情報処理学会論文誌 Vol.60 No.12 2223–2233 (Dec. 2019)

文字列構造に着目したWebアプリケーションに対する攻撃のアノマリ検知手法

鐘本 楊[1,2,a)] 青木 一史[1] 三好 潤[1] 嶋田 創[3] 高倉 弘喜[4]

受付日 2019年3月11日, 採録日 2019年9月11日

概要：Web アプリケーションの脆弱性を悪用する攻撃は絶えず行われており，攻撃の中でも未知攻撃の検知はいまだに重要な問題である．未知攻撃を発見する手法として，アノマリ検知を用いるのが一般的だが，誤検知が多いという課題が存在する．本研究では HTTP リクエストの各要素の文字列の構造に着目し，学習を行うことで，誤検知を低減することを目指す．評価の結果，既存のアノマリ検知手法に比べ提案手法の方が精度が高く，また性能も実用的な処理速度であることを示す．

キーワード：Web セキュリティ，アノマリ検知，HTTP

Detecting Malicious Inputs of Web Application Using Character Structure Profiling

Yo Kanemoto[1,2,a)] Kazufumi Aoki[1] Jun Miyoshi[1] Hajime Shimada[3] Hiroki Takakura[4]

Received: March 11, 2019, Accepted: September 11, 2019

Abstract: Web attacks that exploit web application's vulnerabilities are still occurring and detecting unknown attacks is still major problems. Anomaly detection is effective for detecting unknown attacks, but it also suffers from false positives. To reduce false positives, we propose a novel anomaly detection method which creates abstract structure profile for each HTTP request component. The evaluation results show that our approach is superior than existing methods, and performance is practical.

Keywords: Web security, anomaly detection, HTTP

1. はじめに

Web アプリケーションは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．同時に Web アプリケーションに対する攻撃は日々継続的に行われている．Positive Technologies のレポート[*1]によると IT 企業，銀行，政府の Web サイトに対して 1 日約 1,000 件にも上る攻撃が行われている．

攻撃には既知攻撃もあれば，新たな脆弱性に対する未知攻撃の可能性も存在する．本稿でいう既知攻撃とは過去に観測されたことのある攻撃であり，未知攻撃とは未発見の脆弱性への攻撃や過去に観測されたことのない攻撃のことと定義する．**表 1** は脆弱性情報データベースである CVE[*2]から収集した 2014 年から 2018 年まで毎年発見された脆弱性の件数とそのうち Web アプリケーションに関する脆弱性の件数である．毎年発見される Web アプリケーションに関する脆弱性の件数は依然として減少してい

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories, Musashino, Tokyo 180–8585, Japan
[2] 京都大学大学院情報学研究科
Graduate School of Informatics, Kyoto University, Kyoto 606–8501, Japan
[3] 名古屋大学情報基盤センター
Information Technology Center, Nagoya University, Nagoya, Aichi 464–8601, Japan
[4] 国立情報学研究所アーキテクチャ科学研究系
Information Systems Architecture Science, National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] yo.kanemoto.zx@hco.ntt.co.jp

[*1] https://www.ptsecurity.com/upload/corporate/ww-en/analytics/Web-application-attacks-2018-eng.pdf
[*2] https://cve.mitre.org

表 1 2014 年から 2018 年の間に発見された脆弱性の件数と Web アプケーションに関する脆弱性の件数
Table 1 # of vulnerability discovered between 2014 and 2018.

年	Web 関連	すべて	Web 関連の割合（%）
2014	2,148	7,937	27.06%
2015	1,784	6,487	27.50%
2016	1,228	6,447	19.05%
2017	2,308	14,649	15.76%
2018	2,133	15,998	13.33%

ない．そのため，新たな脆弱性を悪用する未知攻撃は継続的に発生する状況であると考える．

攻撃に対する検知方法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る．シグネチャ検知は既知攻撃のバイト列から特徴的な部分を正規表現あるいはバイト列で表現したシグネチャをあらかじめ用意し，検知対象である 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，難読化された攻撃コードの検知や，あらかじめシグネチャを用意しておくことができない未知攻撃の検知には期待できないことが課題である．そのため，難読化された攻撃コードや未知攻撃をとらえるためにはアノマリ検知が必要である．

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であるプロファイルを作成しておき，検知対象である HTTP リクエストがプロファイ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，たとえ攻撃コードが既知のものと異なったとしても検知することが可能であり，未知攻撃も検出できる可能性がある．Web アプリケーションの URL パラメタを悪用する攻撃に対するアノマリ検知手法に Kruegel らの手法 [1], [2] や Kim らの手法 [3] がある．Kruegel らの手法では HTTP リクエストの URL 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，プロファイルとする手法である．Kim らの手法では Kruegel らの手法と同様の特徴に加えてプロファイル選択の概念を導入し，プロファイルの種類をパラメタごとに使い分けることで精度を改良しようとしている．URL パラメタだけでなく，HTTP リクエスト全体を特徴量として扱う手法には Wang らの手法（Anagram）[5] や Ingham らの手法 [4]，Luo らの手法 [6] や Betarte らの手法 [7] があげられる．

これらの手法ではいずれも HTTP リクエストに出現する文字列そのものを特徴として用いるため，プロファイルを作成する際に与えられた学習データが網羅的でないと，正常な入力値を攻撃として検知（誤検知）してしまう．Kim らの手法や Ingham らの手法ではあらかじめ条件を定めた文字列に関しては正規化（抽象化）されるが，条件に当てはまらない場合，文字列そのものが特徴量として用いられる．あらかじめ条件を定めた文字列とはたとえば，URL やファイルパス，e メールアドレスなど構造が定まっている文字列のことである．

本研究では既存手法と異なり，すべての文字列を対象として正規化を施すことで，より網羅的なプロファイルを作成し，誤検知の低下を目指す．具体的には文字列の型構造を表現した文字クラス列および文字クラス集合を特徴として導入する．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，誤検知率を減少できることを示す．

本研究の貢献は以下のとおりである．

- 文字列をより抽象的に表現する方法を考案し，その方法用いたアノマリ検知手法を提案した．
- 約 134 万件の実データに対して評価を行い，既存手法より高精度であることを示した．
- リクエスト 1 件あたり 1 ミリ秒以内で処理可能であることを示し，実用性が高いことを示した．

2. 既存手法とその課題

本研究で注目する Web アプリケーションに対するアノマリ検知では，HTTP プロトコルの各要素を解釈したうえで，アノマリ検知を適用することが一般である．具体的には HTTP リクエストの URL パス，URL パラメタ，ヘッダ，Cookie など，HTTP リクエストを構成する要素を分解したうえで学習時は各要素ごとにプロファイルを作成し，検知時は各要素ごとに異常か否かの判定を行う．

Web アプリケーションの URL パラメタを悪用する攻撃に対してアノマリ検知を試みた手法に Kruegel らの手法 [1] や Kim らの手法 [3] が存在する．Kruegel らの手法 [1] では，URL パラメタのみを対象として，各 URL パラメタごとにプロファイルを作成する．プロファイルは文字列長，文字分布，文字列構造などの特徴から成る．これらの特徴から正常との乖離度合いを算出して，閾値以上である場合，攻撃として検知する．Kim らの手法 [3] では Kruegel らの手法と同じ特徴を用いるが，プロファイルごとに用いる特徴を選択するところが異なる．

Web アプリケーションの URL パラメタに着目するのではなく，HTTP リクエスト全体に着目した手法に Anagram [5]，Ingham らの手法 [4]，Luo らの手法 [6] や Betarte らの手法 [7] が存在する．Anagram では学習時に HTTP リクエスト全体の文字列に対して n-gram を求め，保持する．検知時は同様に HTTP リクエストに対して n-gram を求め，各 n-gram の出現割合によって攻撃か否かを判定手法である．Ingham らの手法ではまず，トークンと呼ばれる HTTP リクエストに表れる特徴的な表現を定義し，HTTP リクエストに対してトークン列を作成する．たとえば，`GET/HTTP/1.1` のような HTTP リクエストがあった場合，トークン列は（`GET`, `/`, `HTTP/1.1`）となる．学習時は HTTP リクエストから作成したトークン列から各トークンの遷移モデルを作成する．上記の例の場合 `GET` → `/`,

/ → HTTP/1.1 という遷移モデルが作成される．検知時は検知対象の HTTP リクエストのトークン列と学習した遷移モデルの乖離度合いから攻撃を検知することを試みている．Luo らの手法では HTTP リクエストをバイト値に基づいて特徴ベクトル化する．次に並列的に学習器を 1 つずつ順番に構築していく XGBoost [8] を利用して学習モデルを作成し，そのモデルとの乖離をもとに攻撃を検知する．Betarte らの手法ではあらかじめ用意した特徴的な文字列を用意し，HTTP リクエストにその文字列が含まれる数を計算し，特徴ベクトル化する．特徴ベクトルに対して，混合ガウスモデルを利用して学習モデルを作成し，そのモデルとの乖離をもとに攻撃を検知する．

特徴のとらえ方はいずれの既存手法でも異なるが，学習の際に入力値の文字列そのものを用いている点では共通している．Kruegel らの手法 [1] や Kim らの手法 [3] では文字列の構造を表す際に，文字列そのものの遷移モデルを利用している．たとえば，文字列 abc という入力があった場合，a → b，b → c という遷移が学習される．Anagram [5] や Ingham らの手法 [4] では特徴に文字列の n-gram やトークンを用いているが，上記の手法と同様に文字列そのものが特徴となっている．また，Luo らの手法 [6] や Betarte らの手法 [7] では HTTP リクエストに現れる文字列を特徴ベクトル化して特徴量として扱っている．このように，既存手法はいずれも学習時に大量のデータを与えることが可能な場合，あるいは Web アプリケーション自体が静的コンテンツのみで構成されるような単純な仕組みの場合には適用できるが，動的な Web アプリケーションにおいては誤検知が増加してしまうという課題が存在する [9]．

3. 提案手法

既存手法の課題は HTTP リクエストの各要素の入力値に現れる文字列そのものを用いてプロファイルを作成することに由来していた．提案手法では入力された文字列すべてを正規化することでこの問題を解消することを目指す．具体的には提案手法では入力値の文字列を文字種類ごとに定義した文字クラスへの変換を行い，文字列の構造を表現するプロファイルを作成する．

提案手法の概要を**図 1** に示す．提案手法は学習と検知の 2 つの処理に大別される．学習時は正常な HTTP リクエストを入力とし，HTTP リクエストを要素ごとに分解し，要素ごとにプロファイルを作成する．検知時は学習時と同様に入力された HTTP リクエストを要素に分解し，要素ごとに学習したプロファイルと照合し，乖離度を求める．乖離度があらかじめ指定した閾値を超えていれば攻撃と判定し，そうでなければ正常と判定する．以下，図 1 の各ステップで示す学習と検知の詳細について，各節で述べる．

3.1 学習

提案手法の学習プロセスは以下の 4 つのステップで構成される．

Step T1：Parsing 入力された HTTP リクエストを URL パス，URL パラメタの Key と Value，ヘッダの Key と Value，Cookie の Key と Value の要素ごとに分解する．たとえば，以下の HTTP リクエストがあった場合，

```
GET /index.php?id=1&name=alice HTTP/1.1
Host: example.net
User-Agent: Mozilla/5.0
Cookie: PHPSESSID=1234;
```

URL パスは /index.php，URL パラメタは {id: 1, name: alice} といった Key-Value 形式，ヘッダは {Host: example.net, User-Agent: Mozilla/5.0} といった Key-

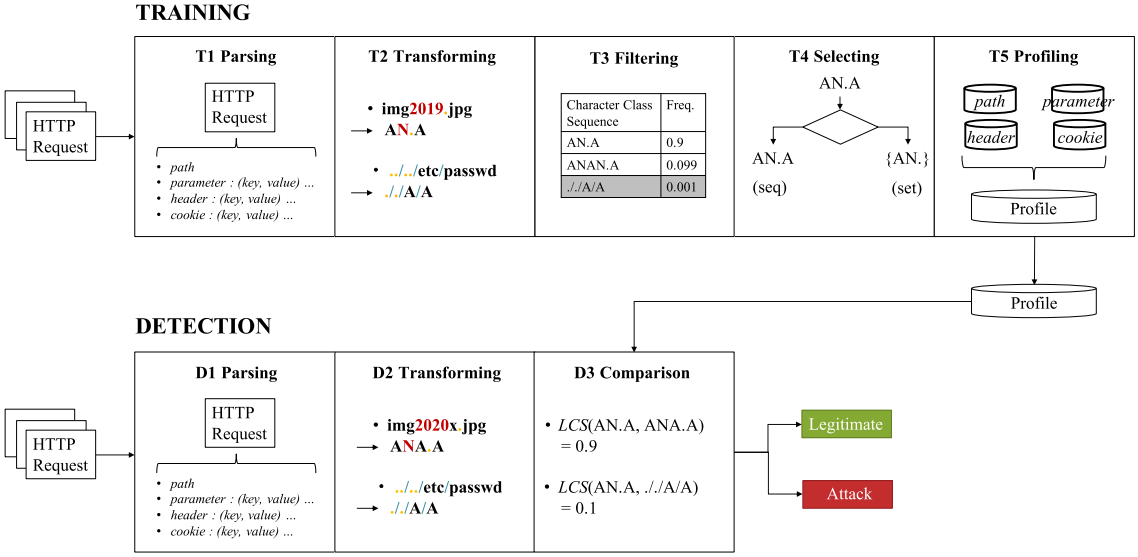

図 1 提案手法の処理概要

Fig. 1 Overview of proposed method.

表 2 文字クラスの定義例
Table 2 Example definition of character class.

文字クラス	説明	例
A	アルファベット	{a, ..., z, A, ..., Z}
N	数字	{0, 1, ..., 9}
T	空白およびタブ	{␣, \t}
Q	クオート	{', "}
B	括弧	{(,), ...}
M	マルチバイト文字	{\x0100, ... }
.	ピリオド	{.}
/	スラッシュ	{/}

Value 形式，Cookie は {PHPSESSID: 1234} といった Key-Value 形式に分解される．

Step T2：Transforming 各要素の入力値の文字列を文字クラス列に変換する．文字クラスとは文字列の型を意味しており，同じ文字種の文字は同じクラスに属す．提案手法では**表 2** に例示するような文字クラスを定義している．たとえば，文字列 img2019.jpg が入力値である場合，以下のような文字クラス列に変換される．

img2019.jpg → (img, 2019, ., jpg) → AN.A

文字列を文字クラス列に変換する仕組みを Algorithm 1 に示す．まず，説明に用いる記号を定義する．ccmap はあらかじめ与えられる文字クラスの定義である．x[0:i] は文字列 x の 0 文字目から i 文字目の部分文字列を意味している．len(x) は x の長さを返す関数である．変換処理は 3 つの関数からなり，まずこの 3 つの関数 PREFIX-MATCH，GET-BEST-CANDIDATE，GET-CC-SEQ を定義する．関数 PREFIX-MATCH は入力文字列とある文字クラスを引数として，文字列を先頭から順にその文字クラスに包含される最長の文字列を見つけ出す処理を行う．関数 GET-BEST-CANDIDATE はあらかじめ定義されているすべての文字クラスに対して関数 PREFIX-MATCH を実施し，関数 PREFIX-MATCH の結果が最長となる文字クラスを選ぶ処理を行う．関数 GET-CC-SEQ は関数 GET-BEST-CANDIDATE を入力値の文字列に対して複数回適用し，入力値の文字クラス列を得る処理を行う．

Step T3：Filtering アノマリ検知において，学習データには実際の通信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．攻撃であるデータを学習してしまうことは検知漏れを発生させる原因となってしまう．そこで，前ステップである要素について得た複数の文字クラス列に対してヒストグラムを作成し，ある文字クラス列の出現確率がきわめて低い場合，これを除外する処理を行う．これは一般的な Web サイトでは実際の通信に含まれる攻撃は正常な通信に対してきわめて少ない状況であることに基づいている．具体的にはある文字クラス列の出現確率が閾値 α（$0 \leq \alpha \leq 1$）未満の場合，これを除外する．たとえば，ある URL パラメタの入力値が img2019.jpg，img2020.png，../../etc/passwd の 3 つである場合，文字クラス AN.A の出現確率は $\frac{2}{3} = 0.66$，文字クラス ./../A/A の出現確率は $\frac{1}{3} = 0.33$ となる．閾値 α を 0.4 とした場合，文字クラス ./../A/A が除外される．この処理により学習時にノイズとなるような攻撃データを学習しないようにし，検知漏れの発生を防ぐ．

Step T4：Selecting 入力値の中にはブログ記事のタイトル名やコメント欄のように，形式が定まっていない自由度の高い入力値も存在する．そのような入力値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誤検知が生じてしまう原因となる．提案手法ではさらに文字クラス集合という概念を導入する．文字クラス集合とは文字クラス列に出現する文字クラスを抽出して重複を排除した

Algorithm 1 Character Class Sequence Generation

```
Require:
    ccmap = {A: abcd, N: 0123, ...}
 1: function PREFIX-MATCH(x, cc)
 2:     for i, y in x do
 3:         if y not in cc then
 4:             return i, x[0:i]
 5:         end if
 6:     end for
 7:     return len(x), x
 8: end function
 9:
10: function GET-BEST-CANDIDATE(cands)
11:     bc, bl = null, 0
12:     for c, l in cands do
13:         if l > bl then
14:             bc, bl = c, l
15:         end if
16:     end for
17:     return bc, bl
18: end function
19:
20: function GET-CC-SEQ(x)
21:     ccseq = []
22:     while len(x) > 0 do
23:         cands = []
24:         for c, cc in ccmap do
25:             l, z = PREFIX-MATCH(x, cc)
26:             if l > 0 then
27:                 cands += [(c, l)]
28:             end if
29:         end for
30:         bc, bl = GET-BEST-CANDIDATE(cands)
31:         ccseq += [bc]
32:         if bl == 0 then
33:             x = x[bl+1:]
34:         else
35:             x = x[bl:]
36:         end if
37:     end while
38:     return ccseq
39: end function
```

集合であり，文字クラス列のように順序性を保持しない．提案手法ではプロファイルの特徴として文字クラス列と文字クラス集合のいずれを用いるかを選択する．自由度の高い入力値の場合，順序に制約を持たない文字クラス集合を適用し，型が決まった入力値の場合，順序に制約を持つ文字クラス列を適用することで，誤検知を低下させつつ，検知漏れが発生することを防ぐ．具体的には，ある要素に対して，前ステップまでに得られた文字クラス列の集合に含まれるユニークな文字クラス列の個数を n とし，閾値 β と比較する．$n < \beta$ の場合，入力値は規則性を有すると見なし，すべての入力値から得られた文字クラス列をプロファイルとする．$n \geq \beta$ の場合，入力値は規則性を持たないものと見なし，得られた文字クラス列から文字クラス集合を求め，プロファイルとする．たとえば，入力値から得られた複数の文字クラス列が (`AN.A`, `ANA.A`, `A/A.A`) とすると，文字クラス集合は {`AN./`} となる．

Step T5：Profiling 提案手法では Key-Value 形式でプロファイルを作成する．プロファイルの Key は HTTP リクエストの各要素の Key に相当する．URL パラメタやヘッダ，Cookie などはすでに Key-Value の形式になっているため，各要素の Key をそのままプロファイルの Key として扱う．また，URL パスや各要素の Key の部分に攻撃コードが挿入される可能性もあるため，URL パス，URL パラメタの Key，ヘッダの Key，Cookie の Key の値を Value としてとらえ，それぞれ個別のプロファイルを作成する．よって上記の例の HTTP リクエストによって作成されるプロファイル全体 P は以下のようになる．$p(x)$ は入力値 x で作成されたプロファイルを返す関数とする．

$$P = \begin{cases} \text{path} & = p(\texttt{/index.php}) \\ \text{parameter key} & = p(\texttt{id}, \texttt{name}) \\ \text{header key} & = p(\texttt{Host}, \texttt{User-Agent}) \\ \text{cookie key} & = p(\texttt{PHPSESSID}) \\ \text{parameters} & = \begin{cases} \texttt{id} & = p(\texttt{1}) \\ \texttt{name} & = p(\texttt{alice}) \end{cases} \\ \text{headers} & = \begin{cases} \texttt{Host} & = p(\texttt{example.net}) \\ \texttt{User-Agent} & = p(\texttt{Mozilla/5.0}) \end{cases} \\ \text{cookies} & = \{\texttt{PHPSESSID} = p(\texttt{1234}) \end{cases}$$

3.2 検知

検知には 3 つのステップがあるが，最初の 2 つのステップ **D1**，**D2** は学習時のステップ **T1**，**T2** と同じである．つまり，学習時と同様に検知対象の入力値を文字クラス列へ変換する．その後，**Step D3 Comparison** にてその文字クラス列とプロファイルの類似度を求める．類似度の求め方は学習したプロファイルの特徴の種別によって異なる．

プロファイルの特徴が文字クラス列の場合 プロファイルに含まれる n 個のユニークな文字クラス列，それぞれについて検知対象の文字クラス列との最長共通部分列 (LCS) [10] を求め，類似度を算出する．類似度 s は 2 つの文字クラス列を x，y とすると，以下の式で定義される．

$$s = \frac{|LCS(x,y)|}{|x| + |y| - |LCS(x,y)|}$$

たとえば 2 つの文字クラス列 `AN.A` と `ANA.A` の類似度は $\frac{4}{4+5-4} = 0.8$ となる．算出した類似度の集合を $S = \{s_1, s_2, \ldots, s_n\}$ とする．次に，得られた複数の類似度 S の中で最大の値 $max(S)$ を求め，アノマリ値 $a = 1 - max(S)$ と閾値 γ ($0 \leq \gamma \leq 1$) を比較する．$a \geq \gamma$ の場合，プロファイルとの乖離が大きいため，攻撃と判定する．$a < \gamma$ の場合，プロファイルと類似しているため，正常と判定する．

プロファイルの特徴が文字クラス集合である場合 プロファイルの文字クラス集合を E_p，検知対象の文字クラス列から得られる文字クラス集合を E_t とすると

$$E_t \subseteq E_p$$

である場合，正常と判定し，そうでなければ攻撃と判定する．文字クラス集合は文字クラス列に比べ，制約条件が少ないため，検知漏れを発生させる可能性が高い．そのため，学習に出現しない文字クラスが 1 つでも発生した場合に攻撃として検知するという厳しい条件を採用している．

4. 評価

提案手法を実装し，実データセットを利用した評価を行った．データセット，実験環境，評価指標および評価結果を各節にて述べる．

4.1 データセット

著者らが外部に公開している複数の Web サーバで観測した HTTP リクエストをデータセットとして用いた．評価はホールドアウト法 [11] を用いて行った．Web サーバごとに学習データ数:検知データ数を 1:2 の割合で用意し，学習データから HTTP レスポンスのステータスコードが範囲 [200, 400) 以外となるデータおよび，攻撃とラベルづけされたデータを除外した．

学習データが検知データより少ない理由は Web サイトの性質を考慮したためである．Web サイトは更新や仕様変更が行われる可能性が高く，更新や仕様変更が行われた際はプロファイルなどを再学習する必要がある．そのため，学習データが多くない状態を想定した．評価にホールドアウト法を選択した理由は，実際の Web サイト更新や仕様変更をふまえた評価を行うためである．本来，Web サイトの更新や仕様変更があると検知精度は低下することが考えられる．クロスバリデーション法のようにデータ全体を分

表 3 データセットの概要
Table 3 Summary of dataset.

サイト名	学習データ数	検知データ数		
		攻撃数	正常数	合計
A	3,534	237	11,498	11,735
B	8,658	311	20,426	20,737
C	10,977	414	30,284	30,698
D	6,454	281	31,820	32,101
E	15,948	233	37,446	37,679
F	36,078	176	90,353	90,529
G	20,653	284	99,469	99,753
H	45,006	4,298	123,927	128,225
I	59,390	2,468	147,085	149,553
J	40,384	140	154,633	154,773
K	51,314	180	163,212	163,392
L	67,659	1,074	197,285	198,359
M	88,222	148	226,061	226,209
合計	454,277	10,244	1,333,499	1,343,743

割しながら検証を行う方式だと，Web サイトの更新や仕様変更という時系列変化を無視してしまうため，実際よりも検知精度が向上してしまう可能性がある．そこで，本評価では，学習データと検知データが時系列順になるようにした．ステータスコードによる除外を行った理由はこのような 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，学習すべき正常なデータではないと判断したためである．

学習データと検知データについては，攻撃か否かを示すラベルを付与した．ラベル付与では OSS の IDS/WAF である ModSecurity [*3] や Snort [12]，フリーで利用できる Emergence Threats [*4] のシグネチャによって攻撃か否かを判断した結果を元に，一部人手でチューニングを行った．

表 3 に評価に使用したデータセットの概要を示す．13 の Web サイトから合計 1,798,020 件の HTTP リクエストを収集し，454,277 件を学習データ，1,343,743 件を検知データとした．収集期間は 2017 年 1 月 1 日から 2017 年 1 月 31 日である．検知データに含まれる攻撃種別の割合は**図 2** に示すとおりである．凡例の SQLI は SQL インジェクション，PT はパストラバーサル，RCE はリモートコード実行，RFI はリモートファイルインクルージョンを表している．複数種類の攻撃が含まれており，評価に妥当なデータセットである．

検知データには学習時には出現しなかった Key が出現することがある．このような Key については学習時にプロファイルが作成されていない．検知時においてこのような未学習な Key に関する扱い方には検討の余地があるが，今回の実験では未学習の Key が発生した場合は異常と判定するには根拠が不足していると考え，正常であると判定することとした．たとえば，学習によって URL パラメタに `id`，`name` のプロファイルが作成されたとする．検知時に `page` という URL パラメタが出現した場合，異常と判定できないため正常であると判定することとする．

*3 https://www.modsecurity.org
*4 https://rules.emergingthreats.net

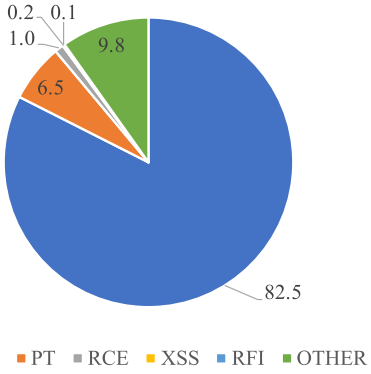

図 2 検知データに含まれる攻撃種別の割合（%）
Fig. 2 Histogram of attack type in detection dataset.

4.2 実験環境

評価に際して，提案手法および既存手法の実装を Python 言語で行った．実行に際しては Python インタープリタより高速な実行環境である PyPy [*5] を利用した．また実験に使用したマシンのスペックは 3.1 GHz Intel Core i5 CPU，16 GB RAM，1 TB SSD であった．

4.3 指標

既存手法と提案手法について検知率（True Positive Rate），誤検知率（False Positive Rate）および処理時間，それぞれの観点で比較した．検知率，誤検知率はそれぞれ以下のように定義した．

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

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

4.4 検知精度

図 3 に提案手法および既存手法の ROC 曲線 [13] を示す．ROC 曲線は横軸に誤検知率，縦軸に検知率をとったうえで，各手法における閾値を調整した際の誤検知率，検知率の点をプロットし，それを線で結合した曲線である．より左上にある曲線の方がより精度が高いことを表している．攻撃検知の分野では正常リクエストが攻撃リクエストより著しく多いことが一般的であるため，高い誤検知率は許容できない．そのため，本研究では誤検知率の許容範囲を 10%までとしている．

既存手法については論文をもとに著者らが実装を行った．Anagram については n-gram の値 n，Luo らの手法では XGBoost の学習繰返し回数の閾値 b，Betarte らの手法

*5 https://pypy.org

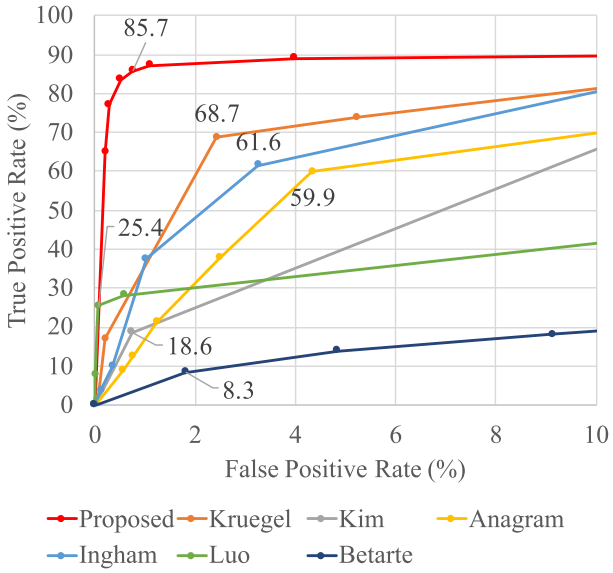

図 3 ROC 曲線
Fig. 3 ROC curve.

では初期クラスタ数 c，またすべての手法において検知時に攻撃として判定する閾値 t が検知精度に対して大きな変化をもたらすと考え，これらのパラメタを調整することで図 3 を求め，最適な検知精度を調査した．より具体的には，$n = [1, 10]$, $b = [10, 1000]$, $c = [1, 500]$ および $t = [0.0, 1.0]$ の範囲内で閾値を変化させた．Luo らの手法で利用する XGBoost は教師あり学習のため，Luo らの手法の評価時のみ学習データに含まれる攻撃ラベルのデータを除外せずに学習を行った．

検知率および誤検知率は Web サイト間で異なるため，Web サイト間のデータ量の差異を考慮し，ROC 曲線を描画する際には各 Web サイトの検知率および誤検知率の加重平均を用いた．つまり，各 Web サイトの検知率および誤検知率を求めた後，検知率の平均値および誤検知率の平均値を ROC 曲線の検知率，誤検知率とした．図 3 より既存手法に比べて提案手法はより低い誤検知率で高い検知率が達成できており，検知精度において既存手法より高い性能であることが示された．Luo らの手法および Betarte らの手法の検知率が特に低い理由は特徴量の取り方が問題であった．Luo らの手法では HTTP リクエスト全体のバイト値を特徴ベクトルとするため，URL 長の変化やヘッダ長の変化が大きいサイトに対しては特徴ベクトルの要素の次元にずれが発生するため高い精度を発揮できないと考える．Betarte らの手法ではあらかじめ用意した攻撃の特徴を示す文字列の出現数を特徴ベクトルとするため，攻撃にこれらの文字列が出現しない場合に検知率は大幅に低減してしまう．

Web サイトごとに検知率，誤検知率にはばらつきがある．**表 4** に Web サイト間の検知率/誤検知率の平均および標準偏差を示す．表 4 では ROC 曲線より得られる各手法にて最大の精度を発揮する閾値を設定した際の結果を用いている．提案手法の検知率の標準偏差（14.8）および誤検知率の標準偏差（0.25）は既存手法と比較して小さく，各 Web サイトに対して安定的に高い精度維持できることを示している．また，既存手法に比べ，閾値調整を繊細に行わなくても，一定の精度を発揮することが可能であることから，新しい Web サイトを検知対象に追加する際も運用者にとって手間が少ないという効果をもたらす．

表 4 Web サイト間の検知率/誤検知率の誤差
Table 4 Difference of true positive rate/false positive rate among web sites.

手法	検知率（%）		誤検知率（%）	
	平均	標準偏差	平均	標準偏差
Proposed	83.5	14.8	0.50	0.25
Kruegel	68.7	22.9	2.43	0.99
Kim	18.0	25.7	0.73	2.54
Angram	59.9	25.4	4.33	2.54
Ingham	60.3	27.8	3.25	2.47
Luo	25.4	31.6	0.07	0.11
Betarte	8.13	10.5	0.92	0.56

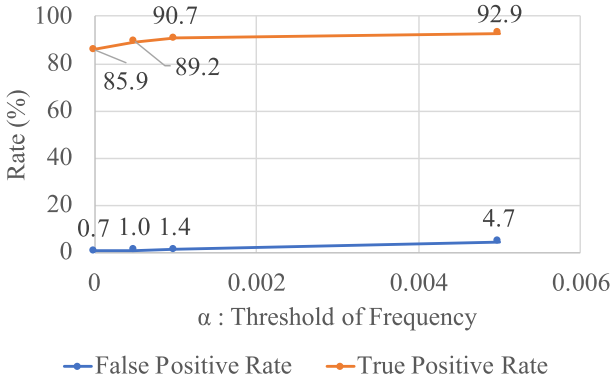

図 4 閾値 α の変化（$\beta = 10$, $\gamma = 0.5$）
Fig. 4 Variation of α ($\beta = 10, \gamma = 0.5$).

4.5 閾値分析

提案手法には学習時のステップ **T3** 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，学習時のステップ **T4** において，順序制約を持つ文字クラス列または順序制約を持たない文字クラス集合のどちらをプロファイルに用いるかを判定する閾値 β，および検知時のステップ **D3** に入力値とプロファイルとの類似度に対して攻撃か否かを判断する閾値 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の検知率と誤検知率の変化に対する分析を行った．

閾値 α に関する分析 閾値の変化にともなう検知率，誤検知率の変化を**図 4** に示す．横軸は閾値のとる値を表しており，縦軸は検知率/誤検知率を表している．α を大きくすると検知率，誤検知率はともに単調増加することが分かる．この原因は α を大きくとった場合，学習時に含まれる出現

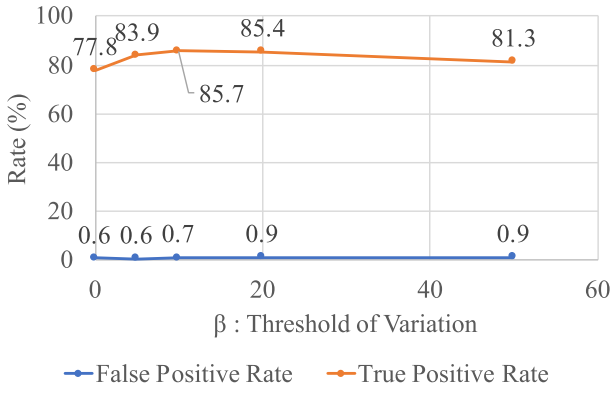

図 5 閾値 β の変化（$\alpha = 0.001$，$\gamma = 0.5$）
Fig. 5 Variation of β ($\alpha = 0.001, \gamma = 0.5$).

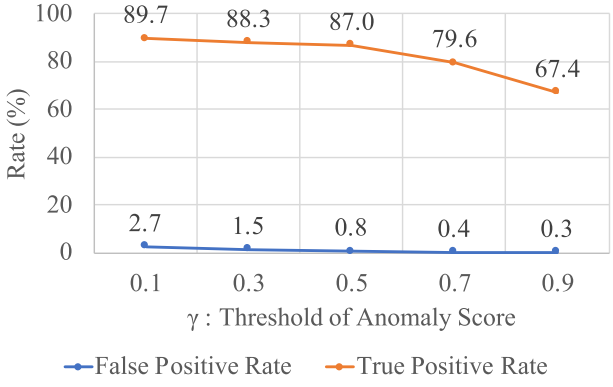

図 6 閾値 γ の変化（$\alpha = 0.001$，$\beta = 10$）
Fig. 6 Variation of γ ($\alpha = 0.001, \beta = 10$).

確率が低いであろう攻撃データを除外するとともに，正常なデータまでも除外してしまうことにある．また，α の値が [0, 0.001] 区間と小さい場合，誤検知率の増加（0.70%）より検知率の増加が大きい（4.79%）が，α の値が [0.001, 0.005] 区間と大きい場合，誤検知率の増加（3.3%）が検知率の増加（2.2%）よりも大きくなるため，誤検知を多発させないためにも，α は 0 に近い値に設定するのが望ましいと考える．

閾値 β に関する分析 図 5 により β の変化に対して誤検知率は大きく変化しないことが分かる．また，検知率にはピークが存在し，今回の実験では $\beta = 10$ 付近であると推測できる．β の値が小さいとき，文字クラス集合がプロファイルの特徴として選択される割合が高くなるため順序性の制約条件が緩和された分，検知率も低い．β の値が大きいとき，文字クラス例がよりプロファイルの特徴として選択される割合が高くなりすぎるため，検知率向上に寄与していた文字クラスによる制約が取り除かれてしまい，検知率が低下した．たとえば，入力値 `hello 2019/03` を学習して，SQL インジェクションである文字列 `or 1=1` を検知することを考える．文字クラス集合として学習する場合，{`ATN/`} となるため，検知時，文字 `=` が学習済みの文字クラスに含まれないため，攻撃として検知できる．しかし，文字クラス列として強制的に学習させた場合，文字クラス列 `ATN/N` となり，`or 1=1` の文字クラス列 `ATN=N` との類似度は $\frac{4}{5+5-4} = 0.66$ と高い値となるため，検知率が低下してしまったことが原因である．

閾値 γ に関する分析 図 6 より，γ が増加すると，検知率，誤検知率は単調減少する．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ため，α と同様，0 に近い値に設定することが望ましい．

4.6 誤検知分析

検知データ全体に対して提案手法の精度が最も高くなる閾値設定である $\alpha = 0.001$，$\beta = 10$，$\gamma = 0.5$ とした場合，提案手法では 5,296 件の誤検知が発生していた．図 7 に

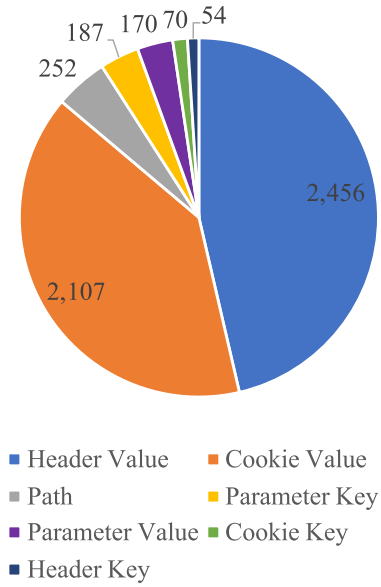

図 7 要素ごとの誤検知の件数
Fig. 7 Parcentage of false positive by detected position.

HTTP リクエストの要素ごとの誤検知の件数を示す．誤検知の多くはヘッダおよび Cookie であり，URL パスや URL パラメタと比較すると入力値の自由度の高さが誤検知の原因になっていると考える．

誤検知の要因は学習時に現れなかった文字列の出現や，文字列の構造の変化である．実験で確認できた主な事例を紹介する．

- **XFF ヘッダ** XFF（X-FORWARDED-FOR）ヘッダはプロキシなどを経由して HTTP リクエストが送信された場合にその送信元を示している．実験では学習時は IPv4 アドレスのみが観測されたため，文字クラス列 `N.N.N.N` といったプロファイルが作成されたが，検知時は `unknown` といった値が観測され，文字クラス列が `A` となり，誤検知となっている．この事象はプロキシサーバの設定の違いによって発生する．
- **Base64 エンコード文字列** HTTP 1.X の場合 Web アプケーションはバイナリなどのデータを HTTP リクエストにて送信する際，Base64 エンコードして ASCII

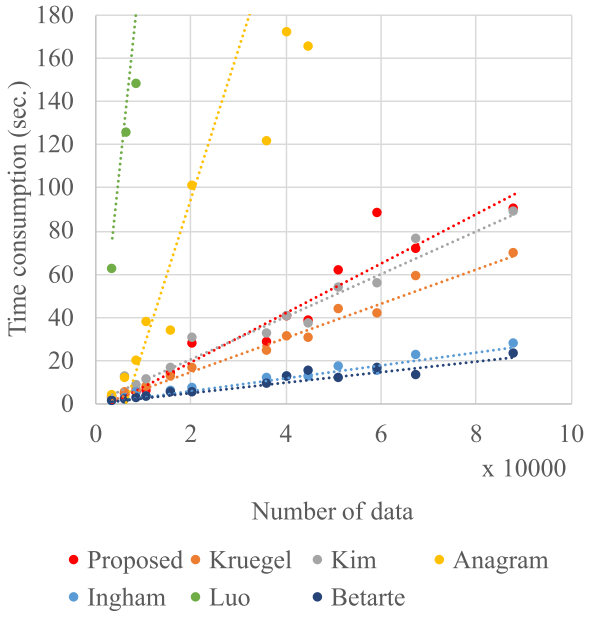

図 8 学習時の処理時間
Fig. 8 Time consumption at training.

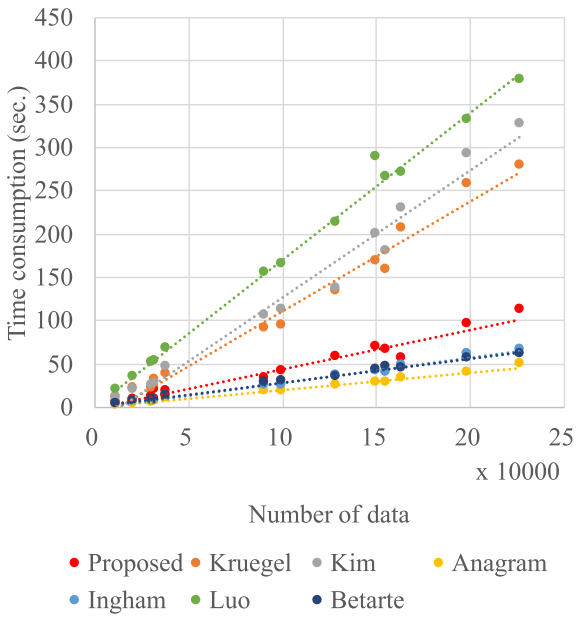

図 9 検知時の処理時間
Fig. 9 Time consumption at detection.

文字に変換してから送信する場合が一般的である．提案手法では Base64 エンコードという型を識別できる訳ではないため，アルファベット，数字などから成る文字クラスの集合として認識する．よく利用される RFC 4648 の Base64 エンコーディングではアルファベット，数字および記号 +，/ の 64 文字で構成される．そのため記号 +，/ は $\frac{2}{64} = 3.13\%$の確率でしか出現しないため，学習時に観測できず，検知時に誤検知を発生させる原因となる．

これらの要因による誤検知は学習時に特徴を工夫して解決することは困難であるため，関連研究にて述べる学習方法の工夫により解決を図ることが可能である．たとえば，XFF ヘッダの場合，1 つの Web サイトの観測データだけでなく，複数の Web サイトの観測データから学習し，プロファイルを共有することで誤検知を減らす可能性を上げることができる．

4.7 処理時間

図 8 と図 9 に，各 Web サイトにおいて学習と検知に要した処理時間を示す．各手法で検知精度が最も高くなるパラメタの場合を計測した．横軸は各 Web サイトの処理データ数であり，縦軸は処理時間（秒）である．破線は各手法の結果の近似線である．近似線より，提案手法の学習時の処理速度は約 1.01 ミリ秒/件，検知時の処理速度は約 0.49 ミリ秒/件であった．既存手法と比較すると，同等ぐらいの処理速度であった．

提案手法はオフラインでの学習を想定しているが，オンラインでの学習を適用した場合でも，秒間約 1,000 件のアクセスが行われる大規模サイトにおいてもリアルタイムな処理が可能であることが期待できる．Web サイトの死活監視サービスを行っている Pingdom[*6] によると Web ページあたりの平均的な HTTP リクエスト数は 110 件ぐらいであるため，検知時に提案手法をインライン[*7]で適用した場合においても，約 54.1 ミリ秒の遅延に抑えることが可能である．同じく Pingdom の調査によると，Web ページの遅延時間は約 3 秒までが理想的とあり，3 秒に対して，54.1 ミリ秒は 1.8%であり，ごくわずかな増加であるので，処理速度面においては実用的であると考える．

処理データ数と処理時間が比例しない箇所が存在する．これは処理時間が HTTP リクエスト数のみならず，各 HTTP リクエストに含まれる URL パラメタの数やヘッダの数および入力値の複雑性にも左右されるためである．

5. 関連研究

ネットワーク通信に対するアノマリ検知を適用する手法 [14], [15], [16] では通信プロトコルに依存せずに対応する必要があるため，ネットワーク通信に現れるバイト列の頻度あるいは，通信の特徴を数値化した後にアノマリ検知を適用することが一般的である．

Web アプリケーションの通信に特化したアノマリ検知手法には古くから文字列長や n-gram などを利用する手法 [1], [3], [4], [5] などが存在する．近年では，並列的に学習器を 1 つずつ順番に構築していく XGBoost を利用する Luo らの手法 [6] や，混合ガウスモデルを利用した Betarte らの手法 [7] が存在する．これらの手法では HTTP リクエストを特徴ベクトルに変換する際に工夫がないと高精度な

[*6] https://www.pingdom.com/
[*7] クライアントーサーバ間に適用し，攻撃と検知した場合遮断できるようにする実施方法

検知は期待できない．提案手法では比較的容易である文字クラス列をあらかじめ定義すればよいため，特徴量の調整が少なく安定した検知精度を発揮できると考える．

2 章で述べた既存手法は特徴のとらえ方を工夫することで検知精度を向上させるものである．しかし，特徴のとらえ方以外にも学習方法を工夫することで誤検知を低減させる手法が存在する．Maggi らは Web アプリケーションの仕様が変化して，URL パラメタの入力値の仕様が変化した場合に誤検知してしまう問題に着目し，HTTP レスポンスの内容を活用して，逐次的に再学習を行い，プロファイルを更新することで誤検知を低減する手法 [9] を提案している．Robertson らは，ある URL パラメタに対する学習データが少ない場合，検知時は誤検知が多発してしまう問題に着目し，ある URL パラメタについて正確なプロファイルを作れない場合でも，他の正確に学習できた URL パラメタのプロファイルを活用することで，誤検知を低減する手法 [17] を提案している．提案手法においてもこれらの学習方法を工夫する手法と組み合わせることでさらに精度を向上させることが可能である．

6. おわりに

本稿では Web アプリケーションの脆弱性を悪用する未知攻撃を検知する際に発生する誤検知の課題に対して，HTTP リクエストの各要素の文字列の構造に着目し，学習を行うことで，誤検知を低減する手法を提案した．評価の結果，既存のアノマリ検知手法に比べ提案手法の方が精度が高く，また処理性能も実用的な値であることを示した．今後は，入力値が学習時に出現する頻度が低い場合，出現しない場合の対応策を検討し，さらなる誤検知の低減を目指す．

参考文献

[1] Kruegel, C. and Vigna, G.: Anomaly Detection of Web-based Attacks, *ACM CCS* (2003).
[2] Kruegel, C., Vigna, G. and Robertson, W.: A multi-model approach to the detection of web-based attacks, *Computer Networks* (2005).
[3] Kim, T.H., Kim, K., Kim, J. and Hong, S.J.: Profile-based Web Application Security System with Positive Model Selection, *Proc. 2nd Joint Workship on Information Security* (2007).
[4] Ingham, K.L. and Inoue, H.: Comparing Anomaly Detection Techniques for HTTP, *RAID* (2007).
[5] Wang, K., Parekh, J.J. and Stolfo, S.J.: Anagram: A Content Anomaly Detector Resistant to, *RAID* (2006).
[6] Luo, Y., Cheng, S., Liu, C. and Jiang, F.: PU Learning in Payload-based Web Anomaly Detection, *Security of Smart Cities, Industrial Control System and Communications* (*SSIC*) (2018).
[7] Betarte, G., Giménez, E., Martínez, R. and Pardo, Á.: Improving Web Application Firewalls through Anomaly Detection, *IEEE ICMLA* (2018).
[8] Chen, T. and Guestrin, C.: XGBoost: A Scalable Tree Boosting System, *KDD* (2016).
[9] Maggi, F., Robertson, W., Kruegel, C. and Vigna, G.: Protecting a Moving Target: Addressing Web Application Concept Drift, *RAID* (2009).
[10] Bergroth, L., Hakonen, H. and Raita, T.: A Survey of Longest Common Subsequence Algorithms, *SPIRE* (2000).
[11] Kohavi, R.: A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection, *IJCAI* (1995).
[12] Roesch, M.: Snort: Lightweight Intrusion Detection for Networks, *LISA* (1999).
[13] Hanley, J.A. and McNeil, B.J.: The meaning and use of the area under a receiver operating characteristic (ROC) curve, *Radiology* (1982).
[14] Wang, K. and Stolfo, S.J.: Anomalous Payload-based Network Intrusion Detection, *RAID* (2004).
[15] Song, J., Ohira, K., Takakura, H., Okabe, Y. and Kwon, Y.: A Clustering Method for Improving Performance of Anomaly-Based Intrusion Detection System, *IEICE Trans. Information and Systems* (2008).
[16] Song, J., Takakura, H., Okabe, Y. and Kwon, Y.: Unsupervised Anomaly Detection Based on Clustering and Multiple One-Class SVM, *IEICE Trans. Communications* (2009).
[17] Robertson, W.K., Maggi, F., Kruegel, C. and Vigna, G.: Effective Anomaly Detection with Scarce Training Data, *NDSS* (2010).

鐘本 楊

2011 年名古屋大学工学部卒業．2013 年同大学大学院情報科学研究科博士前期課程修了．同年日本電信電話株式会社入社．サイバー攻撃対策技術に関する研究に従事．

青木 一史 （正会員）

2004 年東北大学工学部卒業．2006 年同大学大学院情報科学研究科博士前期課程修了．同年日本電信電話株式会社入社．主任研究員．サイバー攻撃対策技術に関する研究に従事．電子情報通信学会会員．

三好 潤

1993 年京都大学工学部卒業．1995 年同大学大学院工学研究科修士課程修了．同年日本電信電話株式会社入社．主幹研究員．ネットワークセキュリティに関する研究に従事．電子情報通信学会会員．

嶋田 創 （正会員）

1998 年名古屋大学工学部卒業．2000 年同大学大学院工学研究科博士前期課程修了．2004 年同大学工学博士．名古屋大学工学部電気系 COE 研究員，京都大学大学院情報学研究科特任助手，同大学院情報学研究科助手，奈良先端大学院大学情報科学研究科准教授を経て，2013 年名古屋大学情報基盤センター准教授．低消費電力と高信頼性を兼ね備えた計算機アーキテクチャとネットワーク関連の研究に従事．電子情報通信学会，IEEE 各会員．

高倉 弘喜 （正会員）

1990 年九州大学工学部卒業．1992 年同大学大学院修士課程修了．1995 年京都大学大学院博士後期課程修了．博士（工学）．米国イリノイ州立大学訪問研究員，奈良先端科学技術大学院大学助手，京都大学講師，助教授（准教授），名古屋大学教授を経て，2015 年から国立情報学研究所教授．サイバーセキュリティ，高機能ネットワークの研究に従事．電子情報通信学会，システム制御情報学会，地理情報システム学会，ACM 各会員．