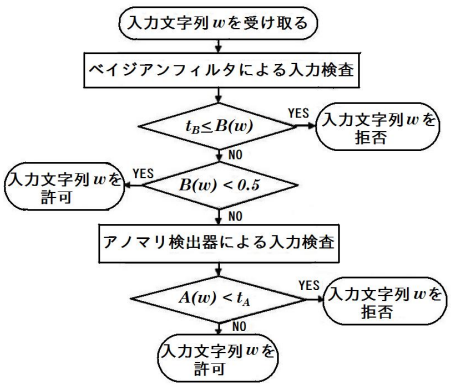

図 2 各要素の入力文字列 w に対する入力検査の手順
Fig. 2 Filtering flowchart for input string w in each element

まず始めにベイジアンフィルタによる入力検査を行う．入力文字列 w に対し，$B(w)$ の値が t_B 以上となった場合は入力文字列 w を拒否する．2.3.2 で述べたように，ベイジアンフィルタでは入力文字列 w に対する $B(w)$ の値が 1 に近づくほど拒否すべき入力文字列であると判断し，逆に 0 に近づくほど許可すべき入力文字列であると判断する．ここで $B(w)$ の値が 0.5 となる場合はちょうど中間の値であり，ベイジアンフィルタによる入力検査の結果に曖昧さが残ることとなる．そのため，$B(w)$ の値がちょうど中間の値である 0.5 以上の値となる場合には，アノマリ検出器により再度入力検査を行う．$B(w)$ の値が 0.5 未満となる場合は，アノマリ検出器による入力検査を行うことなく入力文字列 w を許可する．アノマリ検出器による入力検査では，$A(w)$ の値が t_A 未満であれば入力文字列 w を拒否し，t_A 以上であれば入力文字列 w を許可する．

2.4 学習

ベイジアンフィルタで用いる 2 つの N-gram 統計 N_{bad} と N_{good} は，初期状態ではアノマリ検出器で用いる N_{log} と同じものであるが，管理者が提示する拒否すべき入力と許可すべき入力による学習を繰り返すことで変化していく．以下に拒否すべき入力が提示された場合と許可すべき入力が提示された場合の本 WAF の学習の際の動作を示す．

拒否すべき入力が提示された場合

提示された入力の各要素の入力文字列 w に含まれる全ての 2 文字の部分文字列 c_ic_j に対して，次の操作を同時に行う．

- N_{bad} における文字列 c_ic_j の出現回数 $f_{bad}(c_ic_j)$ を 1 増加する．
- N_{good} における文字列 c_ic_j の出現回数 $f_{good}(c_ic_j)$ を 1 減少する．$f_{good}(c_ic_j)$ は 0 未満の値にはしない．

以上の操作を，提示された入力に含まれるいずれかの要素における $B(w)$ の値が t_B 以上になるまで繰り返す．

許可すべき入力が提示された場合

提示された入力の各要素の入力文字列 w に含まれる全ての 2 文字の部分文字列 c_ic_j に対して，次の操作を同時に行う．

- N_{good} における文字列 c_ic_j の出現回数 $f_{good}(c_ic_j)$ を 1 増加する．
- N_{bad} における文字列 c_ic_j の出現回数 $f_{bad}(c_ic_j)$ を 1 減少する．$f_{bad}(c_ic_j)$ は 0 未満の値にはしない．

以上の操作を，提示された入力に含まれる全ての要素における $B(w)$ の値が 0.5 未満になるまで繰り返す．

N_{bad} と N_{good} における文字列 c_ic_j の出現回数を変化させることで，文字列 c_ic_j の N_{bad} と N_{good} における遷移頻度および入力文字列 w の生成頻度を変化させ，式 (4) で定めた $B(w)$ の値を変化させる．ベイジアンフィルタでは $B(w)$ の値に基づいて入力検査を行うため，N_{bad} と N_{good} における文字列 c_ic_j の出現回数の変化はベイジアンフィルタによる入力検査の結果に影響する．また，文字列 c_ic_j の出現回数を 0 未満の値にしない理由は，遷移頻度が負の値となった場合にその積で求められる入力文字列 w の生成頻度までもが負の値をとりうることとなり，入力文字列の生成頻度を正しく求めることができなくなるためである．

しかし文字列 c_ic_j の出現回数を強制的に変化させることで，文字列 $c_i\bar{c_j}$（$\bar{c_j}$ は c_j 以外の全ての文字を表す）の遷移頻度が同時に変化してしまう．これは文字列 c_ic_j の遷移頻

度 $p(c_ic_j)$ を，式 (1) で示す通り相対頻度の計算により求めるためである．文字列 $c_i\bar{c_j}$ の遷移頻度 $p(c_i\bar{c_j})$ が変化すると，文字列 $c_i\bar{c_j}$ を部分文字列として含む入力文字列 w' に対する $B(w')$ の値も変化することになる．その結果，提示された入力以外の入力における誤検知や検知漏れが発生する．そのため文字列 c_ic_j の遷移頻度 $p(c_ic_j)$ を，ベイジアンフィルタにおいては以下のように改め $p_B(c_ic_j)$ とする．

$$p_B(c_ic_j) = \frac{f_{bad}(c_ic_j)_{or}f_{good}(c_ic_j)+1}{\sum_{c_k\in C} f_{log}(c_ic_k)+96} \quad (5)$$

$f_{bad}(c_ic_j)_{or}f_{good}(c_ic_j)$ は，N_{bad} または N_{good} における文字列 c_ic_j の出現回数であり，$f_{log}(c_ic_j)$ は N_{log} における文字列 c_ic_j の出現回数である．分母を N_{bad}，N_{good} ではなくその初期状態である N_{log} における出現回数で固定し，文字列 $c_i\bar{c_j}$ の遷移頻度の変化を防ぐ．また遷移頻度 $p_B(c_ic_j)$ の最大値は 1 とし，その値が無限に増加することを防ぐ．

どちらの場合でも繰り返しの処理が行われるが，前述の通り文字列 c_ic_j の出現回数 $f_{bad}(c_ic_j)$ と $f_{good}(c_ic_j)$ の最小値は 0 としている．その場合文字列 c_ic_j の遷移頻度 $p_B(c_ic_j)$ は式 (5) より 0.02 未満の値となる．したがって入力文字列 w に含まれる全ての部分文字列の遷移頻度が最小値をとったとき，入力文字列 w の生成頻度は 0.02 未満の値となる．一方で，文字列 c_ic_j の遷移頻度 $p_B(c_ic_j)$ の最大値は 1.0 としている．したがって入力文字列 w に含まれる全ての部分文字列の遷移頻度が最大値をとったとき，入力文字列 w の生成頻度は 1.0 となる．よって式 (4) より，$B(w)$ の最小値は 0.02/1.02 未満すなわち 0.02 未満の値となり，最大値は 1.0/1.02 以上すなわち 0.98 以上の値となる．そのため入力検査における $B(w)$ の閾値 t_B が 0.98 以下であればいずれ拒否すべき入力が提示された場合の処理は停止する．許可すべき入力が提示された場合の処理においても，$B(w)$ の値がいずれ 0.5 を下回るため停止する．

3. 実験

本 WAF のプロトタイプをスクリプト言語 Python を用いて実装し，ランダムに作成したテストデータを用いて実験を行った．テストデータは埼玉大学図書館の OPAC システム [11] へ実際に送信されている HTTP リクエストを参考にして 1,000,100 個作成し，そのうち 100 個が意図的に作成した拒否すべき入力である．拒否すべき入力に含まれる要素への入力文字列としては，値として数値をとると考えられる要素に対しては限界値以上の値，パスに対してはディレクトリトラバーサルを実現するための文字列や存在しないパスの指定，CGI の引数に対しては SQL インジェクションやコマンドインジェクションを実現するための文字列やメタ文字の挿入等を行った．

3.1 実験方法

テストデータを用いて次の手順で実験を行った．

(1) 用意した HTTP リクエストを，等しい大きさの 4 つの集合 S_1，S_2，S_3，S_4 に分割する．各々の集合には 250,025 個の HTTP リクエストが含まれ，そのうち 25 個が拒否すべき入力である．
(2) S_1 を用いて，N_{log}，N_{bad}，N_{good} を作成する．
(3) S_2 に含まれる HTTP リクエストにおける全ての要素に対して $A(w)$ の値を求め，各要素における $A(w)$ の最小値をその要素におけるアノマリ検出器の閾値 t_A として設定する．
(4) S_3 に対し学習前の状態で入力検査を行い，その結果得られる誤検知と検知漏れの入力をそれぞれ学習用の許可すべき入力と拒否すべき入力とする．
(5) 学習の前後で S_3 および S_4 に対して入力検査を行い，それぞれの集合における学習の効果を確認する．

S_2，S_3，S_4 に対しては入力検査を行う．入力検査では，作成した拒否すべき入力以外を拒否してしまった場合には誤検知，許可してしまった場合には検知漏れと評価する．

3.2 実験結果

実験結果を以下に示す．

3.2.1 学習による誤検知と検知漏れの数の変化

学習前の S_3 に対する入力検査から得られた誤検知と検知漏れの入力による学習で，学習後の誤検知と検知漏れの数がどのように変化するかを確認した．誤検知と検知漏れの入力を全て学習した場合，誤検知と検知漏れの入力をその総数の 1/2 だけ学習した場合，誤検知と検知漏れの入力をその総数の 1/4 だけ学習した場合とで比較している．なお，誤検知と検知漏れの入力はランダムに並べ替え順次提示している．表 1，2 に結果を示す．

提示された入力からの学習によって S_3 では誤検知と検知漏れの数が，S_4 では誤検知の数が減少することが確認できる．S_4 における検知漏れの数が減少しなかった理由は，学習用の入力を含む S_3 における拒否すべき入力が S_4 における拒否すべき入力とは異なる内容をもつものであったためと考えられる．また，学習量が多いほどその後の誤検知や検知漏れの減少につながることが確認できる．

表 1 誤検知と検知漏れの学習 (S_3)
Table 1 Learning of false positives and false negatives(S_3)

	学習前	学習量 1/4	学習量 1/2	全て学習
誤検知	145	21	17	0
検知漏れ	10	10	12	6

表 2 誤検知と検知漏れの学習 (S_4)
Table 2 Learning of false positives and false negatives(S_4)

	学習前	学習量 1/4	学習量 1/2	全て学習
誤検知	180	55	52	28
検知漏れ	10	11	10	11

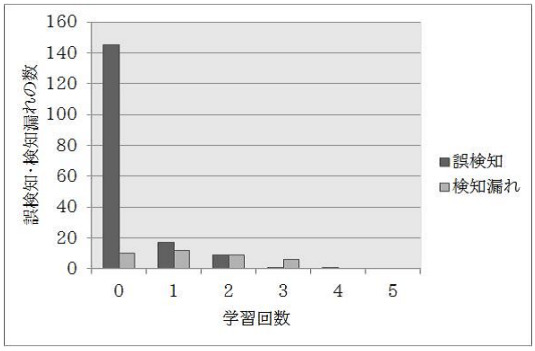

図 3 繰り返し学習を行った結果 (S_3)
Fig. 3 Result of iterated learning(S_3)

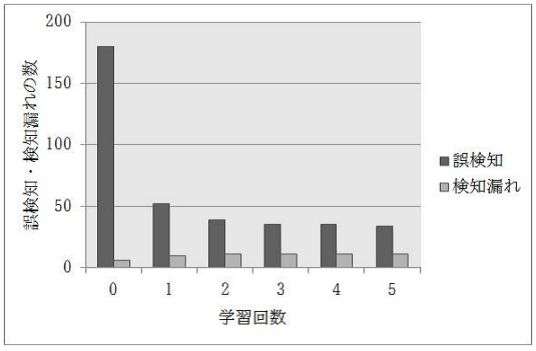

図 4 繰り返し学習を行った結果 (S_4)
Fig. 4 Result of iterated learning(S_3)

3.2.2 誤検知と検知漏れの数の収束

学習によって誤検知と検知漏れの数が変化することが 3.2.1 で確認できたが，検知漏れの数が増加する場合もあった．そのためここでは誤検知の入力と検知漏れの入力による学習を繰り返した場合にその数が一定の値に収束するのかどうかを確認するため，S_3 に対する入力検査とその結果得られる誤検知と検知漏れの入力による学習を繰り返す．また，通常は誤検知や検知漏れの入力があったとしてもそれらを漏れなく全て提示できるとは限らない．そのため今回はそれらの総数の半数を用いることとした．図 3，4 に結果を示す．

結果から，繰り返し学習を行うことで誤検知と検知漏れの数がやがて一定の値に収束していくことが確認できる．

4. まとめと今後の課題

4.1 まとめ

本論文では，Web アプリケーションへの入力に対する入力検査とベイジアンフィルタによる学習が可能な WAF を開発した．開発した WAF ではベイジアンフィルタとアノマリ検出器を併用し，それぞれ HTTP リクエストから得られる N-gram 統計を統計情報として利用する．

本 WAF のプロトタイプを実装して実験を行った結果，ベイジアンフィルタによる学習を行うことで既存の入力検査における誤検知や検知漏れの数が減少していくことが確認できた．

4.2 今後の課題

今後の課題としては，まず入力の提示方法の改善が挙げられる．拒否すべき入力や許可すべき入力を漏れなくかつ誤りなく提示し学習することは実際には難しいと考えられるためである．この課題への対策として，ネットワーク上を流れている Web アプリケーションへの入力が提示すべき入力であるかどうかを事前に本 WAF 側でチェックするという方法が考えられる．具体的な方法としては，HTTP リクエストのみではなく HTTP レスポンスに対する統計情報も作成し，更なる通信内容の解析を行うことで本 WAF に提示すべき入力の候補を絞り込む．

次に，文章等の入力がありうる POST メソッドへの対応が挙げられる．入力として文章を扱う場合には，現在のように入力を単なる文字列として扱うのではなく構文解析等の前処理を行った上で統計情報を作成する必要がある．

参考文献

[1] Krueger, T., Gehl, C., Rieck, K. and Lascov, P.: TokDoc: A Self-Healing Web Application Firewall, *25th Symposium on Applied Computing*, pp. 1846–1853 (2010).
[2] Wang, K., Parekh, J. and Stolfo, S.: Anagram: A content anomaly detector resistant to mimicry attack, *Recent Adances in Intrusion Detection(RAID)*, pp. 226–248 (2006).
[3] Wang, K. and stolfo, S.: Anomalous Payload-based Network Intrusion Detection, *Symposium on Recent Advances in Intrusion Detection* (2004).
[4] Song, Y., Keromytis, A. and Stolfo, S.: Spectrogram: A mixture-of-marcov-chains model for anomaly detection in web traffic, *Network and Distributed System Security Symposium(NDSS)* (2009).
[5] Graham, P.: A Plan for Spam, , available from ⟨http://paulgraham.com/better.html⟩ (accessed 2012-9-12).
[6] Graham, P.: Better Bayesian Filtering, , available from ⟨http://paulgraham.com/better.html⟩ (accessed 2012-9-12).
[7] Düssel, P., Gehl, C., Lascov, P. and Rieck, K.: Incorporation of Application Layer Protocol Syntax into Anomaly Detection, *International Conference on Information Systems Security(ICISS)*, pp. 188–202 (2008).
[8] Rieck, K. and Laskov, P.: Detecting unknown network attacks using language models, *Detection of Intrusions and Malware, and Vulnerability Assesment, Proc. of 3rd DIMVA Conference*, pp. 74–90 (2006).
[9] 藤田拓也，松本章代，テュールスト マーティンヤコブ：ベイジアンフィルタにおける言語知識を用いないトークン抽出方式の提案と評価，情報処理学会論文誌， Vol. 50, No. 9, pp. 2182–2192 (2009).
[10] 北　研二：言語と計算 4 確率的言語モデル，東京大学出版会 (1999).
[11] : 埼玉大学図書館 OPAC システム，Saitama University（オンライン），入手先 ⟨http://opac.saitama-u.ac.jp⟩（参照 2012-11-30）.