情報処理学会論文誌 Vol.60 No.3 945–955 (Mar. 2019)

推薦論文

攻撃コードのエミュレーションに基づくWebアプリケーションに対する攻撃の成否判定手法

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 2018年7月13日, 採録日 2018年12月4日

概要：WAF や IDS 等のセキュリティ製品は Web に対する攻撃検知において重要な役割を担っている．しかし，大量のアラートから重大なインシデントに関わるアラートを人手で探し出すには多くの時間を要する．本研究では，アラートの重要度を決定するための攻撃成否判定手法を提案する．提案手法では攻撃コードのエミュレーションを行い，攻撃成功時の痕跡を抽出する．この痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定し，アラートの重要度を決定する．提案手法の精度・性能評価結果，および発見した攻撃事例から，その実用性を示す．

キーワード：Web セキュリティ，アラート検証，IOC，エミュレーション

Detecting Successful Attacks against Web Application based-on Attack Code Emulation

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

Received: July 13, 2018, Accepted: December 4, 2018

Abstract: Security appliances such as WAFs and IDSs contribute to detecting threats of web attacks greatly. However, it requires much time when we discover critical incident related alerts from massive alerts of security appliances. In this research, we propose a system that verifies the criticalness of alerts based on an indicator of attacks. The proposed system emulates exploit code to extract an indicator of succeeded attacks. By matching the indicator with HTTP response content, we can confirm success or failure of the attack which is directly connected to importance of the alert. We show an effectiveness of the system through accuracy/performance evaluation, and case studies.

Keywords: Web security, alert verification, IOC, emulation

1. はじめに

Web アプリケーションは一般に外部に公開されており，誰でもアクセスできるため，攻撃者も容易にアクセスが可能である．そのため，Web アプリケーションに対する脆弱性を悪用する攻撃や脆弱性の有無を確認するスキャンは日々，大量発生している．Web 攻撃への対策装置である WAF（Web Application Firewall）や IPS（Intrusion Prevention System）は攻撃を遮断するため，Web アプリケーションのセキュリティ担保において重要な役割を担っている．しかし，誤遮断による Web サービスへの悪影響を避けるため，遮断する対象は既知の攻撃と断定できる通

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

本論文の内容は 2017 年 10 月のコンピュータセキュリティシンポジウム 2017（CSS2017）で報告され，同プログラム委員長により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

信に限定していたり，検知のみを実施するIDS（Intrusion Detection System）と同様の運用をしていたりする場合も多く存在する．

WAFは攻撃を検知した際アラートを出力する．そのため，大量の攻撃にさらされるWebサイトの管理者は，大量のアラートを受け取ることとなる．アラートを確認し，インシデントが発生したかを確認する部分は人手で行うことが一般的であるため，大量のアラートの確認には多くの時間を要する．しかし，多くの場合において，成立しない攻撃もアラートに含まれるため，大量のアラートのうち，実際にセキュリティ侵害（改ざん，乗っ取り，漏えい等）に至るものは，ごく一部である．そのため，まずセキュリティ侵害が発生したことを示すアラートであるか否かを見極め，優先して対処できるかが課題である．

本研究では攻撃の成否を判定することで，アラートの重要度を決定する．つまり，攻撃が成功しているのであれば，そのアラートはより重要と判定することでセキュリティオペレーションの効率化につなげる．関連研究ではサーバ内のシステムコールや命令列を観測したり[1], [2], [3], [4], [5]，攻撃の複数の状態をルールとして定義したり[6], [7], [8]，することで成否を判定している．しかし，このような仕組みの導入ではシステム改変が必要となるため，商用環境において導入障壁が大きかったり，攻撃の状態を定義するルール作成を行うため利用者に要求される知識レベルが高いという制約が存在する．本研究ではシステム改変不要（**要件1**）および比較的低い知識レベルの利用者でも運用可能（**要件2**）という2つの要件を満たす手法を提案する．

本研究の貢献は以下のとおりである．

- 本研究ではシステム改変が不要かつ，比較的低い知識レベルの利用者でも運用可能な成否判定手法を提案した．提案手法では攻撃コードのエミュレーションを行い，攻撃の痕跡であるIOC（Indicator Of Compromise）を抽出する．その後，IOCがHTTPレスポンスに含まれるか否かで攻撃の成否を判定し，アラートの重要度を決定する．
- 人工的に作成したデータと実環境のデータで評価を行い，それぞれ，62.9%および52.1%の不要なアラートを削減できること，WAFのアラートでは気づけなかった成功した攻撃を発見できること，約98%のリクエストを1秒以内に処理できることを示し，提案手法の効果を示した．約44%のアラートが見過ごされている状況（2.2節）に鑑みると目標を達成できたと考える．

エミュレーションする際の特徴はエミュレートする部分は攻撃者が悪用する脆弱性ではなく，攻撃者が達成したい目的の部分に着目していることである．これより，多種多様なWebアプリケーションをエミュレートする必要はなく，Webサーバで利用するOSやミドルウェアといった限定的な環境を用意できればよい．

2. セキュリティオペレーションにおける攻撃成否判定の重要性

2.1 攻撃の成否

サーバに対する攻撃の結果は，成功と失敗に大別される．攻撃の成功とは，攻撃者の意図どおりに攻撃が行われている状態のことである．たとえば，以下のようなWebサーバに対する任意のOSコマンド実行を狙う攻撃があるとする．

```
GET /index.php?path=;cat%20/etc/passwd
```

攻撃者が挿入したコマンド`cat /etc/passwd`が実行され，ファイル`/etc/passwd`の内容が以下のようにレスポンスに表示されていた場合攻撃は成功していると判断される．

```
<html><h1>Welcome!</h1>
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin...</html>
```

しかし，以下のようにファイル`/etc/passwd`が表示されていない場合，攻撃者が挿入したOSコマンドは実行されていないため，攻撃は失敗していると判断できる．

```
<html><p>Page not found.</p>...</html>
```

2.2 セキュリティオペレーション

サイバー攻撃が高度化・大規模化している現在では，自組織だけで攻撃に対処できない状況が発生しており，サイバー攻撃の監視・分析を専門の企業に委託することが多くなっている．こうしたサイバー攻撃を監視・分析する企業はSOC（Security Operation Center）を構築し，IDSやWAF等のアラートを24時間365日監視し，侵害がないか分析者が確認している．しかし，脆弱性を悪用する攻撃が日々大量に発生するため，アラートが大量になってしまい，限られた人員ですべてのアラートに対処することが困難になってきている．Ciscoのレポート[*1]によれば，組織で発見された44%ものアラートは対処できずに見過ごされていると報告されている．

アラートへの対処を効率化するためにはアラートを重要度で振り分け，対処すべきものを優先的に対処して，そうでないものは優先順位を下げる，あるいは，対処すべきものから除外する必要がある．攻撃の成否が判定できれば，攻撃が成功していればそのアラートは重要，攻撃が失敗していればそのアラートは重要でないと判別できるようになり，アラートの対処の効率化に寄与できる．しかし，実用化されているIDSやWAFは攻撃検知を行うが攻撃成否の判定まではできていない．

本研究では攻撃検知後，その攻撃の成否を判定を行う手法およびその実装について提案を行う．

[*1] https://www.cisco.com/c/dam/m/ja_jp/offers/173/sc04/cisco-annual-cybersecurity-report-2017.pdf

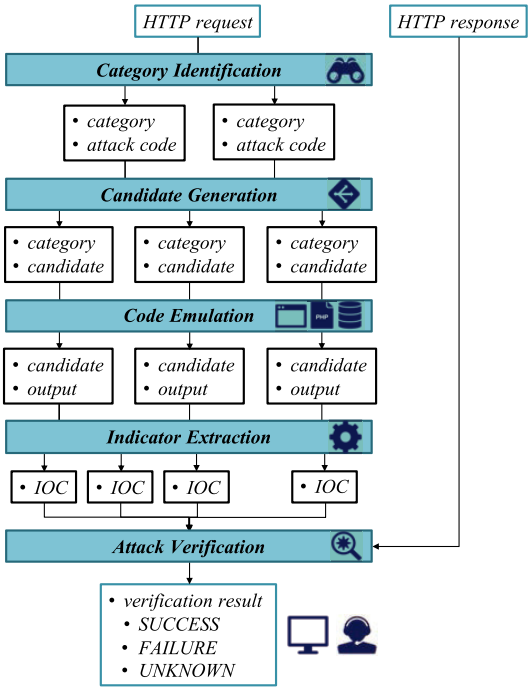

図 1 提案手法の各処理とデータの流れ

Fig. 1 Overview and data flows of proposed method.

3. 提案手法

提案手法の全体像を図 **1** に示す．提案手法では 5 つのステップにより攻撃成否の判定を行う．

提案手法は，WAF 等で攻撃検知された Web 攻撃の HTTP リクエストおよび HTTP レスポンスのデータを入力とする．出力の判定結果は攻撃成功，攻撃失敗および判定不可の 3 つである．まず，攻撃検知された HTTP リクエストに対して，攻撃カテゴリの判定および攻撃コードの抽出を行う（**Category Identification**）．次に，抽出した攻撃コードから複数の部分的な攻撃コード（攻撃コード候補）を生成し（**Candidate Generation**），生成した複数の攻撃コード候補をエミュレータで実行する（**Code Emulation**）．次に，エミュレータの出力を整形し，攻撃の痕跡（IOC）を抽出する（**Indicator Extraction**）．最後に，IOC を HTTP レスポンスと比較し，攻撃が成功したか否かを判定する（**Attack Verification**）．以降，各節にて各処理ステップを詳しく説明する．

3.1 Category Identification

攻撃において脆弱性を悪用する部分はアプリケーションごとに異なり多様であるが，脆弱性を悪用した際に行う動作は限定的である．本研究ではこの動作を**表 1** に示す 4 つのカテゴリに分ける．この 4 つのカテゴリは OWASP Top 10 [*2]でも示されるように脆弱性を悪用する攻撃の代表的なものである．

[*2] 国際的な Web アプリケーションのセキュリティコミュニティである OWASP が公開している対策すべき重要な Web アプリケーションの脆弱性

表 1 攻撃カテゴリ一覧

Table 1 Categories of attacks.

カテゴリ	説明
RCE	OS コマンドやプログラムコードを悪用する攻撃
SQLI	SQL コマンド（DB の機能）を悪用する攻撃
PT	ファイル操作を悪用する攻撃
XSS	HTML や JavaScript を悪用する攻撃

表 2 攻撃カテゴリを判別するための固有表現の例

Table 2 Examples of individual expression for each category.

カテゴリ	サブカテゴリ	固有表現例
RCE	SH	`echo, cat, uname`
	PHP	`phpinfo(), $_GET, $_POST`
	...	...
SQLI		`SELECT, UPDATE`
PT		`../, /etc/, /proc/, *.bak, *~`
XSS		`<script>, <iframe>, alert(`

攻撃カテゴリを判別する理由は 2 つある．1 つ目はエミュレーションの正確性を高めるためである．正しくないエミュレータを利用して得られた攻撃の痕跡はその攻撃の成否を断定する根拠として乏しいので，攻撃の実行に適切な環境を選択することは精度向上のために重要である．たとえば，OS コマンド実行である攻撃コード `cat /etc/passwd` を XSS（Cross-Site Scripting）と誤判定した場合，この攻撃コードの文字列が出現しているか否かで攻撃の成否を判定するになってしまうため誤判定を誘発する．2 つ目は **Code Emulation** ステップの処理時間を短縮するためである．Web アプリケーションを構成する環境は OS の種類，実行環境の種類，DB の種類が一様ではない，また **Candidate Generation** ステップにより複数の攻撃コードの候補が生成されるため，複数回の攻撃コードのエミュレーションを Web アプリケーションを構成するすべての環境で実施すると処理に時間がかかってしまう．そのため，攻撃カテゴリを判別し，攻撃コードに対して適切な環境のみでエミュレーションを行うことで処理時間を短縮する．

攻撃カテゴリの判断には各カテゴリの固有表現に攻撃コードが一致するかどうかで決定する．RCE（Remote Command/Code Execution）カテゴリの攻撃の判定には，OS コマンドやプログラミング言語固有の表現が攻撃コード中に存在するかどうかによって判断する．また，**表 2** に示すように，RCE カテゴリには OS コマンドあるいはプログラミング言語の差異を区別するためのサブカテゴリを設ける．OS コマンドの固有表現には，`/bin` や `/usr/bin` 以下のプログラム名および，`bash` 等のビルドインコマンドの一覧を利用する．PHP 等のプログラミング言語の固有表現には関数名の一覧等を利用する．SQLI（SQL Injection）カテゴリの固有表現には DB にアクセスする際に使用する

SQL 句を利用する．PT（Path Traversal）カテゴリの固有表現にはファイルパスの相対参照をする際に必要な../といった表現，および Web アプリケーションとは直接関連しないパス（/etc，/proc），バックアップファイルパス（*.bak, *~）等の表現を利用する．XSS の固有表現には HTML タグや JavaScript の関数名を利用する．表 2 に各カテゴリに判別するための固有表現の一例を示す．

固有表現の設定は利用者で調整を行うことも可能だが，入手容易な OS コマンドの一覧や関数の一覧等を設定するのみであり，入手困難な攻撃方法に関する情報ではないため，利用者に要求する負担は小さいと考える．

判定では，攻撃コードに対する URL エンコード等はすべてデコードされている状態を前提としている．また，1 つの攻撃コードが複数のカテゴリに分類される場合もある．たとえば，以下の攻撃コードは cat の部分により RCE カテゴリと判定され，../の部分で PT カテゴリと判定される．

```
cat ../../etc/passwd
```

攻撃カテゴリを判定できない場合は判定不可の結果を出力し，処理を終える．

3.2 Candidate Generation

攻撃コードによってはそのままではエミュレーションすることができない場合が存在する．たとえば，SQL インジェクション攻撃では，任意のステートメントを実行するのに

```
UNION SELECT version()#
```

のように UNION を付与したコードの実行を試みる．しかし，このステートメントをそのまま実行すると，UNION という SQL 句が余分となり，正しく実行できない．このステップではエミュレーションの正確性を高めるために，実行する攻撃コードを整形した候補を複数生成する．RCE カテゴリの攻撃コードに対してはコマンド名を攻撃コードの先頭とした候補を抽出する．SQL カテゴリの攻撃コードに対しては (SELECT, UPDATE) 等の SQL コマンドを攻撃コードの先頭とした候補を抽出する．XSS カテゴリの場合，HTML タグ単位で候補を生成する．なお，元々の攻撃コードも候補の 1 つとして扱う．上記の例の場合，

```
UNION SELECT version()#
SELECT version()#
```

の 2 つの攻撃コード候補が生成される．

3.3 Code Emulation

攻撃カテゴリに応じて攻撃コードを実行するエミュレータを用意し，攻撃コードをその環境にて実行する．エミュレーションが終了しないこと（たとえば sleep コマンドの実行）を避けるため，エミュレーションにはタイムアウトを設ける．

エミュレータでは攻撃カテゴリごとに異なる手法で攻撃コードを実行する．RCE カテゴリの攻撃に対しては OS コマンドを実行できる GNU bash や，PHP コードを実行できる PHP インタプリタ等を用いて攻撃コードを実行する．SQLI カテゴリの攻撃に対してはインストールが終了した初期状態の DB サーバを用意し，SQL 文を実行する．PT カテゴリの攻撃に対しては攻撃がアクセスしようとしているファイルパスあるいはファイル名を OS のディレクトリおよびユーザが指定した Web アプリケーションの設定ファイルが保管されているディレクトリから検索し，存在すればそのファイルを表示することでエミュレーションを行う．バックアップファイルパス（*.bak, *~等）の場合は*の部分を抽出し，そのファイルパスが存在すれば，ファイルを表示することこでエミュレーションを行う．XSS カテゴリの攻撃は攻撃コードそのものが HTTP レスポンスに現れるため，エミュレーションは行わず，HTTP レスポンス中での攻撃コード自体の有無によって成否判定を行っている．XSS カテゴリの成否判定については既存手法 [9] 等を用いても構わない．

幅広い攻撃に対応するために，エミュレータ環境には攻撃コードを実行する環境である OS やミドルウェアを複数用意する必要がある．また，エミュレータ環境自体が攻撃コードによって改変される可能性が高いため，以下の 3 つの条件が必要となる．

(1) 複数の OS やミドルウェアで利用可能であること
(2) プログラムを実行した後でも実行前の状態に戻せること
(3) 実行前の状態を高速に復元できること

4 章で詳細に説明するが，QEMU [*3] 等の OS 仮想化技術ではなく，より軽量に実行・復元が可能である Linux コンテナ技術である Docker [*4] を活用して実装を行うことで上記の 2 つの条件を解決した．

実際の環境とエミュレーション環境の差異が生じることで，成否判定の結果に悪影響を及ぼすことがあるため，広く利用される OS や実行環境を用意しエミュレーションを行う．たとえば SQLI カテゴリの攻撃コードに対しては MySQL と PostgreSQL 両方の環境でエミュレーションを行う．提案手法では脆弱性を悪用する部分をエミュレーションするのではなく，脆弱性を悪用した後に実行される攻撃コードのエミュレーションを行う．そのため，多種多様な Web アプリケーションの環境を用意する必要はなく，限られた OS，実行環境や DB を用意すればよい．

エミュレーションする際の特徴は Web アプリケーショ

[*3] http://www.qemu.org/
[*4] https://www.docker.com/

ンの脆弱性ではなく，攻撃者が実行したいコードあるいはコマンドに着目していることである．そのため，多種多様な Web アプリケーションをエミュレートする必要はなく，Web サーバで利用する OS やミドルウェアといった限定的な環境を用意できればよい．

3.4 Indicator Extraction

エミュレータで攻撃コードを実行すると結果が標準出力あるいは標準エラー出力に出力される．そのため，これらの出力を利用して攻撃が成功したことを示す指標である IOC を抽出する．

3.4.1 IOC 候補の抽出

標準出力に出力されたすべての内容を HTTP レスポンスとそのまま比較すると，検知漏れが発生する恐れがある．これは Web アプリケーションによっては画面表示が一部分に制限されていたり，空白等が変換されて，一連の出力にならない等の仕様があるからである．標準出力に対する以下に示す IOC 候補抽出処理を行う．なお，IOC 候補の集合の初期状態は空の集合である．

(1) 出力が複数行の場合は，各行をそれぞれの IOC 候補とする．
(2) 出力の各行がタブ文字もしくは空白文字で区切られている場合は，タブ文字で区切った文字列をそれぞれ IOC 候補とする．

3.4.2 不適切な IOC 候補の除外

IOC は攻撃が成功したことを示す痕跡であるため，攻撃失敗時のレスポンスに現れると誤検知となってしまう．たとえば，攻撃コードが `echo 123`，`echo welcome` の場合，抽出される IOC は `123`，`welcome` となるが，Web ページとしては頻出する表現であるので，攻撃が失敗していても成功したと誤検知してしまう．このような誤検知を多発させないために，不適切な IOC 候補を除外する処理を行う．以下に除外条件を示す．

(1) IOC の文字列長が T 以下（T はあらかじめ指定する）
(2) 汎用的な HTML タグ（例：`<html>`，`<head>`，`<body>` 等）が存在
(3) 一般的に利用される英単語等の文字列（例：`welcome`，`login` 等）が存在

3.4.3 IOC 候補の正規化

最後に，実際の環境とエミュレーション環境の差異を少なくするため，IOC 候補に対して正規化処理を行う．正規化処理では変化しやすい部分である日付や時刻等の数字等を正規表現に変換する．これらの表現はあらかじめ用意する．たとえば，`last` コマンドを実行時には `wtmp begins 15:55:59 2017` という文字列が標準出力として得られるが，本処理によって `wtmp begins \d+:\d+:\d+ \d+`のような IOC 候補に変換される．正規化できない場合は正規化を施さずに処理を終える．

残った IOC 候補を攻撃の成否判定に利用する IOC とする．残った IOC 候補がない場合，攻撃の成否を判断することができないことから，判定不可の結果を出力し，一連の処理を終える．

3.5 Attack Verification

このステップでは HTTP レスポンスに IOC が含まれているか否かで攻撃の成否を判定する．前ステップで IOC が抽出されなかった場合は成否を判定できないため判定不可となる．IOC が存在し，HTTP レスポンスに IOC が含まれる場合は攻撃コードが実行されたと見なせるため，攻撃が成功したと判定する．そうでない場合，攻撃が失敗したと判定する．

4. 実装

提案手法を実装したシステムの全体像を**図 2** に示す．HTTP リクエストおよびレスポンスの取得には OSS のネットワーク監視ツールである Bro[*5] を利用した．攻撃カテゴリの判定および攻撃成否の判定は Python で実装した．攻撃コードをエミュレートする環境には Docker を採用した．Docker は Linux コンテナ技術を活用しており，OS 仮想化に比べ起動・復元が高速に行える．

提案システムでは攻撃が発生するたび，エミュレータとなる Docker Container を作成し，その環境内で攻撃コードを実行する．なお，エミュレーション時に外部への攻撃を防ぐため，ネットワーク通信はすべて遮断した．この設定による影響については 6 章にて述べる．PT カテゴリの攻撃に対しては OS のファイルをエミュレーションに活用する以外にも，Web アプリケーション固有のファイルを追加することで，判定可能な攻撃を増やすことも可能である．実験では，頻繁に発生するコンテンツ管理システム（CMS）である WordPress，Drupal および Joomla 等の設定ファイルを追加した．一例としては WordPress の設定ファイルである `wp-config.php` 等が存在する．

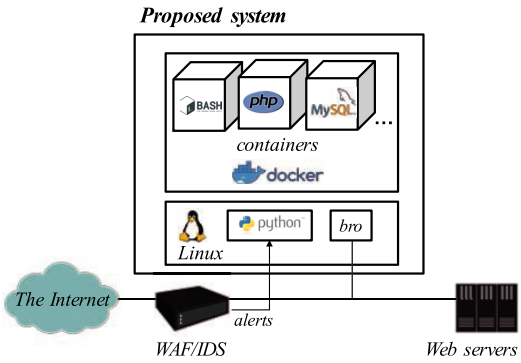

図 2 システム構成
Fig. 2 System architecture.

[*5] https://www.bro.org

表 3 データセット概要
Table 3 Dataset overview.

データセット	D_{lab}	D_{real}
観測環境	仮想環境	実環境
観測期間	–	Feb 2017–Feb 2018
データ数	1,208	184,599
送信元 IP アドレス数	–	10,138
送信先 IP アドレス数	–	848

表 4 集約後の攻撃コードの例
Table 4 Examples of aggregated attack codes.

集約前	集約後

`echo GBJIVY$((45+22))`
`echo GEWHYE$((65+76))`
`echo HWUZPQ$((82+33))`
	`echo GBJIVY$((45+22))`

`sleep(5)` `sleep(10)`	`sleep(5)`
`../../etc/passwd` `../../../etc/passwd`	`../../etc/passwd`

5. 評価

判定精度および処理性能の観点から評価を行った．利用したデータセットの概要を**表 3** に示す．評価には仮想環境で人工的に作成したデータセット D_{lab} および大規模実ネットワーク環境のトラフィックから生成したデータセット D_{real} の 2 種類のデータを用いた．精度評価には D_{lab} と D_{real} を用い，性能評価には D_{real} を用いた．

D_{lab} では検知漏れ評価および誤検知評価のために，2 種類の評価用データ（攻撃成功時データ，攻撃失敗時データ）を以下の手順で用意した．ここでいう検知漏れとは成功した攻撃を失敗と判定したことを意味し，誤検知とは失敗した攻撃を成功と判定したことを意味している．判定不可の場合は検知漏れ，誤検知としては扱わない．

まず，公開されている攻撃コード[*6,*7]および実環境で観測した攻撃コードを収集した．次に，攻撃コードの意味としては同様であるが，乱数等が攻撃コードに用いられている場合を集約した．**表 4** に集約後の例を示す．集約後の攻撃コードは 7,819 個であった．集約する理由は，類似する同様な攻撃コード数の偏りによる精度の変動を避けるためである．これらの攻撃コードが成功した際に出力されると思われる痕跡を手動で付与できた攻撃コード 1,208 個を検証用攻撃データとした．また検証用攻撃データを実行した際の出力を攻撃成功時データとした．また，攻撃失敗時データは，Alexa Top 50[*8] ドメインの Web ページを結合したものを利用した．出力を付与できない攻撃コードは `sleep` のような時間が差異になるものや，`wget` のような外部に通信を行う攻撃である．本研究ではレスポンスに痕跡が現れる攻撃のみを対象としているが，処理時間や通信の試行も対象とすることでより対応できる攻撃が増える可能性があり，今後の課題である．

[*6] https://github.com/fuzzdb-project/fuzzdb
[*7] https://github.com/foospidy/payloads
[*8] https://www.alexa.com/topsites

表 5 検証用攻撃データの攻撃カテゴリ別データ数
Table 5 Data summary for each attack category.

カテゴリ	個数	説明
RCE	217	`cat`, `ls`, `uname`, `whoami`, `last`, `netstat` 等の情報を出力するコマンドの実行
SQLI	222	SELECT 文による DB のバージョン，ユーザ名，DB 名，ホスト名，テーブル名の取得
PT	136	Linux や Windows の設定ファイル等（`passwd`, `win.ini`）の取得
XSS	633	`script`, `img`, `body`, `a`, `embed` 等のタグの挿入，JavaScript コードの挿入

検知漏れ評価では，検証用攻撃データをエミュレートし，攻撃成功時データと合致する IOC を提案システムが出力できたか否かで評価を行う．IOC を出力できなかった場合，検知漏れとして扱う．誤検知評価では，検証用攻撃データをエミュレートし，出力された IOC が攻撃失敗時データに合致する IOC が出現するか否かで評価を行う．IOC が現れた場合，誤検知として扱う．D_{lab} に含まれる攻撃カテゴリごとの検証用攻撃データ数と攻撃コードの概要を**表 5** に示す．D_{real} の攻撃検知にはチューニングされた WAF の検知結果および，独自シグネチャを用いて判定を行った．

評価の際は最小 IOC 文字数の閾値 $T = 5$ として実験を行った．この理由は，$T = 5$ の場合が提案手法の効果を最大限に発揮する可能性が高いためである．T による精度の変化は 5.3.2 項にて詳しく述べる．

5.1 精度評価

D_{lab} を用いて精度評価を行った．評価には攻撃成功に対する適合率（Precision）・再現率（Recall）および，攻撃失敗に対する適合率・再現率の 4 つの指標を用いた．ここでいう適合率とは判定可能と判断したデータに対する判定結果が正しいデータの割合である．また，再現率は全データの指標ラベルに対して，同じ判定結果を出した場合の割合である．つまり，以下のとおり定義できる．

$$\text{適合率} = \frac{\text{判定結果が正しいデータの個数}}{\text{判定可能なデータの個数}} \quad (1)$$

$$\text{再現率} = \frac{\text{判定結果が正しいデータの個数}}{\text{全データの個数}} \quad (2)$$

D_{lab} に対する判定結果を**表 6** に示す．また，判定結果から算出した適合率および再現率の結果を**表 7** に示す．成功判定に対する適合率は 100%であるから，提案手法で攻撃が成功したと判定できた場合，その結果は正しく，重要なアラートであることを意味する．またデータセット全体

表 6 判定結果
Table 6 Evaluation result for D_{lab}.

判定結果＼指標	攻撃成功時データ	攻撃失敗時データ
攻撃成功	868	108
攻撃失敗	0	760
判定不可	340	340
計	1,208	1,208

表 7 D_{lab} に対する適合率および再現率
Table 7 Precision and Recall against D_{lab}.

項目			割合
適合率（Precision）	攻撃成功	100%	868/868
	攻撃失敗	87.5%	760/868
	全体	93.8%	1,628/1,736
再現率（Recall）	攻撃成功	71.9%	868/1,208
	攻撃失敗	62.9%	760/1,208
	全体	67.4%	1,628/2,416

表 8 D_{real} に対する評価結果
Table 8 Evaluation result against D_{real}.

判定結果	個数	割合（%）
成功	190	0.10
失敗	96,223	52.1
判定不可	88,186	47.8

に対しても 93.8%の適合率を示すことから，判定可能な攻撃については高い精度で成否が判定できている．一方，再現率は成功判定・失敗判定どちらも約 60%以上であった．つまり，セキュリティオペレーションを行う分析者にとっては対応すべきアラートが 60%以上削減されるため，稼働削減につながる．

次に，D_{real} に対する評価結果を**表 8** に示す．成功と判定した件数は 190 件であった．190 件のアラートについてその正しさを目視で確認した結果，攻撃成功と確認できたのは 90 件であり，残り 100 件は誤検知であった．この結果は，提案手法が調査すべきアラート数を 184,599 件から 190 件までに絞り込むことができたことを意味する．調査すべきアラート数を大きく減少させれることから，WAF や IDS のアラートでは気付きにくい成功した攻撃を発見できることを示している．攻撃成功と確認できた 90 件の攻撃は重複しており，実際は 3 種類の攻撃であった．誤検知の原因は 5.3 節にて述べ，検知した事例は 5.4 節にて紹介する．判定不可となった攻撃 88,186 件を詳細に分析し，原因を 5 つに大別できた．**表 9** にその結果を示す．最も大きな原因は出力がない攻撃である．これらの攻撃は脆弱性を確認するための処理時間の差を利用した攻撃であったり，サーバ内の設定を変更する攻撃等であった．たとえば，以下の攻撃コードの場合，

```
echo 'Hacked' > /var/www/index.php
```

表 9 判定不可となる原因
Table 9 Reason of unknown result.

原因	割合（%）
出力が存在しない攻撃	55.7
攻撃ではないリクエスト	27.2
未対応のエミュレーション環境への攻撃	14.7
外部通信を行う攻撃	2.37
その他	0.03

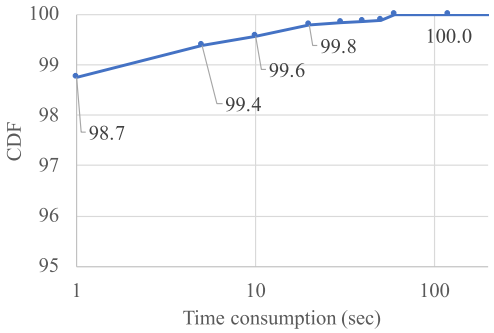

図 3 処理時間の累積分布関数
Fig. 3 CDF of time consumption.

攻撃が成功しても出力はないが Web サイトが改ざんされることとなる．攻撃した際に出力がないため，痕跡が抽出できず判定不可となった．次に原因として多い，攻撃でないリクエストは誤って WAF 等に検知された可能性が高いものであり，攻撃コードが抽出できず判定不可となった．未対応のエミュレーション環境への攻撃は Windows の設定ファイルや特定の Web アプリケーションの設定ファイルを漏洩させる攻撃であった．今回の実験ではエミュレータに利用する OS は Linux が前提であり，また特定の Web アプリケーションに特化していないため，判定不可となった．外部通信を行う攻撃は `wget` や `curl` を利用して攻撃に用いるスクリプトファイル等のダウンロードを行う攻撃であった．提案手法の実装では外部に通信を許可していないため，スクリプトファイルの内容確認できず，判定不可となった．

5.2 性能評価

性能評価では各攻撃リクエストの検証に要する時間を測定した．性能評価に利用したマシンのスペックは 1.3 GHz Intel Core i5 4250U CPU，8 GB RAM，512 GB SSD である．**図 3** に測定結果を示す．約 98%の攻撃リクエストに対する処理が 1 秒以内に終了している．つまり，WAF と Web サーバとの間にインラインで配置して，攻撃が成功したと判定できた場合に遮断を行うという運用をした場合でも，1 つのリクエストに対するレスポンスを約 1 秒程度の遅延に抑えられることを意味する．WAF にて攻撃と判定されるリクエストは，一般に全リクエストのうちのごく一部に限られる．したがって，そのような限られたリクエス

トを処理する場合のみ 1 秒程度の遅延が発生するのであればユーザビリティは大きく損なわれない[*9]ため，性能面でも実用的であると考える．

5.3 分析

5.3.1 誤判定の原因

正しく判定できなかった原因の分析を行った．D_{lab} による評価の結果，提案手法は成功した攻撃を失敗と判定することはなかった．また，D_{real} の攻撃失敗と判定された通信のレスポンスに対して D_{lab} で攻撃成功と判定した際の IOC を照合させたところ，合致するものは存在しなかった．公開されている攻撃コードは頻出するものであるから，D_{real} 中の攻撃が攻撃失敗と判定されている中に含まれている可能性は低いと考える．つまり，攻撃失敗と判定した場合はそのアラートは分析者で再度確認する必要がない信頼性を有することを意味しており，提案手法はオペレーションの効率化に寄与できる．成功した攻撃を失敗判定してしまう場合，結局分析者は再度，失敗判定されたすべての攻撃のアラートを確認し，成功した攻撃を発見せねばならず，セキュリティオペレーションの効率化にならない．

D_{lab} で評価した際，失敗した攻撃を成功と判定した件数は 108 件であった．その原因は IOC の正規化処理が不十分で，IOC に汎用的な表現が残ってしまい，一般的な Web ページの内容にも現れる表現を含んでしまったためである．たとえば，SQL インジェクションで `select version()` といった攻撃コードの場合，MySQL ではバージョンの文字列を表す`\d+.\d+.\d+`といった IOC が抽出される．これは一般的な Web ページにもよく現れる表現であったため，攻撃が成功したと誤って判定してしまっていた．

また 5.1 節で述べた D_{real} で評価した際の 100 件の誤検知原因は，上記の IOC が汎用的である以外に，レスポンスの内容をリクエストの際のボディ部に含めてリクエストを行うアクセスの存在にもある．たとえば，サーバからのレスポンス内容が `<html>Hello World</html>` のような固定値であり，この内容をリクエストのボディ部に設定してアクセスした場合，提案手法ではリクエストの内容がそのまま表示されていると判定し，XSS による攻撃が成功したと判定する．しかし，実際にはレスポンスは固定値のため，攻撃者によって任意のコードを挿入されてないが，攻撃が成功したと誤検知してしまった．

5.3.2 最小 IOC 文字列長 T による精度の変化

提案手法では IOC を抽出する際，最小 IOC の文字列長として閾値 T を設けている（3.4 節）．T の変動による適合率，再現率の変化を**図 4**，**図 5** にそれぞれ示す．適合率に関しては攻撃成功および攻撃失敗ともに頂点が存在する．特に $T = 5$ のときは，攻撃成功の適合率が 100%になり，成功した攻撃を見逃すことがなくなる．再現率に関しては，攻撃成功する場合は頂点（$T = 10$）が存在する．以上のことから，T は $[5, 10]$ の間に設定するのが適切である．

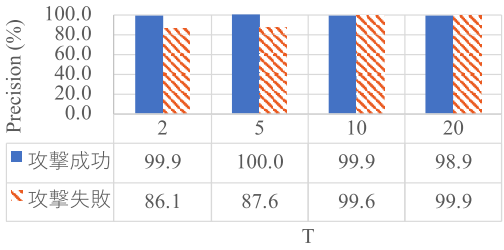

図 4 T による適合率の変化
Fig. 4 Precision variation according to T.

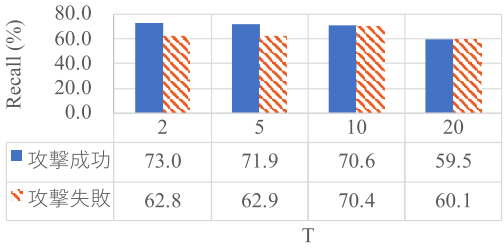

図 5 T による再現率の変化
Fig. 5 Recall variation according to T.

表 10 T による IOC 数の変化
Table 10 Number of IOC variation according to T.

T	2	5	10	20
全 IOC 数	78,662	76,990	68,766	50,134

T の増加に対して，適合率・再現率が単調増加あるいは単調減少にならない理由は IOC の数によるものである．**表 10** より，T と全 IOC 数が反比例していることが分かる．全 IOC 数とは全攻撃コードから抽出した IOC の数である．T が $[0, 5)$ と小さ過ぎるときは不適切な IOC が除外されないため，精度が低い．逆に T が $(10, 20]$ と大き過ぎるときは正しく判定するのに寄与する IOC が除外されてしまい，精度が落ちてしまった．

5.4 事例紹介

D_{real} に対して攻撃成功と判定できた事例について紹介する．今回の実験で攻撃成功と判定できたリクエストは 90 件であるが，攻撃カテゴリ別に分類すると以下の 3 つの事象に分けることができた．事例を示すにあたって，実システムの特定を避けるため，表現を一部置き換えている．

- **事例 1**： SQL インジェクションの脆弱性有無を確認する攻撃
- **事例 2**： WordPress の設定ファイルの漏えい
- **事例 3**： ログインフォームに対する XSS

事例 1 では**図 6** に示す攻撃コードが送られていた．この攻撃コードが実行された場合，攻撃コードの SELECT 文が指定した値が出力される．攻撃コードでは

[*9] https://www.nngroup.com/articles/website-response-times/

```
UNION SELECT CHAR(45,120,49,45,81,45),CHAR
(45,120,50,45,81,45),CHAR(45,120,51,45,81,45),CHAR
(45,120,52,45,81,45), ... -- /*
```

図 6 SQL インジェクションの攻撃コード
Fig. 6 Attack code of the SQL injection.

```
<table>
  <tr><th class="style_th">entry1</th><td class=
  "style_td">-x5-Q-</td></tr>
  <tr><th class="style_th">entry2</th><td class=
  "style_td">-x6-Q-</td></tr>
  ...
  <tr></tr>
</table>
```

図 7 SQL インジェクション攻撃に対する HTTP レスポンス
Fig. 7 HTTP response of SQL injection attack.

```
GET //wp-config.php~ HTTP/1.1
```

図 8 WordPress 設定ファイルのバックアップに対するリクエスト
Fig. 8 Request for backup file of WordPress configuration.

```
/* WordPress Database Table prefix. */
$table_prefix  = 'wp_';
define('WP_DEBUG', false);
if ( !defined('ABSPATH') )
  define('ABSPATH', dirname(__FILE__) . '/');
```

図 9 WordPress 設定のバックアップファイルアクセスのレスポンス
Fig. 9 Response for backup file access of WordPress configuration.

CHAR 関数によって難読化が施されているが，たとえば `CHAR(45,120,49,45,81,45)` は文字列`-x1-Q-`を表す．つまり，SQL 文が実行された場合は上記の CHAR 関数が実行されて，その結果の文字列が出力される．**図 7** に示すこの攻撃に対する HTTP レスポンスを見ると，SQL 文を実行した後の文字列が出力されているので，攻撃が成功していると判定した．今回の攻撃コードは脆弱性の存在を調査するものであり，被害を及ぼすことはないが，パスワードを漏えいさせる等の攻撃に容易に転用できるため重大インシデントにつながる．

事例 2 では，攻撃者は**図 8** に示すように WordPress 設定ファイルのバックアップが置かれていないかをチェックする攻撃を行っている．これに対して HTTP レスポンスでは**図 9** に示す内容を返しており，設定ファイルが漏えいしていると判定した．設定ファイルにパスワード等の機微情報が設定されている場合，パスワード漏えいという重大インシデントにつながってしまう．

事例 3 では**図 10**，**図 11** に示すように攻撃者はある ASP スクリプトの `lang` というパラメタに `script` タグを挿入している．挿入されたタグは HTTP レスポンスにそのまま出力されているため，攻撃が成功したと判定した．

```
GET /*.asp?lang="><script >alert(String.
fromCharCode(88,83,83))</script> HTTP/1.1
```

図 10 ログインフォームに対する XSS 攻撃
Fig. 10 XSS attack against login form.

```
<form method="POST" action="*.asp" name="login">
<input type="hidden" name="lang" value="
 "><script >alert(String.fromCharCode(88,83,83))</script> ">
</form>
```

図 11 ログインフォームに対する XSS 攻撃のレスポンス
Fig. 11 Response for XSS attack against login form.

これらの事例は提案手法を導入して初めて発見したものであり，WAF アラート通知では調査・分析まで至らなかった．

6. 制約

アプローチの特性上，提案手法には以下の攻撃の成否を判定できないという制約が存在する．

- **出力が存在しない攻撃**：提案手法では攻撃が成功したと判定できる指標を用いることで攻撃成否を判定する．しかし，攻撃によっては成功しても指標が出力されない，あるいは存在しない場合もある．
- **未対応のエミュレーション環境への攻撃**：提案手法では可能な限り対象サーバと同等のシステム環境を用いて攻撃をエミュレートする．しかし，特別な製品を用いている場合，エミュレーション時に IOC を抽出できても，対象のシステム環境の出力がまったく異なるため，正しく判定できない．
- **外部通信を行う攻撃**：提案手法では外部への攻撃を避けるためにエミュレーション時はネットワーク通信を遮断している．そのため，攻撃コードが外部のリソースを取得してから実行される場合，正しく判定できない．

7. 関連研究

著者らが既存の研究を調べた限りでは既存の攻撃に対するその成否を判定する手法は，大きく 4 つのアプローチに分けられる [10]．

- **HIDS（Host-based IDS）**：HIDS は OS やアプリケーションが出力する情報に基づいて攻撃検知を行うため，攻撃を正しく検知した際は攻撃が成功した後という可能性が高い．そのため，HIDS がアラートを通知している際は攻撃を成功したと判定していると考えることができる．システムコールの順序や種類を特徴とする検知手法 [1] が一般的であるが，システムコールだけでなく，DB アクセス等のアプリケーションレ

イヤーの情報も用いる検知手法 [2] も提案されている．導入にはシステムコール等を観測する仕組みが必要であるため，商用環境では動作保証が問題となり導入障壁が大きい．

- **SIDS（Stateful IDS）**：SIDS では攻撃が成功した際の状態を定義し，攻撃があった際，あらかじめ定義された状態に至ったかどうかで攻撃の成否を判定する [11]．たとえば，Vigna らの手法 [6] では攻撃の状態遷移に基づいて攻撃の成否を判定する．Sommer らの手法 [7] および Zhou らの手法 [8] は既存の IDS のシグネチャを活用して，シグネチャマッチをネットワーク通信に対して複数箇所で行うことで，攻撃の成否を判定する．また脆弱性自体の処理をとらえる研究 [12], [13] も存在する．問題点として，攻撃状態をルールで定義したり，アプリケーションの脆弱な箇所の動作をルールで定義したりする必要があり，利用者に要求される知識レベルが高い点があげられる．ルールを自動的に生成する手法 [14], [15] も提案されているが，様々なアプリケーションに対する攻撃の成否ラベルを付与した通信の学習データを用意することも必要である．提案手法では利用者に攻撃カテゴリの固有表現を求めるが，入手が容易な OS コマンドの一覧や関数の一覧等を設定するのみであり，SIDS が求める攻撃の状態や脆弱性の処理方法といった攻撃や脆弱性に関する特有な情報ではない一般的な情報であるため，SIDS と比べて，比較的低い知識レベルの利用者でも運用可能である．
- **CIDS（Correlation-based IDS）**：CIDS では IDS のアラートと脆弱性スキャナの情報を関連付けることで攻撃対象のサービス・アプリケーションが脆弱性を持っているかどうかで攻撃の成否を判定する [16], [17], [18]．たとえば，Kruegel らの手法 [16] では IDS のアラートに含まれる CVE 番号と脆弱性スキャナで発見した脆弱性の CVE 番号が同一かどうかで関連付けを行う．この手法では Web アプリケーションや Web サーバに脆弱性が存在するバージョンを利用していないか確認し，その脆弱性に対する攻撃を検知した場合は有効な攻撃と判定する．近年の SIEM（Security Information and Event Management）製品でも同様の判定アルゴリズムを実装しているものも存在する．SIDS と同様に脆弱性スキャナのルールの更新，IDS シグネチャとの関連付けのための情報の付与が必要になり，多様な攻撃に対応することが難しい．
- **EIDS（Emulation-based IDS）**：EIDS は攻撃があった際に，その攻撃コードの挙動をエミュレートし，実行する機械語命令列を抽出する手法である [3], [4], [19]．ホスト内でも同じ機械語命令列の実行を観測した場合，攻撃が実行されたとしてアラートを出力することで攻撃が成功したときのみを判定することが可能となる．これらの手法は x86 アセンブリの攻撃コードのエミュレーションに限られるが，Web アプリケーションに対しては，コードインジェクションの攻撃に特化した WeXpose [5] が存在する．ただ，これらの手法ではエミュレーション後の挙動を実システム内で観測する必要があるため，HIDS と同様に導入先の改変が必要である．

表 11 関連研究との比較
Table 11 Comparison with related works.

	HIDS	SIDS	CIDS	EIDS	本研究
要件 1	✗	✓	✓	✗	✓
要件 2	✓	✗	✗	✓	✓

本研究と関連研究の差異をシステム改変が不要（**要件 1**）および比較的低い知識レベルの利用者でも運用可能（**要件 2**）という 2 つの観点で比較した結果を**表 11** に示す．本研究は既存研究では満たせない 2 つの観点要件を満たす，新しい研究である．

8. おわりに

大量のアラートから重要なインシデントに関わるアラートを人手で探し出すには多くの時間を要する．本研究では，攻撃の成否に応じてアラートの重要度を決定する手法を提案した．提案手法では攻撃コードのエミュレーションを行い，攻撃の痕跡である IOC を抽出する．IOC が HTTP レスポンスに含まれるか否かで攻撃の成否を判定し，アラートの重要度を決定する．評価より 67.4%の割合（再現率）の Web 攻撃に対して，93.8%の精度（適合率）で判定可能であることを示した．また，約 98%の攻撃のエミュレーションにかかる時間が 1 秒以内であることから，実用的な結果が得られた．

参考文献

[1] Hofmeyr, S.A., Forrest, S. and Somayaji, A.: Intrusion detection using sequences of system calls, *Computer Security* (1998).

[2] 鐘　揚，佐藤　徹，谷川真樹：AETP：イベントの関連づけによる web アプリケーションに対する有効な攻撃の検知手法，コンピュータセキュリティシンポジウム 2016 論文集 (2016).

[3] Abbasi, A., Wetzels, J., Bokslag, W., Zambon, E. and Etalle, S.: On emulation-based network intrusion detection systems, *RAID* (2014).

[4] Polychronakis, M., Anagnostakis, K.G. and Markatos, E.P.: Network level polymorphic shellcode detection using emulation, *DIMVA* (2006).

[5] Bellizzi, J. and Vell, M.: Wexpose: Towards on-line dynamic analysis of web attack payloads using just-in-time binary modification, *SECRYPT* (2015).

[6] Vigna, G., Robertson, W., Kher, V. and Kemmerer, R.A.: A stateful intrusion detection system for world-wide web servers, *ACSAC* (2003).

[7] Sommer, R. and Paxson, V.: Enhancing byte-level net-

work intrusion detection signatures with context, *CCS* (2003).
[8] Zhou, J., Carlson, A.J. and Bishop, M.: Verify results of network intrusion alerts using lightweight protocol analysis, *ACSAC* (2005).
[9] Johns, M., Engelmann, B. and Posegga, J.: XSSDS: Server-Side Detection of Cross-Site Scripting Attacks, *ACSAC* (2008).
[10] Hubballi, N. and Suryanarayanan, V.: False alarm minimization techniques in signature-based intrusion detection systems: A survey, *Computer Communications* (2014).
[11] Cuppens, F. and Miège, A.: Alert correlation in a cooperative intrusion detection framework, *IEEE S&P* (2002).
[12] Wang, H.J., Guo, C., Simon, D.R. and Zugenmaier, A.: Shield: Vulnerability-driven network filters for preventing known vulnerability exploits, *SIGCOMM* (2004).
[13] Brumley, D., Newsome, J., Song, D., Wang, H. and Jha, S.: Towards automatic generation of vulnerability-based signatures, *IEEE S&P* (2006).
[14] Massicotte, F., Gagnon, F., Labiche, Y., Briand, L. and Couture, M.: Automatic evaluation of intrusion detection systems, *ACSAC* (2006).
[15] Massicotte, F., Labiche, Y. and Briand, L.C.: Toward automatic generation of intrusion detection verification rules, *ACSAC* (2008).
[16] Kruegel, C. and Robertson, W.: Alert verification - determining the success of intrusion attempts, *DIMVA* (2004).
[17] Massicotte, F. and Couture, M.: Context-based intrusion detection using snort, nessus and bugtraq databases, *PST* (2005).
[18] Valeur, F., Vigna, G., Kruegel, C. and Kemmerer, R.A.: A comprehensive approach to intrusion detection alert correlation, *IEEE DSC* (2004).
[19] 嶋村 誠，河野健二：Yataglass：攻撃の擬似実行による攻撃メッセージの振舞いの解析，情報処理学会論文誌 (2009).

推薦文

本論文は WAF や IDS 等で攻撃検知された Web 攻撃の HTTP リクエストとそのレスポンスのデータを入力として攻撃成否判定手法を提案している．実データに対する実装評価を行い 98%の攻撃リクエストに対して 1 秒以内に判定可能という実用的な提案であり大きく評価できる．よって推薦論文として推薦する．

（コンピュータセキュリティシンポジウム 2017（CSS2017）
プログラム委員長　須賀祐治）

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博士前期課程修了．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．