情報処理学会論文誌 Vol.61 No.5 1080–1091 (May 2020)

HIDSアラート調査のための HTTPリクエストとホストイベントの関連付け手法

鐘本 楊[1,2,a)] 青木 一史[1] 三好 潤[1] 小谷 大祐[3] 岡部 寿男[3]

受付日 2019年10月2日, 採録日 2020年2月4日

概要：攻撃ツールの進化によりWebアプリケーションに対する攻撃数は増加の一途をたどっている．IDSはこれらの攻撃を検知し，システム管理者に通知する役割を担っている．IDSはその形態からネットワーク型（NIDS）およびホスト型（HIDS）に大別される．HIDSはホスト上で観測できる細かなイベントに基づいて攻撃の成否を判定できるためより精度が高い通知が可能である．しかし，システムコールやデータベースへのSQLクエリ発行の情報のみを入力として利用しており，これらの情報がどのHTTPリクエストによって発生したものであるか関連付いていない．そのため，被害の原因調査に必要な攻撃対象のWebアプリケーションのURLや攻撃元などの情報を出力できず，管理者がこれらの情報を手動で特定する必要があり，時間を要する．本研究では，HIDSの入力であるシステムコールやSQLクエリ発行などのイベントをそれらを発生させたHTTPリクエストを処理したスレッドのIDと高精度な処理開始および終了時刻に基づいて関連付けを行うことで，HIDSで検知した際に管理者が攻撃対象のWebアプリケーションのURLや攻撃元のIPアドレスを特定できるようにする．評価では，提案手法が誤った関連付けをすることがなく，Webアプリケーションに与えるパフォーマンス低下を5%程度に抑えた実用的な手法であることを示す．

キーワード：Webセキュリティ，HIDS，イベント関連付け，システムコール

Correlating HTTP Request with Host Events for Efficient Host based Intrusion Detection Alert Analysis

Yo Kanemoto[1,2,a)] Kazufumi Aoki[1] Jun Miyoshi[1] Daisuke Kotani[3] Yasuo Okabe[3]

Received: October 2, 2019, Accepted: February 4, 2020

Abstract: The number of attacks against web applications has been increasing due to the evolution of attack tools. IDS is responsible for detecting these attacks and notifying system administrators. IDS is roughly classified into two types: network type (NIDS) and host type (HIDS). NIDS is easy to deploy, but the number of alerts becomes large because NIDS send alerts when an attack was failed too. Since HIDS only notifies when the attack is successful, more accurate notification is possible. However, it is not possible that HIDS outputs information such as the URL of the target web application or the attack source that is necessary for investigating the cause of the attack, because HIDS uses only system call and SQL query which is not correlated to which HTTP request. Therefore, administrators need to identify this information manually, which takes time. In this paper, we propose a method to correlate system calls and SQL query with HTTP requests. To do so, when HIDS detected an abnormal system call or an abnormal SQL query, the administrator can identify information related to the attacked web application. The evaluation results show the proposed method is practical because it achieves no incorrect correlation and only 5% performance degradation.

Keywords: Web security, HIDS, event correlation, system call

[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] 京都大学学術情報メディアセンター
Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606–8501, Japan
[a)] yo.kanemoto.zx@hco.ntt.co.jp

1. はじめに

Webアプリケーションは外部に公開されているため，つねに攻撃の脅威にさらされている．攻撃者は継続的にWebアプリケーションの脆弱性を発見するために，ボットを活用したスキャン [1] やGoogle Dorking [2] を行っているため，Web空間では攻撃が日々大量に発生している．世界でも有数の大規模CDN（Contents Delivery Network）事業者であるAkamaiの観測によると，2015年には13億件だった攻撃数が，2017年には35億件へと増加しており，近年では1日あたり1,000万件以上もの攻撃が発生している[*1]．

攻撃検知に用いられるIDS（Intrusion Detection System）はその利用形態から大きくネットワーク型（NIDS）およびホスト型（HIDS）の2つに大別される．WAF（Web Application Firewall）を含むNIDSはサーバの改変を必要としないため導入しやすく，Webアプリケーションに対する攻撃を検知するために広く利用されている．NIDSの問題はアラート数が膨大になってしまうことである．この原因は2つある．1つ目はNIDSとHIDSに共通する課題であるが，正常アクセスに対する誤検知である．誤検知を低減する手法が古くから多くの研究者によって提案されている [3], [4], [5], [6]．2つ目は攻撃ではあるが失敗した攻撃の検知である．一般にNIDSは攻撃の成否を判定することができないため，失敗した攻撃の検知を減らすことは困難である．一方，HIDSはホスト内の情報を活用して攻撃検知を行うシステムである．HIDSは攻撃が成功してサーバに対して影響を与えている状態を異常としてとらえることができる．HIDSとNIDSを比較した場合，HIDSは失敗した攻撃を検知せず，攻撃成功時のみを検知する可能性が高いため，NIDSより精度の高い検知が可能と考える．HIDSには，ファイルの完全性を確認することで改ざんを検知する仕組みなどホスト内の様々な情報を活用するものがあるが [7]，本論文では，OSのシステムコールやDBに対するSQLクエリ発行の情報をもとに攻撃検知を行うHIDSを対象として考える．

IDSがアラートを通知すると，管理者は攻撃による被害有無の確認や被害が再発しないよう暫定対処を行うための情報を収集する調査を行う．ここでいう暫定対処とはたとえば，攻撃の標的となったWebアプリケーションのURLに対するアクセスを禁止したり，攻撃の送信元となっているIPアドレスを遮断したりなど根本的な解決ではないが，一時的に同じ攻撃によって再度被害が発生しないようにするための処置である．そのため，アラートの調査では標的アプリケーションのURLや送信元IPアドレスなどの情報を含むHTTPリクエストの情報が必要である．HIDSではシステムコールやSQLクエリを始めとするホストに関する情報を入力とするため，改ざんされたファイルや実行されたコマンドなどの被害については出力可能であるが，対象のWebアプリケーションのURLや送信元IPアドレスなどの情報を出力できない．そのため，管理者はこれらの情報をHIDSの検知結果から人手で特定する必要があるため，調査に時間を要してしまう．

本研究ではHIDSの入力であるシステムコールやSQLクエリ発行に対して，それらを発生させたHTTPリクエストを処理したスレッドのIDと高精度な処理開始および終了時刻に基づいてHTTPリクエストを関連付ける手法を提案する．関連付けにより，HIDSが異常と検知したシステムコールやSQLクエリ発行がどのHTTPリクエストによって発生したかを特定できるようになる．そのため，管理者はHIDSの結果から被害の原因となったWebアプリケーションに関する情報は把握することができる．

本研究の貢献は以下のとおりである．

- HIDSの入力であるシステムコールやSQLクエリ発行に対してHTTPリクエスト情報を関連付けることで，HIDSが検知した際に標的となったWebアプリケーションや攻撃の送信元を特定する手法を実現した．
- 様々なWebアプリケーションに対して関連付け精度および処理速度の評価を行い，誤った関連付けがないこと，およびWebアプリケーションのパフォーマンス低下を平均5%程度に抑えられることを示した．

2. 既存のイベント関連付け手法とその課題

システムコールやSQLクエリ発行の情報にHTTPリクエストを関連付けることは異なる種類のイベントを関連付けることと考えることができる．異なる種類のイベントを関連付ける手法にはイベントの発生時刻の近さやイベントを発生させたプロセスのID（PID）あるいはスレッドのID（TID）などの識別子の関係を利用して関連付ける手法が提案されてきた．Skopikらの手法 [8] は一定の時間間隔をあらかじめ定義し，その間隔内に起きたイベントを関連付けるものである．しかし，適切な時間間隔を決めることは困難であり，定義した時間間隔が長すぎると関係のないイベントを関連付けてしまったり，時間間隔が短すぎると関係するイベントであるが関連付けられない事象が発生してしまうため正確性が不十分である．イベントにPID，TIDや親プロセスID（PPID）など明示的に関連付けることができる情報が含まれている場合，BackTracker [9] が利用できる．この手法ではあらかじめ指定した期間に発生するイベントに対してPID/TIDなどが一致するか否かで関連付けを行うため，時間間隔のみによる関連付け手法より正確である．たとえば，HTTPリクエストがつねに異なるプロセス/スレッドで処理される場合，HTTPリクエストと関連付けるべきイベントのPID/TIDは一意に定まる

[*1] https://www.akamai.com/us/en/resources/our-thinking/state-of-the-internet-report/

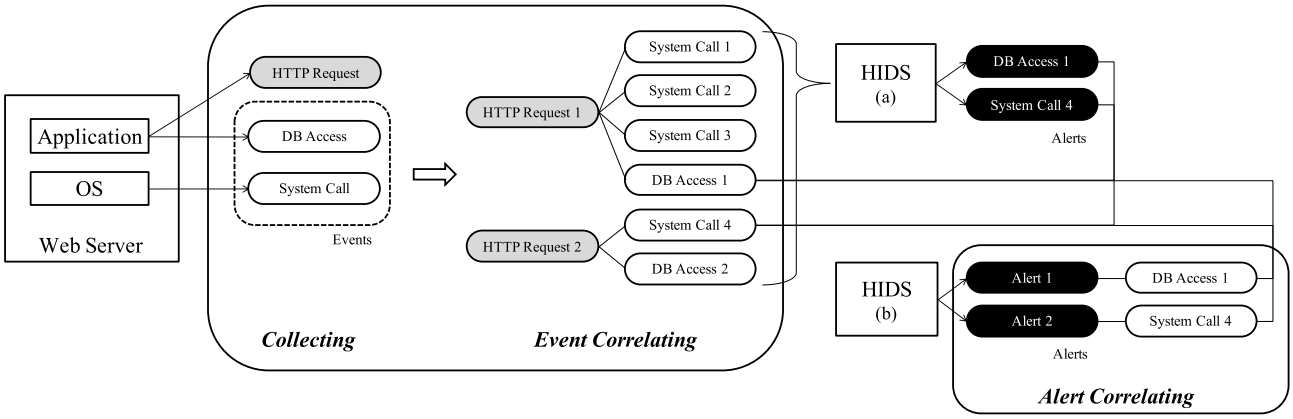

図 2 提案手法の概要
Fig. 2 Overview of our approach.

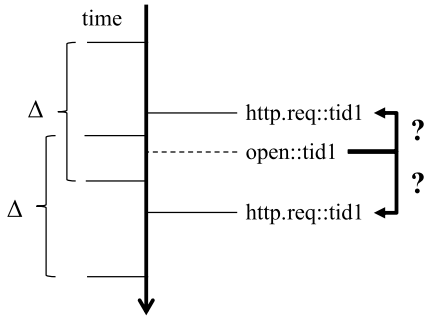

図 1 PID/TID に基づく関連付けの課題
Fig. 1 Problem of correlation method based on PID/TID.

ため BackTracker による関連付けは正確である．しかし，サーバプロセスのような同じプロセス/スレッドで異なる HTTP リクエストを処理する場合，Skopik らの手法と同様な課題が発生する．**図 1** にこの問題の様子を示す．2 つの HTTP リクエストがスレッド 1 によって処理され，その間にスレッド 1 からファイルアクセスが発生した．Skopik らの手法と同様，BackTracker は PID/TID という関連付けする際の制約があるものの，関連付け対象となるイベントから時間間隔 Δ に含まれるイベントを関連付ける．そのため，図 1 のような場合このファイルアクセスはどちらも HTTP リクエストの発生時刻から Δ 以内に含まれるため，どちらか断定することができないという課題が存在する．

本論文ではこれらの関連付けの正確性に関する課題を解決する関連付け手法を提案する．

3. 提案手法

図 2 に提案手法の概要を示す．提案手法は HTTP リクエストおよびイベントの収集（Collecting），HTTP リクエストとイベントの関連付け（Event Correlating）およびイベントとアラートの関連付け（Alert Correlating）の 3 つのフェーズに大別される．提案手法では 2 つの HIDS の形態を想定している．図 2 の HIDS (a) のように HIDS が提案手法で収集したイベントを入力として，アラートを出力する際にそのイベントを出力できる形態および図 2 の HIDS (b) のように HIDS が提案手法で収集したイベントを入力とせず，出力するアラートとイベントを関連付ける必要がある形態である．

Collecting フェーズでは OS のカーネルやプロセスからシステムコールや SQL クエリ発行に関する関数呼び出しの情報を収集する．本研究ではこれらの情報をイベントと呼び，システムコールや SQL クエリ発行の単位をイベントの単位として扱う．Event Correlating フェーズでは収集した複数のイベントがどの HTTP リクエストに関連するものかを複数の条件に基づき関連付けを行う．関連付けにより，ある HTTP リクエストがサーバ内のどのようなイベントを引き起こしたかが判別される．その結果，HIDS に提案手法で収集したイベントを入力とすることで，HIDS が異常なイベントを検知した際に，そのイベントがどの HTTP リクエストに関連付いているものかを特定することが可能となる．Alert Correlating フェーズではアラートとイベントの関連付けを行う．この処理は HIDS が図 2 の HIDS (b) の形態の場合のみに実施する．

図 3 に提案手法が利用者に対して提示する情報のイメージを示す．提案手法の出力は HIDS のアラート，関連付いた HTTP リクエスト，および関連付いたイベントの 3 つの情報である．4 から 8 行目は HIDS アラートに関する情報であり，3 つのイベントが HIDS によって異常と検知されたことを示す．10 から 12 行目は HIDS アラートが示すシステムコールを引き起こした HTTP リクエストの内容を示している．14 から 20 行目は上記 HTTP リクエストが引き起こしたすべてのイベントを示している．これらの情報が関連付けされていることにより，利用者は HIDS アラートの分析の際に被害発生の原因となる HTTP リクエストを自ら関連付ける必要がなくなり，分析時間が短縮できると考える．

```
1  ================================================
2  * Report of proposed system #1
3  ================================================
4  * HIDS alert:
5    * anomaly events:
6      - execve, sed -ie s/deny/allow/g .htaccess
7      - open,   ./sed3wZccC, O_RDWR|O_CREAT
8      - rename, ./sed3wZccC, .htaccess
9  ------------------------------------------------
10 * correlated http request:
11   - POST /user/register?
12     element_parents=account/mail/%23value
13 ------------------------------------------------
14 * correlated host events:
15   - syscall, open, .htaccess, O_RDONLY:O_CLOEXEC
16   - syscall, connect, /var/lib/mysql/mysql.sock
17   - db,      SELECT cid, data, ... FROM ...
18   ...
19   - syscall, open, .htaccess, O_RDONLY
20   - syscall, open, ./sed3wZccC, O_RDWR,O_CREAT
21   - syscall, rename, ./sed3wZccC, .htaccess
22 ================================================
```

図 3 利用者に提示する情報の例
Fig. 3 Example report sent to user.

3.1 Event Correlating フェーズ

既存の関連付け手法 [8], [9] の問題は一定の時間間隔を定めて関連付けるため，関連付けが不正確になることにあった．そのため，本手法では既存手法が利用する PID/TID などの識別子による区別に加え，あらかじめ定義された一定の時間間隔ではなく，HTTP リクエストの処理をふまえて動的に決定される時間制約を設けることでイベントがどの HTTP リクエストによって発生したかを区別する．以下に関連付けを行う条件を示す．

条件 1 イベントを発したスレッドが HTTP リクエストを処理したスレッドと同一，あるいはイベントを発したプロセスが HTTP リクエストを処理したプロセスを根とするプロセスツリーに含まれる場合

条件 2 イベントの発生時刻が HTTP リクエストの処理スレッドによる処理開始時刻と処理終了時刻の間にある場合

条件 1 および**条件 2** 両方が満たされる場合に関連付けを行う．

条件 1 は既存手法 [9] と同様に PID や TID の一致に基づいた関連付けの条件である．Web サーバは一般に複数のプロセスを起動する，リクエストの処理を行う動作モデルには，各プロセス内に 1 つのスレッドを用意し待機する Prefork モード，各プロセス内に複数のスレッドを用意し待機する Worker モードの 2 つが存在する．いずれの動作モードであっても，1 つのスレッドは 1 つの HTTP リクエストの処理しか行わない．近年，これらの動作モデルよりリソース活用が効率的な Event モードという動作モデルも利用されてきているが，待機プロセス/スレッドが同時に 2 つ以上のリクエストを処理することはないという点は

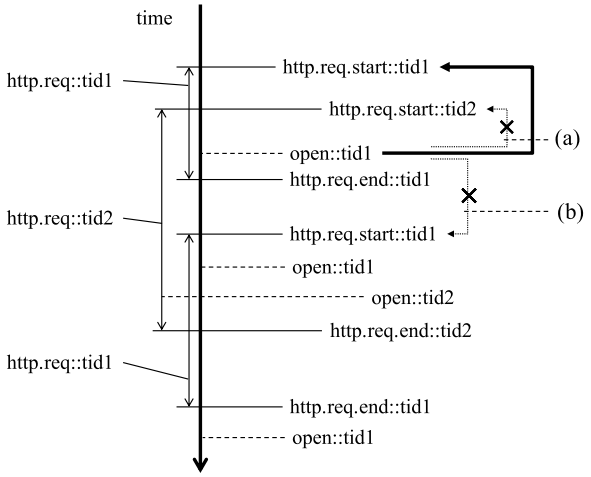

図 4 TID の関係性に基づく関連付け
Fig. 4 Event correlation based on equal TID relationship.

Prefork/Worker モードと共通している．本手法では上記のような 1 つのプロセスまたはスレッドは HTTP リクエストの処理を終えるまで他の HTTP リクエストの処理を行わない Web サーバを対象とする．

Web サーバは HTTP リクエストを受け取ったら，待機しているスレッドのどれかに HTTP リクエストを割当て処理を行う [10]．このとき，待機スレッドは HTTP リクエストを同時に 2 つ以上処理を行うことはないため，TID の一致という条件からイベントが HTTP リクエストに関連するか否かを見分けることができる．システムコールが OS コマンドの実行など，子プロセスを生成する場合，実行される OS コマンドは HTTP リクエストを処理しているスレッドとは異なるスレッドでの処理になる．そのため，プロセスの親子関係を加味した，プロセスツリーに基づく関連付けを行う．具体的には `fork` あるいは `clone` システムコールの戻り値が実行された OS コマンドを処理するプロセスの PID となるため，子プロセスが生成された際は，子プロセスから発生したイベントを親プロセスで処理している HTTP リクエストに関するイベントとして関連付ける．

条件 2 は提案手法独自の関連付けの条件である．スレッドによる HTTP リクエストの処理開始および処理終了時の高精度な時刻を取得し，あるイベントがどの HTTP リクエストの処理中に発生したかを判定する．これにより，同一の TID となるスレッドで処理される異なる HTTP リクエストに起因して発生したイベントを区別する．この条件は 1 つのスレッドが同時に 2 つ以上の HTTP リクエストを処理することがないという Web サーバの特性 [10] に基づいている．高精度な時刻情報をどのように取得するかは 3.3 節にて詳説する．以下に 2 つの関連付けの例を示す．

図 4 に 3 つの HTTP リクエストとそれに関連付けられたイベントの例を示す．3 つの HTTP リクエストはそれぞれスレッド 1，スレッド 2，スレッド 1 によって処理されている．この例ではイベントと HTTP リク

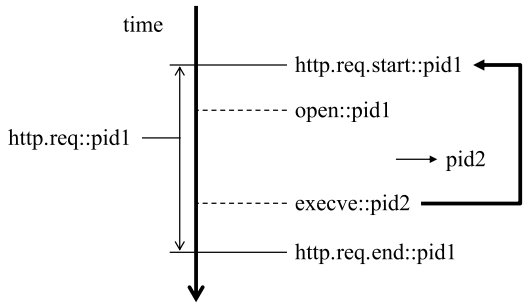

図 5 プロセスツリーに基づく関連付け
Fig. 5 Event correlation based on process tree relationship.

エストの TID が一致したことにより関連付けられた様子を示している．図 4 ではイベントの種類（event type）とそのイベントの TID/PID/PPID を表すのに，`event_type::tid1` や `event_type::pid1` のように表現する．`http.req.start::tid1` と `http.req.end::tid1` はスレッド 1 で処理された HTTP リクエストの処理開始と処理終了を示すイベントである．`open::tid1` はスレッド 1 から発生したファイル操作イベントである．このファイル操作イベントはスレッド 1 より発生していること（条件 1）および HTTP リクエストの処理開始時刻と処理終了時刻の間にあること（条件 2）から，`open::tid1` はスレッド 1 で処理された HTTP リクエストに関連するものであって，スレッド 2 で処理された HTTP リクエストに関連するものではないことが図 4 (a) より分かる．加えて，`open::tid1` が最初のスレッド 1 で処理された HTTP リクエストに関連付くものであって，2 番目のスレッド 1 で処理された HTTP リクエストに関連するものではないことが図 4 (b) より判断できる．

2 つ目の例はプロセスツリーの関係性に基づく関連付けの例である．**図 5** にその概要を示す．OS コマンド実行の場合，コマンドは HTTP リクエストを処理するプロセスから生成された子プロセスとして実行される．これは TID の一致に基づく追跡ができないため，PID および PPID の関係性を利用し，プロセスツリーに基づいて関連付けを行う．

図 5 にプロセスツリーに基づいた関連付けの様子を示す．プロセス 1 が HTTP リクエストを処理する際にコマンド実行が発生し，`fork` システムコールの戻り値より，プロセス 2 が生成されていることが分かる．`execve::pid2` および `open::pid2` はプロセス 2 により発生したイベントである．`fork` システムコールよりプロセス 2 はプロセス 1 から発生していることが分かるため，これら 2 つのイベントはプロセス 1 で処理した HTTP リクエストに関連するものであると判断する．

3.2 Alert Correlating フェーズ

従来提案されてきた手法ではどれも異常なシステムコールを検知する手法 [11], [12], [13] や異常な SQL クエリ発行を検知する手法 [14], [15] であるため，アラートとして異常となったシステムコールの情報や SQL クエリの情報を出力することが可能であると考える．そのため，これらの手法がアラートを出力した際はそのアラートを引き起こしたシステムコールの時刻や発行した際の PID/TID，SQL クエリの内容をもとに提案手法で収集したシステムコールや SQL クエリと一意に関連付けが可能と考える．つまり図 2 の HIDS (a) あるいは (b) いずれかの形態に該当し，HIDS のアラートからそれを引き起こした HTTP リクエストまで追跡可能と考える．

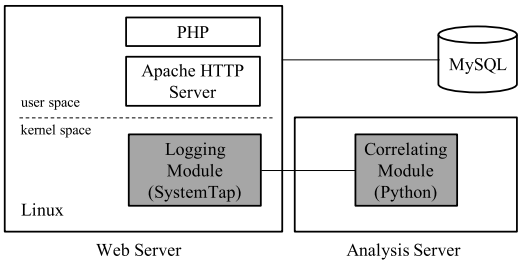

図 6 プロトタイプシステムの概要
Fig. 6 Overview of our prototype system.

3.3 実装

提案手法を検証するため，プロトタイプ実装を行った．**図 6** にプロトタイプシステムの全体像を示す．図中の灰色部分が提案手法に関わる部分である．3 章に示した Collecting フェーズの処理はロギングモジュール（Logging Module）に実装され，3.1 節および 3.2 節に示した，Event Correlating フェーズおよび Alert Correlating フェーズの処理は Correlating Module に実装されている．

プロトタイプでは PHP で作成された Web アプリケーションが Apache HTTP Server 上で動作し，DB に MySQL を用いている状況を想定した．Web サーバの動作モデルは Prefork/Worker，どちらも対応できるよう実装を行った．本実装では Apache HTTP Server と PHP を対象としているが，これは提案手法がこの実装に制限されているというわけではない．Web サーバが Nginx や lighthttpd などに変わったとしても同じ要領でイベントを取得可能である．実際，Nginx や lighthttpd でも同様にイベントを取得できることを確認した．

イベントは監視対象のサーバで動作するロギングモジュールによって収集され，収集したイベントは関連付けを行う別サーバに送られる．提案手法では SystemTap [*2] を利用してロギングモジュールを作成した．このモジュールによってシステムコールや SQL クエリ発行および HTTP リクエストの収集を行う．SystemTap は，カーネルモジュールを作成しシステムコールやユーザ空間の関数をランタイムフックするためのフレームワークである [16], [17]．ラン

[*2] https://sourceware.org/systemtap/

タイムフックとはプログラム実行時に目的とする関数などが実行された際に追加の処理を行わせる仕組みである．SystemTap はユーザが作成したスクリプトを入力として，それをカーネルモジュールとしてコンパイルし，動作中のカーネルに挿入する．SystemTap はパフォーマンス測定に利用された実績 [18] があり，サーバに与える影響はきわめて小さいと考えたため本実装に採用した．以下，提案する関連付け手法で必要になる HTTP リクエストの処理開始と処理終了の時刻，システムコール，SQL クエリ発行の取得方法について詳説する．

HTTP リクエストの処理開始と処理終了の時刻． Apache HTTP Server の場合，HTTP リクエストは関数 `ap_process_request` によって処理されているため，この関数を呼び出す前後でログを出力するようランタイムフックを行った．HTTP リクエストの処理開始時刻と処理終了時刻は SystemTap が提供する現在時刻をマイクロ秒の精度で取得する関数 `gettimeofday_us` を使用することで実現した．HTTP メソッド，リクエスト URL，バージョンやユーザエージェントなどの HTTP リクエストに関する情報は関数 `ap_process_request` の第 1 引数である構造体 `request_rec` から取得した．

システムコール． システムコールの情報は SystemTap の機能を活用することで取得した．SystemTap ではシステムコールの種類だけでなく，各システムコールの引数および戻り値まで取得可能である．

SQL クエリ． DB を PHP 言語などのミドルウェアから利用する場合，一般にはその DB が用意するラッパーを経由する．たとえば，MySQL の場合クライアントライブラリである `libmysqlclient.so` かネイティブドライバーである `mysqlnd.so` を介して SQL クエリ発行が行われる．そのため，本実装ではラッパーに実装されている SQL クエリ発行を担う関数である `mysql_query`，`mysql_send_query` および `php_mysqlnd_conn_data_query_pub` を監視することで，PHP アプリケーションから DB にアクセスする際に使用される SQL クエリを取得した．

4. 評価

提案手法を関連付け精度，処理速度，使用リソース量の観点から評価を行った．

4.1 関連付け精度

目的． 提案手法はイベントの関連付けするための手法であり，正しく関連付けを行えているかが実用性に大きく影響する．そのため，提案手法および既存手法のイベント関連付け精度の評価を行った．

準備． **図 7** に示す関連付け精度評価用 Web アプリケーションを作成した．URL パラメタに識別子（uid）となる値を設定して HTTP リクエストを行うと，識別子がシステムコールや DB アクセスのイベントに現れる仕様である．評価の際に利用した DB には追加で定義したスキーマはなく，デフォルト状態のものを利用した．4 行目の `fopen` 関数によってファイルパス `/tmp/$uid` にアクセスを行うが，評価時はこのファイルパス対応するファイルを用意しなかった．ファイルアクセスは失敗となるが，ファイルアクセスに相当するシステムコールは発行されるため評価に影響しないと考えたからである．

```
1  <?php
2  $uid = $_GET['uid'];
3  // file access
4  $fp = fopen("/tmp/{$uid}", 'r');
5  fclose($fp);
6  // network access
7  $socket = socket_create(AF_INET, SOCK_STREAM,
8                          SOL_TCP);
9  $result = socket_connect($socket, '127.0.0.1',
10                          $uid);
11 socket_close($socket);
12 // command
13 system("cat /tmp/{$uid}");
14 // db access
15 $mysqli = new mysqli('127.0.0.1',
16                       'root', 'root',
17                       'information_schema');
18 $sql = "SELECT {uid} FROM information_schema";
19 $mysqli->query($sql)->close();
20 ?>
```

図 7 イベント関連付け精度評価用 Web アプリケーション
Fig. 7 Web application for evaluating event correlation accuracy.

各イベントに現れる識別子が関連付けられた HTTP リクエストに現れる識別子と一致しており，かつ正しい識別子であるイベントのみが関連付けられている場合，正しい関連付けと判断した．つまり，関連付けるべき識別子のイベントが関連付いていない，あるいは異なる識別子のイベントが HTTP リクエスト関連付けられている場合誤った関連付けと判断した．

評価用 Web アプリケーションに対して同時接続数 1，5，10，50，100 および 500 として 10,000 回の HTTP リクエストを送信し，HTTP リクエストとイベントの取得を行った．HTTP リクエストの送信には Apache JMeter [*3] を利用した．Web サーバには Prefork モードおよび Worker モードの Apache HTTP Server を利用し，サーバパラメタはデフォルト値[*4] を利用した．サーバが Prefork モードの場合，同時接続数が 50 以上およびモードが Worker サーバの場合，同時接続数が 500 以上の場合において，ロギングモジュールの性能限界により取得したイベントの欠損が発生していたため，これらの場合の測定値は除外した．

[*3] http://jmeter.apache.org/
[*4] デフォルト値は次のとおり．MinSpareServers: 5，MaxSpareServers: 10, MaxClients: 150, MinSpareThreads: 25, MaxSpareThreads: 75，ThreadsPerChild: 25

表 1 HTTP リクエストとホストイベントの関連付け精度
Table 1 Event correlation accuracy for each concurrency.

サーバモード	同時接続数	最大 RPS	平均 RPS	関連付け精度		
				BackTracker		提案手法
Prefork	1	113	66	83.56%	($\Delta = 0.10$)	100%
	5	490	480	88.53%	($\Delta = 0.01$)	100%
	10	521	517	17.65%	($\Delta = 0.01$)	100%
Worker	1	81	48	21.01%	($\Delta = 0.10$)	100%
	5	333	329	12.24%	($\Delta = 0.01$)	100%
	10	346	340	0.23%	($\Delta = 0.01$)	100%
	50	316	312	0.1%	($\Delta = 0.01$)	100%
	100	309	298	0.01%	($\Delta = 0.01$)	100%

比較として BackTracker [9] を利用した際の精度評価も行った．BackTracker の実装は論文に基づいて著者らが行った．BackTracker では時間間隔 Δ を指定する必要があるため，今回の評価では Δ を 1s，0.1s，0.01s および 0.001s とした際に最も関連付け精度が高い結果を BackTracker の関連付け精度とした．

結果．サーバモードおよび同時接続数を変化させた場合のイベント関連付け精度を**表 1** に示す．BackTracker は同時接続数が大きくなるほど誤った関連付けを起こしやすくなる傾向にあるが，提案手法は関連付けを誤ることが発生しなかった．今回の実験で秒間最大 521 リクエストまで正しく関連付けが行われることを確認できた．

考察．著者ら管理する個人のブログサイトおよび企業の Web サイトではそれぞれ，最大 RPS（Request Per Second）が 74 および 362 程度であることから，個人のブログサイトや企業の Web サイト程度のアクセスがある環境であれば十分実用可能な性能であると考える．実 Web サイトに対して適用可能かは厳密には最大同時接続数をもとに評価する必要があるが，Web サーバのアクセスログから同時接続数を把握することは困難であるため，本論文では最大 RPS をもとに適用可否の評価を行った．数千数万のアクセスが同時発生する大規模環境については後述するログ量の問題やロギングモジュールの性能面の問題があるため，提案手法およびそのプロトタイプは企業や個人などの一般的な Web サーバへの適用を想定している．

4.2 処理性能

目的．提案手法では Web サーバ内でログを取得するため，Web サーバのパフォーマンスに影響を与えることが想定できる．パフォーマンス低下が実用的である範囲内であるかを評価するため，**表 2** に示す著名な Web アプリケーションが動作する Web サーバに対して本手法を適用した際のスループット低下率，最大メモリ使用量，最大 CPU 使用率，平均ディスク使用量を調べた．

Web サーバの処理性能に影響を与えるのはイベントロギングを行うカーネルモジュールであり，関連付けを行う部分は Web サーバの処理性能に対して影響を与えないため，この部分の性能測定は実施しなかった．性能測定に使用したマシンのスペックは Intel Xeon 2.40-GHz CPU，8 GB RAM，128 GB HDD，CentOS 7.1 であった．

表 2 性能測定に用いた Web アプリケーションの一覧
Table 2 List of web applications used for throughput evaluation.

名称	バージョン	利用用途
Joomla	3.4.4	コンテンツ管理
Drupal	7.3.1	コンテンツ管理
MediaWiki	1.26.2	コンテンツ管理
WordPress	4.4.1	コンテンツ管理

4.2.1 HTTP リクエストのスループット

目的．HTTP リクエストのスループット低下は Web ページの表示速度の低下を意味し，ユーザビリティを損なうことになってしまうため，モジュールやシステムの導入においては HTTP リクエストのスループット低下を抑えることが必要である．そのため，提案手法のロギングモジュールを導入した際の HTTP リクエストのスループット低下を評価した．

準備．Web アプリケーションのパフォーマンス測定ツールとして有名な Apache JMeter を利用して測定を行った．スループット値は 1,000 リクエストを同時接続数 10 の状態で送信し計測することを 1 セットとし，5 セット測定した際の平均で求めた．Web サーバには Prefork モードの Apache HTTP Server を利用し，サーバパラメタはデフォルト値を利用した．T_d と T_e をそれぞれイベントロギングを無効にした場合と有効にした場合のスループットとし，スループット低下率は $\frac{T_d - T_e}{T_d}$ で求めた．

結果．表 3 に測定結果を示す．どのアプリケーションも平均スループット低下率が 5%以下であった．Joomla の平均スループット低下率が最も大きい結果となった．

考察．イベント収集を無効にした際の Joomla の RPS が 13.56 であるため，4.62%のスループットの低下はリクエストあたり約 73 ミリ秒の遅延を意味する．Joomla では各機能のアクセスに対して約 5 回のリクエストを行うことか

表 3 スループットの低下率
Table 3 Throughput overhead.

名称	機能	RPS	平均	低下率	平均	UP 割合/リクエスト	平均	ログ量/リクエスト	平均
Joomla	view top	13.01	13.56	4.79%	4.62%	16.86%	22.21%	35.50 KB	51.126 KB
	login	19.24		3.84%		16.18%		30.83 KB	
	create post	9.09		5.47%		39.09%		60.63 KB	
	view post	12.69		4.39%		16.70%		77.53 KB	
Drupal	view top	18.47	15.84	2.69%	2.67%	43.83%	49.13%	36.79 KB	45.06 KB
	login	16.80		4.44%		53.69%		61.02 KB	
	create post	14.23		0.17%		46.67%		42.50 KB	
	view post	18.43		2.78%		45.03%		31.59 KB	
	comment	11.27		3.28%		50.84%		53.39 KB	
MediaWiki	view top	8.95	6.89	2.97%	2.49%	32.13%	37.02%	45.32 KB	62.67 KB
	login	6.03		1.98%		29.00%		60.77 KB	
	create post	5.68		2.77%		41.92%		81.91 KB	
WordPress	view top	10.02	9.50	1.81%	1.54%	5.72%	4.89%	45.46 KB	40.91 KB
	login	14.73		0.62%		3.29%		34.67 KB	
	create post	3.55		1.52%		4.97%		47.17 KB	
	view post	12.55		2.08%		3.86%		42.32 KB	
	comment	5.74		1.70%		6.61%		34.96 KB	

ら，ユーザが体感する遅延は 0.37 秒程度である．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との調査[*5]があるため，スループット面では実用的であると考える．

スループット低下には 2 つの要因が存在する．1 つ目の要因はユーザ空間のプログラムに対するランタイムフックがある．提案システムではカーネルモジュールとして動作するため，システムコールはカーネル空間から取得可能であるが，HTTP リクエストや SQL クエリ発行に関する情報はユーザ空間のプログラムにフックを行うことが必要である．カーネル空間からユーザ空間に対してランタイムフックを行う際は uprobe [19] という仕組みが利用されるが，uprobe を介することでスループット低下率はさらに増加する．そのため，uprobe を介して取得する必要があるイベントの割合がスループット低下の要因につながっていると考える．表 3 中の UP 割合がすべてのイベントのうち，uprobe を介するイベントの割合である．UP 割合では Joomla の場合およびログ量では MediaWiki の場合を除いて，おおむねスループット低下率とは比例した関係であると考える．2 つ目の要因は出力するログ量である．ロギングモジュールが出力する情報が多いほど出力に時間を要するためスループット低下も大きいと考える．そのため，この 2 つの要因を削減することで今後よりスループット低下を抑えることが可能性が高いと考える．

4.2.2 リソース利用量

目的． Web サーバはそのアクセス数に応じて利用できるリソースが限られている場合もある．そのため，提案手法のロギングモジュールを利用した際の最大メモリ使用量，最大 CPU 使用率，平均ディスク使用量を調べ，過剰なリソース利用がないかを評価した．

表 4 リソース使用量
Table 4 Resource usage of proposed method.

名称	メモリ	CPU	ディスク/リクエスト
Joomla	53.4 MB	7.80%	12.0 KB
Drupal	53.4 MB	9.40%	11.3 KB
MediaWiki	53.3 MB	4.60%	10.8 KB
WordPress	53.3 MB	4.00%	7.00 KB

準備． ディスク使用量は出力されたログファイルを `gzip` で圧縮した際の値を計測した．圧縮した値を用いて比較する理由は，プロトタイプ実装のログフォーマットに冗長な部分があり，ログフォーマットの効率化によって改良可能なため，この部分による影響を除外するためである．

結果． リソース利用量の計測結果を**表 4** に示す．

考察． 50 MB 程度のメモリ使用量および 4%程度の CPU 使用率は近年の Web サーバに問題ない使用量だと考える．しかし，取得するログによるディスク使用量は一般的な場合に比べると非常に多い．法令やガイドラインでは，様々なセキュリティインシデントへの対処のためにログの保存期間として 3 カ月間や 1 年間，あるいはさらに長期間を定めている [20], [21]．一般的な Web サーバの圧縮時のアクセスログのディスク使用量はリクエストあたり約 20 B に対し，本手法で取得するログのディスク使用量はリクエストあたり平均約 8.75 KB であるため，通常のアクセスログに比べ，400 倍を超える容量となるが，提案手法で取得するログは，通常のアクセスログの補完として，攻撃発生時

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

表 5 Drupal に対する OS コマンドインジェクションに対する関連付け結果
Table 5 Xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx.

No.	HIDS 異常判定	種類	内容
1		http	POST /user/register?element_parents=account/mail/%23value
2		syscall	open, .htaccess, O_RDONLY:O_CLOEXEC
3		syscall	net, connect, /var/lib/mysql/mysql.sock
4		db	SET NAMES utf8mb4
5		db	SELECT cid, data, ... FROM cache_container WHERE cid IN (*) ORDER BY cid
6	✓	syscall	execve, sed -ie s/allow,deny/deny,allow/g .htaccess
7		syscall	open, /lib64/libc.so, O_RDONLY:O_CLOEXEC
8		syscall	open, /proc/meminfo, O_RDONLY:O_CLOEXEC
9		syscall	connect, /var/run/nscd/socket
10		syscall	open, /etc/nsswitch.conf, O_RDONLY:O_CLOEXEC
11		syscall	open, /etc/ld.so.cache, O_RDONLY:O_CLOEXEC
12		syscall	open, /etc/passwd, O_RDONLY:O_CLOEXEC
13		syscall	open, /lib64/libpcre.so, O_RDONLY:O_CLOEXEC
14		syscall	open, /proc/filesystems, O_RDONLY
15		syscall	open, /usr/lib64/charset.alias, O_RDONLY,O_NOFOLLOW
16		syscall	open, .htaccess, O_RDONLY
17	✓	syscall	open, ./sed3wZccC, O_RDWR,O_CREAT,O_EXCL
18	✓	syscall	rename, ./sed3wZccC, .htaccess

に即時確認用として短期保存を想定することで運用可能と考えている．

4.3 事例分析

提案手法の効果は HIDS アラートが発生した際にどの HTTP リクエストがそのアラートを引き起こしているかを特定できる所にあり，人手で特定する必要がないため，アラートに対する調査の時間短縮に貢献できる所にある．提案するイベント関連付け手法を活用した際の効果について 2 つの事例をもとにどのように HIDS アラート調査の時間短縮に貢献したかを評価した．評価に際しては Mutz らの手法 [13] を参考に著者らが実装した HIDS を活用した．

4.3.1 事例 1

実験環境にて脆弱性が存在する Web アプリケーション Drupal 8.5.0 を用意し，OS コマンドインジェクションの脆弱性（CVE-2018-7600）に対して攻撃を行った．HIDS はコマンドの頻度から異常なコマンド実行を検知することが可能である．**表 5** に提案手法で出力された単一のリクエストの処理の開始から終了までのイベントの一部を抽出した結果を示す．HIDS では No.6，17 および 18 のイベントを異常として検知していた．提案手法にて関連付けた No.1 の HTTP リクエストの情報によりこの攻撃は URL パス `/user/register` に対するものであることが分かる．この情報により，管理者は脆弱性に対するパッチが提供される前に，この URL に対するアクセス制限を設けるなどの暫定対処を行うことが可能となる．

人手でこれらのイベントを関連付ける場合，まず，No.6，17 および 18 の HIDS が異常判定したイベントから No.1 に示す HTTP リクエストを探す必要がある．イベントの発生時刻が近いことに基づき関連付ける必要がある．イベントの時刻精度が十分でない場合，あるいは短時間に複数リクエストが発生している場合，正しく関連付けるには Web アプリケーションの仕様や Web サーバの状況に鑑みて行う必要があり，時間を要すると考える．また，攻撃となった HTTP リクエストによって何がなされたのかの全体像を把握するためには，SQL クエリの発行ログやシステムコールログなどから No.2 から No.5 および No.7 から No.16 のイベントを関連付ける必要がある．通常，SQL クエリ発行ログにはどのプロセス/スレッドで発生したものなのかを示す識別子は存在しないため，Web アプリケーションの仕様に基づいて関連付ける必要があり，時間を要すると考える．提案手法により上記の関連付け作業が必要なくなるため，分析による時間を短縮できると考える．

4.3.2 事例 2

我々は HIDS およびブログサイトを日々運用しており，2019 年 4 月 1 日から 2019 年 7 月 22 日の間で 282,585 件の HTTP リクエストを観測し，そのうち 49 件の HTTP リクエストによるイベントが異常として検知されていた．検知したイベントすべてを確認し，侵害を示すものではなかったことから 49 件の検知はすべて誤検知であった．

表 6 に誤検知した際に提案手法が関連付けたイベントの一部を示す．HIDS は No.2 から No.11 まですべてのイベントを異常として検知していた．しかし，関連付いた HTTP リクエストの内容を確認しても攻撃コードが含まれていなかった．今回誤検知した動作は我々が運用している Web アプリケーション WordPress の更新確認にともなう

表 6 誤検知に対する関連付け結果
Table 6 Xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx.

No.	HIDS 異常判定	種類	内容
1		http	GET /category/events
2	✓	syscall	open, /lib64/libsoftokn3.so, O_RDONLY:O_CLOEXEC
3	✓	syscall	open, /lib64/libfreeblpriv3.so, O_RDONLY:O_CLOEXEC
4	✓	syscall	open, /etc/passwd, O_RDONLY
5	✓	syscall	open, /etc/pki/nssdb/pkcs11.txt, O_RDONLY
6	✓	syscall	open, /proc/sys/crypto/fips_enabled, O_RDONLY
7	✓	syscall	open, /lib64/libnsssysinit.so, O_RDONLY:O_CLOEXEC
8	✓	syscall	open, /etc/pki/nssdb/cert9.db, O_RDWR:O_CREAT:O_CLOEXEC
9	✓	syscall	open, /etc/pki/nssdb/key4.db, O_RDWR:O_CREAT:O_CLOEXEC
10	✓	syscall	open, /etc/pki/nss-legacy/nss-rhel7.config, O_RDONLY
11	✓	syscall	open, /lib64/libnsspem.so, O_RDONLY:O_CLOEXEC

ものであるため，HIDS の誤検知であると確認できた．このように HTTP リクエスト情報が関連付いていることにより，HTTP リクエストに攻撃コードが含まれているか否かの情報を加味して攻撃に対応できるようになるため，管理者にとってより容易に攻撃に対する対応が可能となる．No.4 のイベントは攻撃によってもよく引き起こされるため，HIDS の結果だけから判断すると侵害されていると判断していまう可能性も高いが，HTTP リクエストの情報とともに判断することによってより早期に誤検知と判断できる．

5. 提案手法の実用時における考慮事項

5.1 HIDS アラートの情報量による制限

3.2 節で述べたように提案手法では HIDS アラートがイベントと関連付けられることが前提条件となる．たとえば，HIDS アラートに異常となったイベントの時刻，イベントを発生させたプロセスの ID，イベントを発生させたスレッドの ID などの情報が存在すれば，これらの情報をもとに HIDS アラートとイベントを一意に関連付けることが可能である．しかし，一般に利用されている HIDS の中にはアラートに出力できる情報が限られている場合がある．この場合，HIDS がアラートを出力しても，提案手法で取得したイベントと関連付けることができない．たとえば Tripwire Enterprise [*6] を始めとする HIDS 製品では，アラートとホストイベントを関連付けられる情報がアラート中に存在しないため，本手法は適用できない．

5.2 サーバ改ざんの可能性

攻撃者が Web サーバの制御を取得できてしまった場合に，提案手法を無効化できる恐れがある．提案システムではカーネルモジュールを利用してイベント取得を行っている．そのため，攻撃者が管理者権限を奪取でき，またこのカーネルモジュールを特定した場合は，カーネルモジュールを停止することで提案手法は利用できなくなってしまう．この制約に対しては完全性のチェックを実行時に行う仕組みを導入することで今後，解決可能であると考える．

[*6] https://www.tripwire.co.jp/products/enterprise/

6. 関連研究

6.1 NIDS

NIDS はパケットや通信データから特徴量を抽出し，攻撃検知を行う．WAF は NIDS の領域において，Web アプリケーションに特化したものである．研究は古くから行われている [4], [22], [23]．NIDS はシステム改変が少ないことから導入が容易であるが，失敗した攻撃もアラートとして通知してしまうため，管理者の調査すべきアラートが大量になってしまい，管理者の負担が大きくなりがちである．

6.2 EIDS

NIDS の課題に鑑み，著者らはネットワーク通信におけるリクエストおよびレスポンスを両方監視し，リクエストに含まれる攻撃コードをエミュレーションし得られる攻撃の痕跡がレスポンスに含まれる場合に攻撃の成否を判定する手法 [24] を提案した．これより，一部の攻撃については攻撃が成功したという結果をもとに優先的に調査できるようになった．この手法ではネットワーク通信に攻撃の痕跡が現れる攻撃のみ判定可能であり，サーバ内のファイルの内容を改変するあるいは DB の内容を消去するといったホスト内部に閉じた痕跡を残す攻撃に対しては成否を判定できないため，人手によるアラート調査を要する．

6.3 HIDS

HIDS はホストであるサーバや端末の OS から収集できる情報を利用して攻撃検知を行う．サーバ改変を要するため，導入時にアプリケーションの動作に悪影響がないか検証を再度行う必要があるため手間を要する．システムコールの順序やシステムコールの引数を特徴量として異常検知を行う手法 [11], [12], [13] や，Web アプリケーションに特

化し，DB アクセスの際に利用される SQL クエリを特徴量として異常検知を行う手法などが存在する [14], [15]．これらの手法では OS や DB など被害が発生する部分から特徴量を抽出するため，攻撃の成否を判定でき，攻撃成功時のみをとらえてアラートを通知することができると考えれる．NIDS を比較した場合，HIDS は失敗した攻撃に対しては検知せず，攻撃成功時のみを検知するため，NIDS より精度の高い通知が可能と考える．しかしながら，HIDS には HTTP リクエストの情報が関連付いていないため，被害が発生させた原因となった Web アプリケーションや攻撃元といった被害の再発防止のための暫定対処に寄与する情報は管理者がサーバのログから調査する必要がある．

7. おわりに

Web アプリケーションに対する攻撃数は増加の一途をたどっている．NIDS は導入しやすいが，失敗した攻撃に対してもアラート通知を行うため，管理者が対応すべきアラート通知が大量になってしまう．HIDS はホストで発生したイベントを利用するため，アラートが攻撃が成功した場合となる可能性が高く，より精度の高い通知が可能である．しかし，被害の原因調査に必要な攻撃対象の Web アプリケーションの URL や攻撃元など，攻撃となった HTTP リクエストに関する情報を出力できないため，人手の調査を要し時間がかかってしまう．本研究では，HIDS の入力であるシステムコールや SQL クエリ発行をそれらを発生させた HTTP リクエストを，処理したスレッドの ID と高精度な処理開始および終了時刻に基づいて HTTP リクエストと関連付けることで，HIDS で検知した際に管理者が攻撃対象の Web アプリケーションに関する情報を瞬時に特定できるようにした．評価では，提案手法が誤った関連付けをすることがなく，Web アプリケーションに与えるパフォーマンス低下を 5%以下に抑え，実用的であることを示した．今後の課題として，容量を抑えたログの記録方法の検討や，高いスループットが要求される環境においても提案手法を利用できるようパフォーマンス低下をさらに抑えたロギングモジュールの作成に取り組む．

参考文献

[1] Canali, D. and Balzarotti, D.: Behind the Scenes of Online Attacks: An Analysis of Exploitation Behaviors on the Web, *NDSS* (2013).

[2] John, J.P., Yu, F., Xie, Y., Abadi, M. and Krishnamurthy, A.: Searching the Searchers with Searchaudit, *USENIX Security* (2010).

[3] Kruegel, C., Vigna, G. and Robertson, W.: A multi-model approach to the detection of web-based attacks, *Computer Networks* (2005).

[4] Ingham, K.L., Somayaji, A., Burge, J. and Forrest, S.: Learning DFA representations of HTTP for protecting web applications, *Computer Networks* (2007).

[5] Robertson, W.K., Maggi, F., Kruegel, C. and Vigna, G.: Effective Anomaly Detection with Scarce Training Data, *NDSS* (2010).

[6] Maggi, F., Member, S., Matteucci, M. and Zanero, S.: Detecting Intrusions through System Call Sequence and Argument Analysis, *IEEE Trans. Dependable and Secure Computing* (2010).

[7] Bridges, R.A., Glass-Vanderlan, T.R., Iannacone, M.D. and Vincent, M.S.: A Survey of Intrusion Detection Systems Leveraging Host Data, *ACM Comput. Surv.* (2019).

[8] Skopik, F. and Fiedler, R.: Intrusion Detection in Distributed Systems using Fingerprinting and Massive Event Correlation, *GI-Jahrestagung* (2013).

[9] King, S.T. and Chen, P.M.: Backtracking Intrusions, *ACM SOSP* (2005).

[10] Menasce, D.A.: Web Server Software Architectures, *IEEE Internet Computing* (2003).

[11] Hofmeyr, S.A., Forrest, S. and Somayaji, A.: Intrusion Detection using Sequences of System Calls, *Computer Security* (1998).

[12] Wagner, D. and Soto, P.: Mimicry Attacks on Host-Based Intrusion Detection Systems, *CCS* (2002).

[13] Mutz, D., Valeur, F., Vigna, G. and Kruegel, C.: Anomalous System Call Detection, *ACM Transactions on Information and System Security* (*TISSEC*) (2006).

[14] Halfond, W.G.J. and Orso, A.: AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks, *Proc. 20th IEEE/ACM International Conference on Automated Software Engineering* (2005).

[15] Valeur, F., Mutz, D. and Vigna, G.: A learning-based approach to the detection of SQL attacks, *DIMVA* (2005).

[16] Prasad, V., Cohen, W., Eigler, F.C., Hunt, M., Keniston, J. and Chen, J.: Locating system problems using dynamic instrumentation, *Ottawa Linux Symposium* (2005).

[17] Eigler, F.C.: Problem solving with systemtap, *Ottawa Linux Symposium* (2006).

[18] Jacob, B., Larson, P., Leitao, B. and Silva, S.D.: SystemTap: Instrumenting the Linux Kernel for Analyzing Performance and Functional Problems, *IBM Redbook* (2008).

[19] Keniston, J., Mavinakayanahalli, A., Panchamukhi, P. and Prasad, V.: Ptrace, Utrace, Uprobes: Lightweight, Dynamic Tracing of User Apps, *Linux Symposium* (2007).

[20] 企業における情報システムのログ管理に関する 実態調査：独立行政法人情報処理推進機構（IPA）(2016).

[21] 高度サイバー攻撃への対処におけるログの活用と分析方法：一般社団法人 JPCERT コーディネーションセンター (2016).

[22] Kruegel, C. and Vigna, G.: Anomaly Detection of Web-based Attacks, *ACM CCS* (2003).

[23] Song, Y., Keromytis, A.D. and Stolfo, S.J.: Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic, *NDSS* (2009).

[24] 鐘本 楊，青木一史，三好 潤，嶋田 創，高倉弘喜：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，情報処理学会論文誌 (2019).

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．

小谷 大祐 （正会員）

2016 年京都大学大学院情報学研究科知能情報学専攻博士後期課程修了．同年より京都大学学術情報メディアセンター助教．京都大学博士（情報学）．インターネットアーキテクチャ，ネットワークセキュリティの研究に従事．電子情報通信学会，ACM，IEEE 各会員．

岡部 寿男 （正会員）

1988 年京都大学大学院工学研究科修士課程修了．同年京都大学工学部助手．同大型計算機センター助教授等を経て，2002 年同学術情報メディアセンター教授．博士（工学）．2005 年より国立情報学研究所客員教授．インターネットアーキテクチャ，ネットワーク・セキュリティ等に興味を持つ．電子情報通信学会フエロー．システム制御情報学会，日本ソフトウェア科学会，IEEE，ACM 各会員．