AVT Lite: 攻撃コードのエミュレーションに基づくWeb攻撃の成否判定手法

鐘　揚[1]　青木　一史[1]　三好　潤[1]　嶋田　創[2]　高倉　弘喜[3]

概要：WAF や IDS など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，攻撃の成否に応じてアラートの重大度を決定するシステムを提案する．提案システムでは攻撃コードのエミュレーションを行い，攻撃の痕跡を抽出する．攻撃の痕跡が HTTP レスポンスに含まれるか否かで攻撃の成否を判定し，アラートの重大度を決定する．提案システムの精度・性能の面での評価結果，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

AVT Lite: Detection Successful Web Attacks based-on Attack Code Emulation

Yang ZHONG[1]　Kazufumi AOKI[1]　Jun MIYOSHI[1]　Hajime SHIMADA[2]　Hiroki TAKAKURA[3]

Abstract: Security appliance such as WAFs and IDSs contribute to detecting threats of web attacks greatly. However, discovering security appliance alerts related to critical incident manually takes much time. In this research, we propose the system that verifies the emergency level of alerts based on success or failure of attacks. Proposed system emulates exploit code to extract indicators of compromise. Verification is made by matching indicators with HTTP response content. We show the effectiveness of proposed 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攻撃や脆弱性スキャンは日々，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信に限定的であったり，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 サイトでは，大量のアラートが通知されてしまう．アラートを確認し，重大な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であるため，大量のアラートの確認に多くの時間を要する．ここでいう重大とは実際にセキュリティ侵害 (改ざん，乗っ取り，漏洩等) が発生したことを指す．そのた

[1] NTT セキュアプラットフォーム研究所
NTT Secure Platform Laboratories
[2] 名古屋大学情報基盤センター
Information Technology Center, Nagoya University
[3] 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

© 2017 Information Processing Society of Japan

め，セキュリティオペレーションの現場では重要なアラートの判断が課題となっている．

既存研究では様々なアプローチでこの課題に取り組んでいる．WAF は一般的にネットワーク通信のリクエストのみを対象として攻撃検知を行う方式である NIDS (Network-based IDS) に分類される．NIDS は攻撃を検知できるが攻撃の成否は判断出来ない．これ対して，HIDS (Host-based IDS) はホスト内で攻撃検知を行うことから，攻撃が成功していることを検知できるが，ホスト内の挙動を観測するため，システム改変が必要となり，商用環境では導入障壁が大きい．SIDS (Stateful IDS) や CIDS (Correlation-based IDS) は 2 章で詳説するが，攻撃に対して細かなルールを作成する必要があり，様々な攻撃方法が存在する Web アプリケーションにとっては検知ルール更新していくコストが大きい．EIDS (Emulation-based IDS) は攻撃コードのエミュレーションを行うため細かな攻撃に対するルールを作成する必要はないが，既存研究 [7–9] では実ホストと同じ条件でその挙動を観測することが必要なため，実ホスト内で動作させる必要があり，HIDS と同じ欠点を持つ．

本研究では商用環境での利用を鑑み，システム改変が不要及び，攻撃個別の対応が不要という 2 つ要件を定義し，要件を満たす，攻撃の成否に応じてアラートの重大度を決定するシステム AVT Lite[*1]を提案する．AVT Lite 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．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，アラートの重大度を決定する．EIDS では攻撃コードのシステム内での挙動を特徴としていたが，AVT Lite では攻撃による出力を特徴とすることで，ホスト内を改変することなく攻撃の成否を判定できる．

2 章では攻撃成否判定に対する既存研究を紹介し，その課題を示す．3 章では提案システムの AVT Lite について述べる．5 章では AVT Lite に対するの精度・性能評価結果及び実際に攻撃成功を検知した事例について考察する．AVT Lite の制約を 6 章にて述べ，7 章にて本稿をまとめる．

2. 攻撃に対する成否判定手法

2.1 既存研究

著者らが既存の研究を調査し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．

- **HIDS：** HIDS (Host-based IDS) は古くから研究されており，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．例えば，STIDE はシステムコールの順序を特徴として用いる検知手法 [1] である．学習したシステムコールの順序と異なる順序が現れた場合，攻撃として検知する．近年，著者らはシステムコールのみならず，DB アクセスなどのアプリケーションレイヤーの情報も用いる検知手法 AVT [2] を提案した．

HIDS は NIDS と異なり，ホスト内の情報を用いるため，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．

- **SIDS：** SIDS (Stateful IDS) では攻撃が成功した際に発生する状態を定義し，攻撃があった際，予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．例えば，Vigna らの手法 [3] では攻撃の状態遷移に基づいて攻撃の成否を判定する．Robin らの手法 [4] 及び Zhou らの手法 [5] は既存の IDS のシグネチャを活用して，シグネチャのパターン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．
- **CIDS：** CIDS (Correlation-based IDS) では IDS のアラートと脆弱性スキャナの情報を関連づ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．関連付けの方法はで判断している．例えば，Kruegel らの手法 [6] 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断する．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．
- **EIDS:** EIDS (Emulation-based IDS) 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 [7, 8]．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 [9] が存在する．

2.2 要件及び課題

本研究では Web に対する攻撃の成否を判定する際に商用環境で必要な 2 つの要件を以下に示す．

- **要件 1：** システム改変が不要
- **要件 2：** 攻撃個別の対応が不要

システムを改変して導入する場合，動作保証が問題となるため商用環境では導入障壁が大きい．そのため，できるだけ既存のシステムを改変しないアプローチが望まれる (**要件 1**)．個々の攻撃に対して検知ルールを人手で定義する必要がある場合，更新にかかる運用コストも大きい．その

[*1] AVT: Alert Verification Technology

© 2017 Information Processing Society of Japan

ため，攻撃個別の対応が不要で更新が少ないことが望まれる (**要件 2**) ．

2.1 節で述べたアプローチを**要件 1**，**要件 2** について比較した結果を表 1 に示す．どのアプローチも 2 つの要件を同時に満たすことはできていない．本研究で提案する AVT Lite は EIDS のアイデアに基づいて，EIDS でホスト内の挙動の検査を行う部分をネットワーク通信に適用することでシステムを改変することなく検査を行えるように改良した．3 章にて提案システムを詳しく述べる．

表 1: 既存手法の課題
Table 1 Conventional methodology evaluation

	HIDS (AVT)	SIDS	CIDS	EIDS
要件 1	✗	✓	✓	✗
要件 2	✓	✗	✗	✓

3. 提案システム

本稿ではシステムへの改変をせず (**要件 1**)，かつ攻撃個別の対応が不要 (**要件 2**) な Web アプリケーションに対する攻撃の成否を検証するシステム AVT Lite を提案する．全体像を図 1 に示す．AVT Lite には大別して 4 つのステップが存在する．

AVT Lite では WAF や IDS 等で攻撃検知された HTTP リクエスト及び HTTP レスポンスのログを入力とする．攻撃検知は例えば，既存のシグネチャ検知[*2]のツールである Snort[*3]や ModSecurity[*4]を利用したり，通常のアクセスとは異なる特徴を示すリクエストを検知するアノマリ検知の手法 [10, 11] 等を利用する．AVT Lite では攻撃検知された HTTP リクエストに対して，攻撃カテゴリの判定及び攻撃コードの抽出を行う (**Category Identification**) ．攻撃カテゴリの判定とはコマンド等を実行する攻撃なのか，あるいは SQL 文を実行して DB を悪用する攻撃なのかを判定することである．この時，攻撃コードを発見できた場合，その攻撃コードを仮想環境 (エミュレータ) で実行をエミュレートする (**Code Emulation**) ．エミュレートの結果，出力される表示を攻撃コードが実行された際の指標である IOC として抽出する (**Indicator Extraction**) ．最後に，抽出された IOC を実際の HTTP レスポンスと比較することで，攻撃が成功したか否かを判断する (**Alert Verification**) ．以降，各節にて各処理を詳しく説明する．

[*2] 事前に攻撃コードをパターン化した情報を持っておき，ネットワーク通信がそのパターンにマッチしたかどうかで攻撃を検知する方式である
[*3] https://www.snort.org/
[*4] https://www.modsecurity.org/

3.1 Category Identification

表 2 に示す 4 つのカテゴリにその攻撃を分類する．攻撃カテゴリを判別する理由は次ステップ (**Code Emulation**) 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 であれば CentOS や FreeBSD など，ランタイムであれば PHP や Java など，DB であれば MySQL や PostgreSQL など多様にあり，攻撃コードのエミュレーションを各環境でそれぞれ実施するため処理に時間がかかる．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，RCE カテゴリにはサブカテゴリが存在し，OS コマンドあるいはプログラミング言語 (PHP, Python 等) の差異を区別するために利用する．固有表現の収集では OS コマンドであれば，`/bin` や`/usr/bin` 以下のプログラム名及び，`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では関数名の一覧等を利用する．SQLI (SQL Injection) カテゴリの判定には DB にアクセスする際に使用する SQL 文の SQL コマンド及びビルトイン関数が存在するかどうかによって判断する．PT (Path Traversal) カテゴリは相対参照をする際に必要な`../`といった表現，及び Web アプリケーションとは直接関連しないパスの指定 (`/etc`，`/proc`) の有無によって判断する．XSS (Cross-Site Scripting) カテゴリの判定には HTML タグの表現の有無によって判断する．表 3 に各カテゴリに判別するための固有表現の一例を示す．判定する際，攻撃コードに対する URL エンコード等は全てデコードされている状態を前提としている．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．

表 2: 攻撃カテゴリ一覧
Table 2 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	表示 HTML を悪用する攻撃

3.2 Code Emulation

攻撃カテゴリに応じて攻撃コードを実行する仮想環境 (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 (例えば `sleep`

© 2017 Information Processing Society of Japan

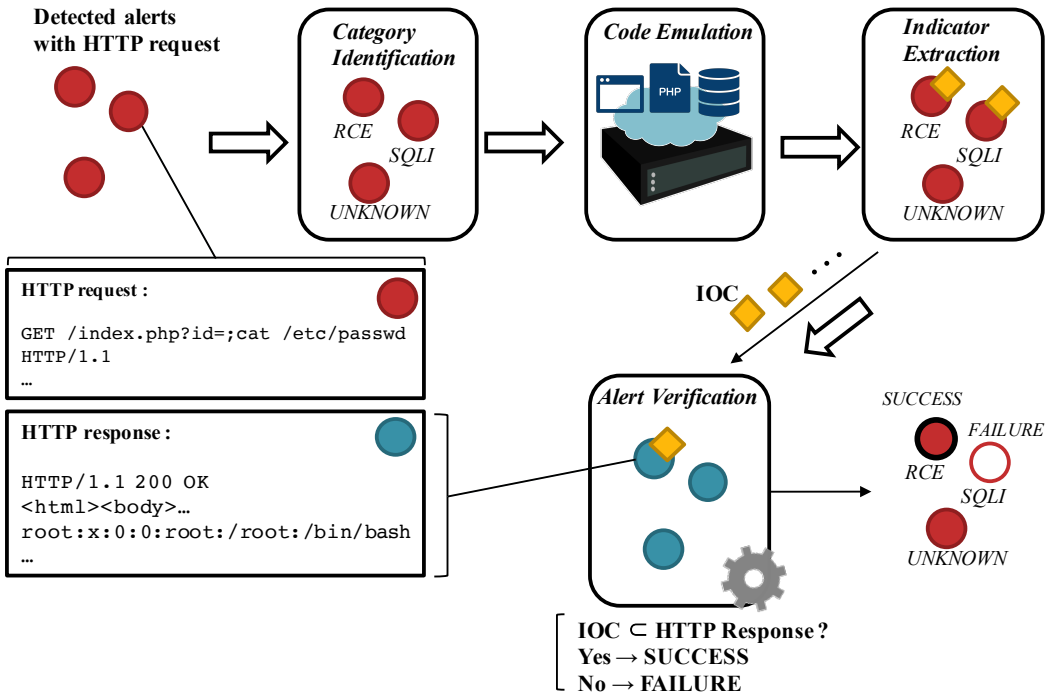

図 1: AVT Lite の各処理の概要
Fig. 1 Overview of AVT Lite

表 3: 攻撃カテゴリを判別するための固有表現の例
Table 3 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, concat()`
PT		`../, /etc/, /proc/`
XSS		`<script>, <iframe>`

コマンドの実行) を避けるため，エミュレーションにタイムアウトが設定されている．

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デフォルト環境の DB サーバを用意し，攻撃コードの 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．XSS カテゴリの攻撃に対しては攻撃コードがそのまま出力されるので，エミュレートを実行しない．

エミュレータ環境は幅広い攻撃に対応するため様々なアプリケーションを搭載する必要があり，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 2 つの条件が必要となる．

(1) プログラムを実行した後でも実行前の状態に戻す (スナップショット) 機能

(2) スナップショットされた環境を高速に復元する機能

4 章で詳細に説明するが，AVT Lite では QEMU[*5] ような OS の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 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．そのため AVT Lite では複数環境でエミュレーションを行うこと及び，次ステップの **Indicator Extraction** で抽出された IOC を正規化することで，可能な限り実際の環境とエミュレーション環境の差異を減少させることを行っている．

3.3 Indicator Extraction

エミュレータ環境で攻撃コードを実行すると通常は実行結果が標準出力あるいは標準エラー出力に出力される．Web サーバでは一般に標準出力を HTTP レスポンスとしてクライアントに送信し，標準エラー出力をログとしてシステム内に残す仕組みになっている．よって，標準エラー出力が HTTP レスポンスに残ることは可能性は低いため，標準出力の内容のみを攻撃が成功したことを示す指標である IOC として利用する．

AVT Lite ではまず，標準出力に出力された全ての内容を 1 つの IOC 候補として保持する．しかし，この IOC 候補を HTTP レスポンスとそのまま比較する場合，Web ア

[*5] `http://www.qemu.org/`

© 2017 Information Processing Society of Japan

プリケーションの表示の仕様によって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．そのため，以下に示す 2 つの整形処理を標準出力に対して施す．この処理により，Web アプリケーションの表示の制約等によって攻撃が成功した際の見逃しを避ける．

(1) 1 つの IOC 候補が複数行に跨がる場合は，それを各行に区切り，複数の IOC 候補とする
(2) 1 つの IOC 候補がタブ文字で区切られている場合は，それをタブ文字で区切り，複数の IOC 候補とする

また，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も行う．以下に除外条件を示す．

(1) IOC の文字列長が N 以下の場合除外する (N は予め指定する)
(2) 一般的に利用される文字列の場合除外する (例：`<html>`，`<head>`，`<body>`等)

最後に，実際の環境とエミュレーション環境の差異を減少するため，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，数字等を正規表現等に変換する．例えば，`last` コマンドを実行した時の IOC 候補は以下のように変換される．

```
wtmp begins 15:55:59 2017
⟶ wtmp begins \d+:\d+:\d \d+
```

上記の処理を施した後，残った IOC 候補を攻撃の成否検証に利用する 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．

3.4 Alert Verification

検証のステップでは抽出された IOC とそれに対応する HTTP リクエストのレスポンスに IOC が含まれているかどうかで攻撃が成功したかどうかを判定する．IOC が含まれている場合，攻撃コードが実行されたことを示しているため，攻撃が成功したと判断する．そうでない場合，攻撃が失敗したと判断する．

4. 実装

AVT Lite のシステム全体像を図 2 に示す．AVT Lite はまず syslogd を介して通信ログや検証結果の通知を行う．攻撃カテゴリの判定及び攻撃成否の検証は独自の Python プログラムによって実施している．攻撃コードをエミュレートする環境には Docker[*6] を採用した．Docker は Linux コンテナ技術を活用しており，ハイパーバイザ等に比べ起動・復元等が高速に行える．

AVT Lite システムでは攻撃が発生する度，Docker を用いてエミュレーション環境を作成し，その環境内で攻撃コードが実行される．なお，エミュレーション時に外部への攻撃を発生させないため，ネットワーク通信は全て遮断した状態でエミュレーションを行っている．この設定による影響については 6 章にて述べる．

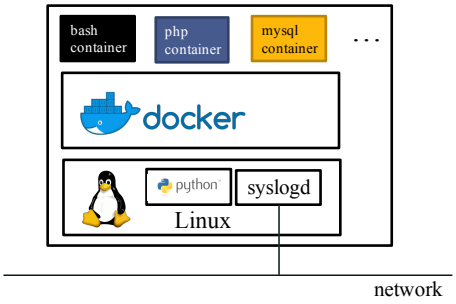

図 2: AVT Lite システム構成
Fig. 2 AVT Lite system architecture

5. 評価

AVT Lite に対して判定精度及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 4 に示す．精度評価には仮想環境で人工的に作成したデータセット LAB 及び大規模実ネットワーク環境から収集したトラフィックから生成したデータセット REAL を利用した．性能評価にはデータセット REAL を利用した．データセット LAB の作成には脆弱性検証等に用いられる Web アプリケーションである DVWA[*7] を利用した．DVWA は設定によって脆弱性がある状態と脆弱性がない状態を作り出せるため，両方の状態に対して同じ攻撃を行った．攻撃に利用した攻撃コードは Web 攻撃に頻繁に出現するものを選択した．データセット LAB に含まれる攻撃カテゴリ毎のリクエスト数と攻撃コードの内容を表 5 に示す．データセット REAL の攻撃検知には WAF の検知結果及び，独自シグネチャを用いて判定を行った．

表 4: データセット概要
Table 4 Dataset overview

データセット	LAB	REAL
観測環境	仮想環境	実環境
観測期間	–	Feb 2017 – Jun 2017
データ数	88	166,542
攻撃成功数	44	–
攻撃失敗数	44	–
送信元 IP アドレス数	–	5,059
送信先 IP アドレス数	–	752

[*6] https://www.docker.com/
[*7] www.dvwa.co.uk/

© 2017 Information Processing Society of Japan

表 5: データセット LAB の攻撃カテゴリ別攻撃内容とデータ数
Table 5 Attack codes detail for each attack category of dataset LAB

カテゴリ	個数	説明
RCE	10	`cat`，`ls`，`uname`，`whoami`，`last`，`netstat` 等の情報を出力するコマンドの実行及び，`curl`，`wget` 等の外部接続を行うコマンドの実行
SQLI	13	SELECT 文による DB のバージョン，ユーザ名，DB 名，ホスト名，テーブル名の取得及び，INSERT 文，UPDATE 文による DB の改変
PT	10	`/etc`，`/proc` 以下のファイルの取得
XSS	11	`script`，`img`，`body`，`a`，`embed` 等のタグを利用した JavaScript コードの実行

5.1 精度評価

データセット LAB を用いて AVT Lite の攻撃成否判定の精度評価を行った．評価には成功する攻撃に対する適合率・再現率及び，失敗する攻撃に対する適合率・再現率の 4 つの指標を用いた．ここでいう適合率とは AVT Lite が検証可能と判断したデータに対する検証結果が正しいデータの割合である．また，再現率は指標ラベルに対して，同じ検証結果を出した場合の割合である．つまり，以下の通り定義できる．

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

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

評価結果を表 6 に示す．成功判定に対する適合率は 100%であるから，AVT Lite で攻撃が成功したと判定できた場合，そのアラートは必ず重大なインシデントであることを示す．またデータセット全体に対しても約 90%の適合率を示すことから，検証可能な攻撃については正しく成否が判定できている．一方，再現率は成功判定・失敗判定どちらも低く約 60%程度である．しかし，攻撃全体に対して約 60%の攻撃を正しく検証できることから，セキュリティ機器から通知される半数以上のアラートは AVT Lite によって自動的に検証され，更に詳細な分析が必要かどうかが決定される．つまり，セキュリティオペレーションを行う分析官にとっては Web に対する脅威という面では約半分の稼働削減につながる．一方約 39%の攻撃の成否は検証できないあるいは検証結果が正しくない．その理由については 5.3 節にて議論する．

次に，データセット REAL に対して AVT Lite を適用した時の結果を表 7 に示す．成功と判定した件数は 122 件であった．この 122 件には重複した攻撃が含まれており，実際は 2 種類の攻撃であった．実際に HTTP レスポンスを確認し，確かに攻撃コードが実行され攻撃が成功したと推測できる痕跡が出力されていることを確認した．そのため，攻撃が成功する際の適合率が 100%であることが実データをにおいても示された．検知した事例は 5.4 節にて紹介する．

表 6: データセット LAB に対する検証結果
Table 6 Evaluation result for dataset LAB

項目		値
適合率	攻撃成功	100.0% (24/24)
	攻撃失敗	83.3% (30/36)
	全体	90.0% (54/60)
再現率	攻撃成功	54.5% (24/44)
	攻撃失敗	68.2% (30/44)
	全体	61.3% (54/88)

表 7: データセット REAL に対する検証結果
Table 7 Evaluation result

AVT Lite 判定結果	個数	割合 (%)
成功	122	0.07
失敗	100,167	60.15
判定不可	66,253	39.78

5.2 性能評価

性能評価では AVT Lite の各攻撃リクエストに対する検証にかかる時間を測定し，評価を行った．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，8GB RAM，512GB SSD である．検証にかかる時間とは，Category Identification のステップ (3.1 節) から Alert Verification のステップ (3.4 節) までの処理時間である．

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 0.5 秒以内に終了しており，1 秒間に 2 リクエストまでは遅延なく処理できる．例えば WAF と Web サーバ間に AVT Lite を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してレスポンス時間が約 0.5 秒程度の遅延する程度に抑えられる．つまり，WAF 等での攻撃検知が誤って AVT Lite での処理が行われても，レスポンス遅延が 0.5 秒程度に抑えられる．一般的に 1 秒程度の遅延であればユーザビリティは大きく損なわれない[*8]ため，AVT Lite は性能面でも実用的であると考える．

5.3 分析

AVT Lite の検証結果が正しくなかった原因について分析を行った．データセット LAB の指標と AVT Lite の判定結果を表 8 に示す．成功した攻撃を失敗判定した 6 件と

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

© 2017 Information Processing Society of Japan

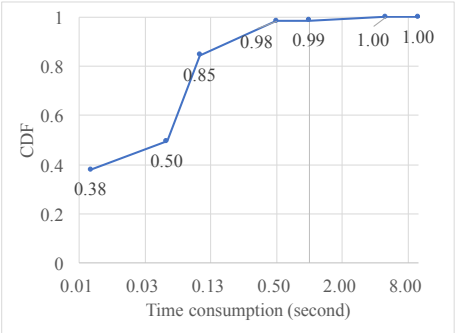

図 3: 処理時間の累積分布関数
Fig. 3 CDF of time consumption

表 8: 判定結果
Table 8 Evaluation result

指標 / 判定結果	成功 (44)	失敗 (44)
攻撃成功	24	0
攻撃失敗	6	30
判定不可	14	14

判定不可となった攻撃の 14 件についてその原因を分析し，以下の 2 つに帰着できた．

- **原因 1**： 実環境とエミュレータ環境の違いによってマッチする IOC とならなかった
- **原因 2**： 出力を行う攻撃コードでないため，IOC が抽出されなかった

原因 1 については例えば，攻撃対処サーバの情報を調べるために `whoami` コマンドの実行や，`SELECT version();` のような SQL インジェクション攻撃の場合，出力がエミュレータと実際の環境で異なる場合が存在するため，攻撃成否判定を誤ることがある．出力が異なる例を表 9 に示す．

表 9: エミュレータと実環境の差異例
Table 9 Evaluation result

攻撃コード	エミュレータ	実環境
`whoami`	`root`	`apache`
`SELECT version();`	`10.2.7-MariaDB maria-jessie|`	`5.5.52-MariaDB`

ユーザ名を表示する `whoami` コマンドの場合，実際の Web サーバでは `apache` や `www-data` などの Web サーバのプロセスを起動しているユーザが出力となるが，エミュレータでは `root` ユーザでコマンドを動作させているため差異が生じ，検証が正しく行われない．SQL インジェクションの場合は利用している DB のバージョンが異なるため，検証の結果を誤ってしまう．そのため，今後は対象 Web サーバ等の環境が事前に把握できる場合はそのバージョンと同じものをエミュレータで用いることで差異を無くすことが可能である．また，Web サーバ環境が事前に把握できない場合は複数のバージョンを用意し，複数のバージョンを並行してエミュレーションすることで実環境とエミュレータ環境の差異を減少させれると考える．

原因 2 を引き起こすのは Web サーバ内の情報を抜き出す攻撃でなく，改変を行う場合の攻撃が大多数である．例えば，`crontab` コマンドを利用して定期的に別のプログラムを実行する，あるいは `UPDATE` や `INSERT` コマンドを利用して DB を改変する場合も攻撃が成功したとしても，一切の出力を行わないため，AVT Lite の方式では対応できない．これらの攻撃を検証するためにはホスト内部で攻撃の成否を検証するアプローチ [2] が必要となる．

5.4 事例紹介

データセット REAL に対して攻撃成功と判別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 122 件であるが，攻撃カテゴリ別に分類すると以下の 2 つの事象に分けることが出来る．

- **事例 1**： URL パラメタに対して SQL インジェクションの脆弱性がないかを確認する攻撃
- **事例 2**： WordPress の設定ファイルの漏洩

事例 1 の SQL インジェクションの事例では図 4 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 CHAR 関数によって難読化が施されているが，例えば `CHAR(45,120,49,45,81,45)` は文字列`-x1-Q-`を表している．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 5 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考える．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，攻撃コードの SQL 文を変更することで，DB の管理者のパスワードを漏洩させる等，容易に実行できるため重大なインシデントにつながる．

```
1 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 4: SQL Injection の攻撃コード

事例 2 の WordPress の設定ファイルの漏洩では，攻撃者は図 6 に示すように WordPress の設定のバックアップファイル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 7 に示す内容を返しており，設定ファイルが漏洩していると判定できる．また，図 7 に示さないが，実際に設定ファイルに記された機微情報も出力されていることを確認した．

© 2017 Information Processing Society of Japan

```
<body>
  <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><th class="style_th">Entry3</th><td class="style_td">-x7-Q-</td></tr>
    <tr><th class="style_th">Entry4</th><td class="style_td">-x9-Q-</td></tr>
    <tr></tr>
  </table>
</body>
```

図 5: SQL Injection 攻撃に対する HTTP レスポンス

```
GET //wp-config.php~ HTTP/1.0
```

図 6: WordPress 設定ファイルのバックアップファイルに対するアクセス

```
/**
 * WordPress Database Table prefix.
 * You can have multiple installations in one database if you give each
 * a unique prefix. Only numbers, letters, and underscores please! */
$table_prefix  = 'wp_';
define('WP_DEBUG', false);
/* That's all, stop editing! Happy blogging. */
/** Absolute path to the WordPress directory. */
if ( !defined('ABSPATH') )
  define('ABSPATH', dirname(__FILE__) . '/');
```

図 7: WordPress 設定バックアップファイルアクセスの HTTP レスポンス

6. 制約

アプローチの特性上，AVT Lite には以下の制約が存在する．

- **出力がない攻撃**：AVT Lite 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の成否を検証する．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．例えば，攻撃コードによって以下の OS コマンドを実行する場合，

```
echo '<?php phpinfo();' > /var/www/index.php
```

攻撃が成功しても何ら指標はないが，Web サイトのトップページが改ざんされることとなる．

- **外部リソースを利用する攻撃**：AVT Lite 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，正しく処理できない．
- **特殊環境に対する攻撃**：AVT Lite 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，特殊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全く異なるため，正しく検証できない．

7. おわりに

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，攻撃の成否に応じてアラートの重大度を決定するシステム AVT Lite を提案した．AVT Lite 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，アラートの重大度を決定する．判定精度は 90%以上であり，ほぼ全て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．今後の課題は，エミュレーション環境の多様化及び，最適な IOC の抽出方法が挙げられる．

謝辞　本研究のテーマ選定にあたりご協力頂いた NTT セキュリティ・ジャパン技術部，岸川雄輝氏に感謝する．

参考文献

[1] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. 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] Giovanni Vigna, William Robertson, Vishal Kher, and Richard A. Kemmerer. A stateful intrusion detection system for world-wide web servers. In *ACSAC*, 2003.
[4] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *CCS*, 2003.
[5] Jingmin Zhou, Adam J. Carlson, and Matt Bishop. Verify results of network intrusion alerts using lightweight protocol analysis. In *ACSAC*, 2005.
[6] Christopher Kruegel and William Robertson. Alert verification - determining the success of intrusion attempts. In *DIMVA*, 2004.
[7] Ali Abbasi, Jos Wetzels, Wouter Bokslag, Emmanuele Zambon, and Sandro Etalle. On emulation-based network intrusion detection systems. In *RAID*, 2014.
[8] Michalis Polychronakis, Kostas G. Anagnostakis, and Evangelos P. Markatos. Network level polymorphic shellcode detection using emulation. In *DIMVA*, 2006.
[9] Jennifer Bellizzi and Mark Vell. Wexpose: Towards online dynamic analysis of web attack payloads using just-in-time binary modification. In *SECRYPT*, 2015.
[10] Christopher Kruegel and Giovanni Vigna. Anomaly detection of web-based attacks. In *CCS*, 2003.
[11] Yang Zhong, H. Asakura, H. Takakura, and Y. Oshima. Detecting malicious inputs of web application parameters using character class sequences. In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, 2015.

© 2017 Information Processing Society of Japan