Web アプリケーションのパラメタを悪用する攻撃のアノマリ検知手法

鐘　揚† 朝倉　浩志† 高倉　弘喜‡ 大嶋　嘉人†

†NTT セキュアプラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11
{zhong.yang, asakura.hiroshi, oshima.yoshihito}@lab.ntt.co.jp

‡名古屋大学情報基盤センター
464-8601 愛知県名古屋市千種区不老町
takakura@itc.nagoya-u.ac.jp

あらまし Web アプリケーションの脆弱性を悪用した攻撃とその被害が後を絶たない．中でも，SQL インジェクションやコマンドインジェクションなどといった Web アプリケーションのパラメタを悪用するタイプの攻撃と，それらによる Web サーバ内のコンテンツの改ざん，情報の漏洩といった被害は 2012 年以降増加傾向にある．これらの攻撃を検知する手法としては，未知の攻撃も検知し得るという点でアノマリ検知のアプローチが有望である．しかし，既存手法はパラメタ値そのものを特徴として正常モデルを作成し，攻撃の検知に用いるため，正常モデル作成時に与えた学習データとわずかに異なる正常なリクエストを攻撃と判定してしまう誤検知が多いという問題があった．そのため本稿ではパラメタ値として与えられる文字列を抽象化した型構造で捉える新たなアノマリ検知手法を提案する．また，実験により提案手法が既存手法に比べ検知率を同程度に維持しながら誤検知率を減少させられることを示す．

An Anomaly Detection Method for Parameter Manipulation Attacks to Web Application

Yang Zhong† Hiroshi Asakura† Hiroki Takakura‡ Yoshihito Oshima†

†NTT Secure Platform Labotatories
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585, JAPAN
{zhong.yang, asakura.hiroshi, oshima.yoshihito}@lab.ntt.co.jp

‡Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 461-8601, JAPAN
takakura@itc.nagoya-u.ac.jp

Abstract Web attacks that exploit vulnerabilities of web applications are still major problems. Especially, the number of attacks that maliciously manipulate parameters of web applications such as SQL injections and Command injections is increasing from 2012. To detect these attacks particularly in the case of unknown, anomaly detection is effective. However, existing anomaly detection methods often raise false alarms to normal requests whose parameters slightly differ from those of learning data because they use an exact string which appears in parameter values as features of normal models. In this paper, we propose a new anomaly detection method using abstract structure of parameter values as features of normal models. The results of experiments show that our approach decreases false positive rate than existing methods with similar detection rate.

1 はじめに

Web アプリケーションは今や種々様々なサービスの実現手段に採用され，社会基盤を支える重要な役割を担っている．しかし，Web アプリケーションの脆弱性は絶えることなく発見され，それらを悪用する新たな攻撃が次々と生み出されている．中でも，SQL インジェクションやコマンドインジェクションなど Web アプリケーションのパラメタに不正なコードを挿入し Web アプリケーションの異常な動作を引き起こす，いわゆる，パラメタを悪用する攻撃が，コンテンツ改ざんや情報漏洩といった深刻な被害を次々ともたらしている．Web アプリケーションの攻撃検知にはシグネチャ検知とアノマリ検知の 2 つのアプローチがあり，パラメタを悪用する攻撃にも有効である．シグネチャ検知は既知の攻撃のペイロードから特徴的な部分を抽出し，部分文字列や正規表現の形で表したシグネチャとして予め用意し，検査対象である HTTP リクエストがシグネチャにマッチした場合，攻撃として検知する方式である．しかし，攻撃ペイロードの難読化によってシグネチャへのマッチを回避されてしまう．また予めシグネチャを用意しておくことができないため未知の攻撃の検知は期待できない，という課題がある．

アノマリ検知は正常な HTTP リクエストを学習して正常モデルを作成しておき，検査対象である HTTP リクエストが正常モデルから乖離した場合，攻撃として検知する方式である．そのため，例え攻撃ペイロードが既知のものと異なったとしても，同一の正常モデルを用いて検知することが可能であり，未知の攻撃も検出できる可能性がある．

Web アプリケーションのパラメタを悪用する攻撃に対してアノマリ検知のアプローチでの検知を試みた既存の手法に次の 2 つがある．Kruegel らの手法 [1] では HTTP リクエストのパラメタから，文字列長，文字分布，文字列構造などの複数の特徴を抽出して，正常モデルとする手法である．Kim らの手法 [2] では Kruegel らの手法と同様の特徴に加えて学習モデル選択の概念を導入し，学習モデルをパラメタ毎に使い分けることで検知率，誤検知率を改善させている．しかし，これらの手法ではパラメタ値そのものを文字列構造として用いるため，正常モデルを作成する際に与えられた学習データでは正常なパラメタ値のバリエーションを網羅することができず，正常なパラメタ値も攻撃として誤検知してしまうという課題が存在する．本稿ではこの課題を解決するために，パラメタ値として与えられる文字列を抽象化した型構造で捉える新たなアノマリ検知手法を提案する．また，実験により提案手法は既存の手法に対して検知率を同程度に維持しながら，誤検知率を減少させることを示す．

2 章では以降の議論の準備として，Web アプリケーションのパラメタのアノマリ検知に関する共通的な事項を整理する．3 章では既存手法で用いられる特徴について言及し，特に，本稿で注目する特徴による正常モデルの生成方法と課題について詳しく述べる．4 章では提案手法について解説を行う．5 章では既存手法と提案手法の比較評価を目的として実施した実験の方法について述べ，6 章ではその結果について評価及び考察を行う．最後に，7 章では提案の内容と効果，並びに，今後の課題について述べ，本稿をまとめる．

2 準備

Web アプリケーションのパラメタのアノマリ検知では，HTTP リクエストの URL 部から抽出した各パスの各パラメタ毎に正常モデルを作成し，各リクエスト毎に攻撃か否かの検知を行う．例えば，以下の HTTP リクエストがあった場合，

GET /index.php?id=1&name=Alice

パス部は “/index.php” であり，パラメタ部は “id=1&name=Alice” である．パラメタ部は 0 個以上のパラメタを含み，各パラメタは (パラメタ名，パラメタ値) のような組で表現できる．同一のパラメタ名であっても，異なるパスで現れる場合は別のパラメタとして扱う．

学習時は，学習データとして与えられた HTTP リクエストの URL 部からパラメタを抽出し，パ

ラメタ毎にパラメタ値の特徴を抽出して正常モデルを作成する．検知時は学習時と同様に検査対象となる HTTP リクエストの URL 部に含まれるパラメタを抽出し，各パラメタに対応する正常モデルと比較して乖離が大きい場合に攻撃として検知する．アノマリ検知手法の設計においては，パラメタ値の特徴の選択と特徴の取り方，すなわち，正常モデルの構成方法と、正常モデルと検査対象との比較方法が要点となる．

3 既存手法及びその課題

Kruegel らの手法 [1] では，パラメタ値の文字列長，文字分布，文字列構造，列挙性など，6 つの特徴が提案されている．文献 [1] によれば，その中でもパラメタ値の文字列構造が最も多様な攻撃を検知できる特徴であり，本稿でもこの特徴に着目して論じる．Kruegel らの手法ではパラメタ値の各文字を入力とする非決定性オートマトン (NFA) を正常モデルとしている．正常モデルの作成方法は 2 ステップに分かれている．まず学習データに表れる当該のパラメタそれぞれについて，各パラメタ値を 1 文字ずつに分解し，各文字を入力とする NFA を作成する．次に作成された NFA に対して状態数が最小となるよう状態の結合を行い [3]，その結果を正常モデルとする．例えばあるパラメタについて，学習データの中に “ab” と “abab” の 2 種類のパラメタ値が存在した場合，まず図 1 に示す NFA が作成される．その後状態数が最小となるよう状態結合を繰り返して得られる図 2 に示す NFA が正常モデルとなる．

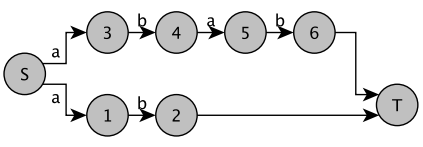

図 1: 状態結合前の NFA

検知時はこの NFA から検査対象のパラメタ値が出力できるか否かで正常か攻撃かを判別する．出力可能であれば正常，不可能であれば攻撃と判定される．例えば，図 2 が正常モデルで

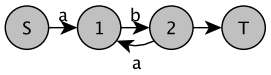

図 2: 状態結合後の NFA

あった場合，“ababab” は出力可能であるため正常と判定されるが，“abc” は出力不可能であるため攻撃と判定される．しかし，この手法ではパラメタ値に現れる文字そのものを用いて正常モデルを作成するため，学習データにおける正常なパラメタ値の網羅性が必要となる．そのためパラメタ値のばらつきが大きいパラメタでは，十分な学習データを用意することが困難であり，その結果として正常なパラメタ値に対する誤検知を多く発生してしまうという課題が存在する．

Kim らの手法 [2] における学習では，まずパラメタ値が特定クラスを持つかどうかを判断する，特定クラスとは URL，ディレクトリ，メールアドレスなど文字列の型に相当するものである．特定クラスが存在する場合それを正常モデルとする．特定のクラスが存在しない場合，特殊文字で各パラメタ値を分割し，分割した文字を文字ブロック列とし，各文字ブロックを入力とする決定性オートマトン (DFA) を作成する．特殊文字とは “?”，“/” などのような記号である．例えば，パラメタ値が “Pukiwiki/1.4” の場合，

(Pukiwiki, /, 1, ., 4)

という文字ブロック列が作成される．その後，文献 [4] に示される手法によって状態結合を行う．検知では検査対象のパラメタに対して特定クラスが存在する場合，検査対象のパラメタ値がその特定クラスに属する値であれば正常と判定され，属さなければ攻撃と判定される．検査対象のパラメタに特定クラスが存在しない場合，正常モデルである DFA から検査対象のパラメタ値が出力できるか否かで正常か攻撃かを判別する．しかしこの手法では正常なパラメタ値が単純な構造を持ち，特定クラスに該当する場合は良いが，複数の特定クラスで形成されている場合，Kruegel らの手法と同様，パラメタ値に現われる文字そのものを用いて正常モデル (DFA) を作成するため，学習データに現れなかった正

常な入力に対して誤検知を起こしてしまうという課題がある．

4 提案手法

既存手法の課題はパラメタ値に現れる文字そのものを用いて正常モデルを作成することに由来していた。これを解消するためにはパラメタ値の文字列についてより高いレベルの抽象化が必要である．そこで，提案手法ではパラメタ値を異なる文字クラスに属す複数の部分文字列からなる構造体と見なしてモデル化することで，既存手法の課題を解消し，誤検知を減少させることを目標とする．

4.1 学習

提案手法の学習プロセスは以下の 3 つのステップで構成され，全てのパラメタそれぞれについての正常モデルを作成する．

Step 1 対象パラメタのパラメタ値それぞれについて文字クラス列に変換する．文字クラスとは文字列の型である．提案手法では文字クラスとして，アルファベット，数字，記号のような基本的なものに加え，URL，メールアドレスのような複雑な構造をもつものも定義している．提案手法はパラメタ値を文字クラス列への変換をするために，各文字クラスの正規表現を定義している．表 1 に文字クラスの正規表現の一部を示す．例

表 1: 文字クラスの正規表現の例

String Class	Regex Expression
NU (numeric)	[0-9]+
AL (alphabet)	[a-zA-Z]+
SP (space)	[ ]+
DO (dot)	[.]+
SY (symbol)	[?!#$%&=]+

えばパラメタ値 “1.png”が与えられた場合，

“1.png” → (1, ., png) → (NU, DO, AL)

のようにパラメタ値の先頭から順に各文字クラスの正規表現にマッチする最長の文字列を見つけ出し，それをパラメタ値の末尾まで順次繰り返して 1 つの文字クラス列を得る．

全ての学習データについて上記の処理を行い，各パラメタについて複数の文字クラス列 (文字クラス列集合) を生成する．

Step 2 学習データには実際のトラフィックを利用することが一般的であり，その中には攻撃を意図したものが含まれている可能性がある．そこで，対象パラメタについて Step1 で得た文字クラス列集合の中から，その出現確率が閾値 $\alpha(0 \leq \alpha \leq 1)$ 未満の文字クラス列を除外する．

Step 3 パラメタの中にはコメント欄のようにその値の形式が定まっていない自由度の高いものも存在する．そのようなパラメタについて文字クラス列のように制約の強い形式を当てはめると多数の誤検知が生じると想定される．そこで，対象とするパラメタの正常モデルに用いる特徴として文字クラス列集合と文字クラス集合のいずれを用いるかを選択する．文字クラス集合とは文字クラス列集合に出現する文字クラスを抽出して重複を排除した集合であり，文字クラス列のような順序性を保持しない．具体的には，対象のパラメタについて学習データとして与えられたパラメタ値の個数を N_p とし，Step2 までに得られたユニークな文字クラス列の個数を U_p とし，

$$R = \frac{U_p}{N_p}$$

で定義される圧縮度 $R(0 \leq R \leq 1)$ を求め，閾値 β と比較する．$R < \beta$ の場合，当該パラメタは文字クラス列の出現順について規則性を有すると見なし，文字クラス列集合を正常モデルとする．$R \geq \beta$ の場合，当該パラメタは文字クラスの出現順について規則性をもたないものと見なし，文字クラス集合を正常モデルとする．例えばあるパラ

メタが Step2 により，

“First commit” → (AL, SP, AL)
“config.php” → (AL, DO, AL)
“Version update” → (AL, SP, AL)

のように 3 つの文字クラス列から文字クラス列集合が作成されたとした場合，圧縮度は $R = \frac{2}{3} = 0.66$ となる．ここで，閾値 β が 0.5 だった場合，$R > \beta$ となるため，上記の文字クラス列集合に現れるユニークな文字クラスを集めて得られる以下の文字クラス集合をこのパラメタの正常モデルとする．

{AL, SP, DO}

4.2 検知

検知ではまず，学習と同様の方法で検査対象のパラメタ値を文字クラス列へ変換する．正常モデルが 4.1 節の Step3 で示した文字クラス列集合である場合，そこに含まれる文字クラス列それぞれについて，検査対象の文字クラス列との LCS(最長共通部分列) [5] を求め，更にそれぞれの類似度 S を計算する．類似度 S は 2 つの文字クラス列を C_x，C_y とすると，以下の式で定義される．

$$S = \frac{|LCS(C_x, C_y)|}{|C_x| + |C_y| - |LCS(C_x, C_y)|}$$

例えば 2 つの文字クラス列 (AL, NU, DO, AL) と (SY, AL, SY, AL) の類似度は $\frac{2}{4+4-2} = 0.33$ となる．次に，得られた複数の類似度 S の中で最大の値 S_{max} と閾値 $\gamma (0 \leq \gamma \leq 1)$ を比較し，$S < \gamma$ の場合，パラメタ値は正常モデルとの類似度が低いため，攻撃と判定する．正常モデルが文字クラス集合である場合，正常モデルの文字クラス集合を E_m，検査対象の文字クラス列から Step3 で得られる文字クラス集合を E_t とすると

$$E_t \subseteq E_m$$

である場合，正常とし，そうでなければ攻撃と判定する．

5 実験

5.1 データセット

著者らが外部に公開している複数の Web サーバで受信した HTTP リクエストからクエリ部を含む URL 部のみを抽出し，以下の処理を行ったものをデータセットとして用いた．まず，サーバ毎に，データをアクセス日時で昇順にソートしてから 2 分割し，前半を学習用に，後半を検査用に用いることとした．学習用データからは，HTTP レスポンスコードが 404 及び 500 であるものを除いた．このような HTTP リクエストは脆弱な Web アプリケーションが稼働していないかを調査するスキャンの可能性が高く，正常なデータとして学習すべきものではないと判断したためである．検査用データについては，各手法による検知結果の正誤を判定するため，攻撃か否かを示すラベルを付与した．具体的な方法としては，複数の WAF 製品によって攻撃か否かを判断した結果を元に，一部手動でのチューニングを行った．なお，検査用データには，SQL Injection，RFI，XSS などの攻撃が含まれていることを確認した．

検査用データには学習用データで現れなかったパラメタ名も存在した．このようなパラメタについては学習時に正常モデルが作成されないことになり，検査時の扱い方には検討の余地があるが，今回の実験では全ての手法においてそのようなパラメタ値は正常と判定することとした．

5.2 環境

実験に際して、提案手法及び既存手法の実装を Python で行った．また実験に使用したマシンのスペックは 1.3Ghz Intel Core i5 CPU，8GB RAM, 512GB SSD であった．

6 評価

既存手法と提案手法について検知率 (Detection Rate)，誤検知率 (False Positive Rate) 及

び処理時間において比較した．検知率，誤検知率はそれぞれ以下のように定義する．

$$\text{検知率} = \frac{\text{攻撃データを攻撃として検知した数}}{\text{攻撃データ数}}$$

$$\text{誤検知率} = \frac{\text{正常データを攻撃として検知した数}}{\text{正常データ数}}$$

6.1 検知精度

各 Web サーバにおける各手法での検知結果を表 2 に示す．既存手法は複数の特徴を用いる手法であるが，ここでは文字列構造の特徴を用いる部分のみを用いた検知結果との比較を行っている．既存手法では正常モデルとの完全一致により攻撃か否かを判定するため閾値が存在しないが，提案手法では予備実験により閾値を選定し，それぞれ $\alpha = 0.001$，$\beta = 0.5$，$\gamma = 0.1$ を用いている．表 2 で示される Average はサーバごとのデータ数の偏りを考慮し，各サーバの検知率，誤検知率に対して平均をとった値である．平均値を見ると既存手法に比べ，提案手法は検知率を同程度に維持しながら，誤検知率を大幅に減少できていることが分かる．しかし，サーバ別に見ると，提案手法ではサーバ E, J, K, L において検知率が著しく低い．これらのサーバの学習データを分析したところ，学習時に除外すべき攻撃データが大量に含まれていた．これらを正常モデルとして学習してしまったことが検知率を低下させた原因と考えられる．

図 3 に Kruegel らの手法の文字列長，文字列分布，文字列構造，文字列の列挙性の 4 つの特徴を用いた場合と提案手法の ROC 曲線 [6] を示す．誤検知率の許容範囲は 10%までとしている．図 3 より誤検知率が 6%までは提案手法の方が検知率が高いことが分かる．すなわち，誤検知率の許容範囲を 6%以下にした場合，提案手法の方がより良い検知率を達成できると言える．

6.2 処理時間

図 4 と図 5 に，それぞれ学習と検知に要した時間を示す．横軸は処理データ数 (HTTP リクエスト数) であり，縦軸は処理時間 (秒) である．提案手法は学習と検知の双方において既存手法より処理時間が長い．これは各パラメタ値を文字クラス列に変換する部分で既存手法より多くの時間を要しているためである．処理データ数が増加すると，処理時間が増加する傾向にあるのは既存手法も同様であるが，提案手法ではその関係が線形ではなく，極端に多くの時間を要している外れ値が存在する．これは提案手法の処理時間は処理データ数のみでなく，パラメタ値の複雑性に大きく影響を受けるためである．

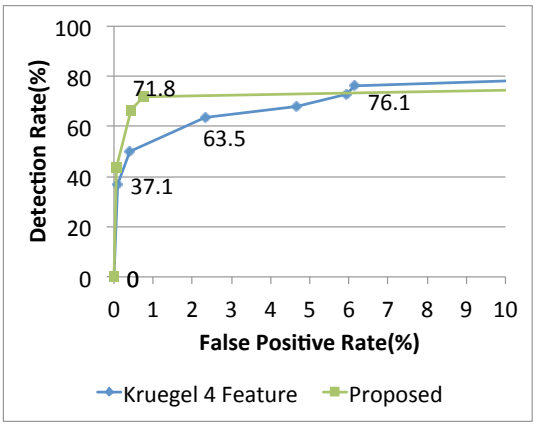

図 3: ROC 曲線（$\alpha = 0.001$，$\beta = 0.5$）

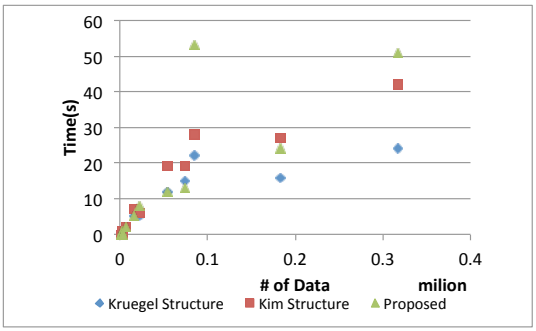

図 4: 学習時の処理時間

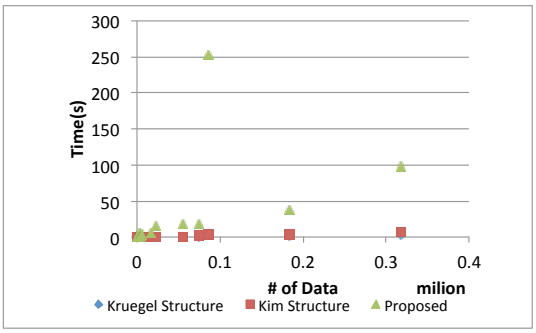

図 5: 検知時の処理時間

表 2: 各サーバに対する攻撃検知精度

Server	Detection Rate (%)			# of Attack	False Positive Rate (%)			# of Normal
	Kruegel	Kim	Proposed		Kruegel	Kim	Proposed	
A	100.0	100.0	97.56	41	0.28	0.28	0.28	725
B	100.0	100.0	100.0	93	8.86	9.28	**2.35**	722
C	97.01	97.01	95.52	67	19.91	19.91	**3.24**	2406
D	93.26	100.0	82.02	89	3.58	7.69	**0.05**	4081
E	94.23	94.23	42.31	52	7.9	11.09	**0.02**	6051
F	85.23	85.23	**100.0**	149	0.24	4.23	**0.13**	16467
G	30.48	35.14	**90.54**	666	8.18	21.16	**0.91**	21393
H	91.49	94.33	94.33	141	5.28	27.64	**0.15**	54840
I	95.13	95.7	85.92	2834	2.24	6.74	**0.56**	72136
J	42.03	42.03	25.36	138	1.14	1.75	**0.91**	85422
K	51.8	51.8	30.63	222	0.25	6.32	**0.21**	183270
L	8.78	12.84	**18.24**	148	1.47	1.59	**0.01**	317118
Average	74.12	75.69	71.87		4.94	9.81	**0.74**	

6.3 閾値分析

提案手法では学習時の Step2 において，出現率が少ない文字クラス列を除外するために利用する閾値 α，学習時の Step3 において，パラメタの文字クラス列の出現順について規則性を有するかどうかを判定する閾値 β 及び検知時にあるパラメタとその正常モデルとの類似度に応じて攻撃かどうかを判定する閾値 γ という 3 つの閾値が存在する．それぞれの閾値について値を変化させた場合，検知率と誤検知率がどのように変化するかの分析を行った (図 6，図 7，図 8)．各図の左側の縦軸は検知率のスケールを示しており，右側は誤検知率のスケールを示している．図 6 より α と検知率，誤検知率は正の相関関係があることが分かる．この原因は α を大きく取った場合，各パラメタに対して学習時に含まれる攻撃データを除外すると供に，正常モデルまでを除外してしまうことにある．また，α の値が小さい場合は誤検知率より検知率の増加の方が大きいが，α の値が大きい場合，誤検知率の増加の方が大きくなるため，α は 0 に近い値に設定する必要がある．

図 7 により $\beta > 0.2$ の部分では検知率，誤検知率は大きく変化しないことが分かる．図 9 にパラメタ値の複雑性を現す圧縮度 R の頻度分布を示す．横軸は R の値を 0.1 ずつ区切り，各区間の最大値を示しており，縦軸はその区間に含まれる学習時のパラメタの個数を示している．パラメタの圧縮度 R が区間 $(0.9, 1.0]$ に含まれるパラメタは全て $R = 1$ であり，この原因はそのパラメタに対して学習データが 1 つしか存在しなかったためである．また，このようなパラメタ値の数は全体から見て 1%以下とごく少数であるため，今回使用した学習データではほぼ全てのパラメタの圧縮度は 0.5 以下に存在しており，コメント欄のようにその値の形式が定まっていない自由度の高いパラメタの存在が少数だったと言える．そのため，β を 0.2 より大きく変化させても検知率，誤検知率にほとんど変化が見られない結果となった．

図 8 より，γ と検知率，誤検知率は負の相関関係にあり，γ の増加に対して誤検知率より検知率の方が減少が著しい．そのため，α と同様，0 に近いの値に設定する必要がある．

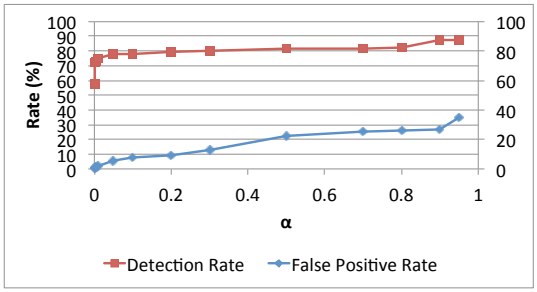

図 6: 閾値 α の変化 ($\beta = 0.5$，$\gamma = 0.1$)

7 おわりに

本稿では Web アプリケーションのパラメタを悪用する攻撃を検知する手法としてアノマリ

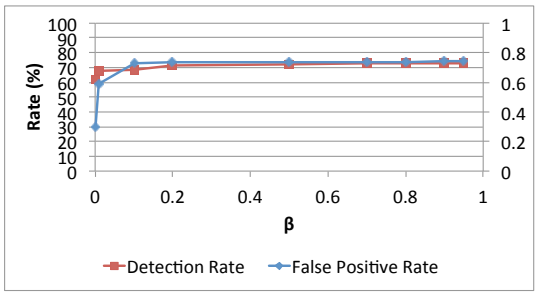

図 7: 閾値 β の変化 ($\alpha = 0.001$，$\gamma = 0.1$)

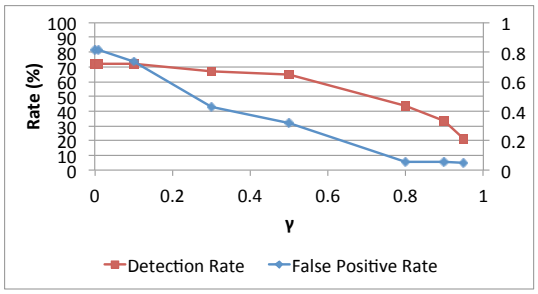

図 8: 閾値 γ の変化 ($\alpha = 0.001$，$\beta = 0.5$)

検知を用いた新たな手法を提案した．既存手法ではWebアプリケーションのパラメタ値そのものを特徴として用いるため，学習データと同様の形式であるが値が完全には一致しないような正常リクエストを攻撃として誤検知してしまう問題があった．この問題を解消するために著者らは，パラメタ値として与えられる文字列を抽象化し，文字クラス列という複合的な型構造で捉え，正常モデルに用いる手法を提案した．また，実データを用いた実験によって，狙い通りに誤検知を抑制する効果を示した．今後の主な課題としては，データの複雑性に大きく影響されてしまう処理時間の改善があると考えている．

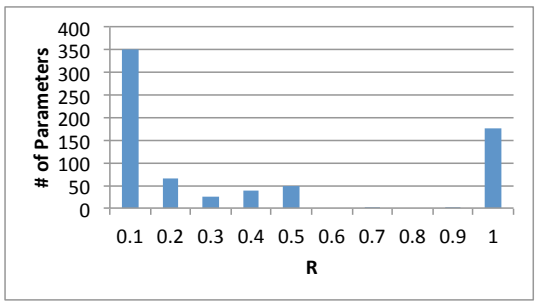

図 9: 全パラメタに対する圧縮度 R の分布

参考文献

[1] Kruegel Christopher and Giovanni Vigna. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security*, 2003.

[2] Tae Hyung Kim, Kyungtae Kim, Jong Kim, and Sung Je Hong. Profile-based web application security system with positive model selection. In *Proceedings of the 2nd Joint Workship on Information Security*, 2007.

[3] Andreas Stolcke and Stephen Omohundro. Hidden markov model induction by bayesian model merging. In *Advances in neural information processing systems*, 1993.

[4] Kevin J Lang, Barak A Pearlmutter, and Rodney A Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. *Grammatical Inference*, 1998.

[5] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, 2000.

[6] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. In *Hanley, James A and McNeil, Barbara J*, 1982.