情報処理学会論文誌 Vol.62 No.12 1985–1995 (Dec. 2021)

SQLインジェクション攻撃対象の内部情報に依存しない攻撃フェーズ識別手法

黒木 琴海[1,†1,a)] 鐘本 楊[1] 青木 一史[1] 野口 靖浩[2] 西垣 正勝[2]

受付日 2021年3月8日, 採録日 2021年9月9日

概要：Structured Query Language（SQL）インジェクションはいまだに多発しており，個人情報の漏洩といった重大な被害を引き起こす攻撃である．Web Application Firewall（WAF）はSQLインジェクションにも対応しているが，多くの場合，検知の機能のみが用いられ，検知した攻撃の分析は人手で行われている．しかし，SQLインジェクションは大量に発生するため，WAFによって検知されたすべてのSQLインジェクションを人手で分析することは困難である．本稿では，攻撃対象のシステムの内部情報の利用や，分析用の詳細ログを収集するためのシステムの改変が困難な状況を想定して，HTTPリクエストに含まれる部分的なSQL文を分析対象に攻撃フェーズを識別する手法を提案する．提案手法では，WAFアラートに関連するHTTPリクエストに含まれる部分的なSQLクエリのみを解析対象として，初期状態のDBMSを用いて動的解析することで攻撃対象システムの内部情報や対象システムの改変を必要としない．評価より，提案手法は人工データセットに対して78.1%の精度で正しく識別できたこと確認した．また，提案手法により，実環境で約5割の被害を及ぼさないフェーズにある攻撃を識別し，これを分析から除外することで人手分析の稼働を減少できる効果がある可能性を示す．

キーワード：ネットワークセキュリティ，Webセキュリティ，SQLインジェクション，WAF，SOCアナリスト，攻撃フェーズ識別

SQL Injection Attack Phase Identification without Relying on Internal Information of Attack Targets

Kotomi Kuroki[1,†1,a)] Yo Kanemoto[1] Kazufumi Aoki[1] Yasuhiro Noguchi[2] Masakatsu Nishigaki[2]

Received: March 8, 2021, Accepted: September 9, 2021

Abstract: Structured Query Language (SQL) injections are still one of the major reasons that lead to serious damage such as leakage of personal information. Web Application Firewalls (WAFs) are used to detect SQL injections, but manual analysis of the detection result is also required because of false positives. However, since a large amount of SQL injection occurs, it is difficult to manually analyze all the SQL injections detected by the WAF. In this paper, we propose a method to identify the phase of a SQL injection in order to improve the efficiency of manual analysis. In the proposed method, only the partial SQL query included in the HTTP request related to the WAF alert is analyzed. The proposed method do not use the internal information of the attack target system and also do not modify the target system. The attack SQL query is dynamically analyzed using a emulator. Evaluation results revealed that the proposed method was able to correctly identify the intention with an accuracy of 78.1% and 73.8% for an artificial dataset and a real-world dataset respectively. We also show that the proposed method possibly reducing the operation time of manual analysis by identifying over 50% SQL injection attacks that does not cause any damage.

Keywords: SQL injection, WAF, SOC analyst, attack phase identification

1. はじめに

インターネットに公開されている Web アプリケーションは攻撃者からのアクセスが容易なため，日々大量の攻撃に晒されている．データベースを利用する Web アプリケーションにはオンラインショッピングやインターネットバンキングなどが存在し，顧客の個人情報やクレジットカード情報といった重要な情報を保持している．そのため，データベース内の情報を狙う Structured Query Language（SQL）インジェクションは Web アプリケーションに対する攻撃の中でも量が多く，深刻な影響をもたらす [1]．

SQL インジェクションの攻撃の脅威から Web アプリケーションを守るためにこれまでに攻撃検知や防御の研究は多く行われており [2], [3], [4]，多くの企業などで攻撃の検知・防御のために SQL インジェクションの検知機能が搭載された Web Application Firewall（WAF）が広く利用されている．WAF では正規表現などで表されたシグネチャを利用し，HTTP リクエストの内容がシグネチャに合致した際に，その HTTP リクエストを攻撃としての検知し，通信を遮断することが可能である．しかしながら，WAF の検知は完璧ではなく正常なリクエストを誤って攻撃として検知する誤検知が発生してしまう場合も存在する．Web アプリケーションの運用者は正常なリクエストの誤検知によって Web アプリケーションの可用性が損なわれることを懸念して，WAF の運用では攻撃を遮断せず検知のみを行う場合も少なくない．

WAF で攻撃を遮断をせず検知のみを行う場合，検知された攻撃に対して人手で分析を行い，被害が発生した場合はその攻撃への対処を行う必要がある．このような攻撃の分析をリアルタイムで行うためには Security Operation Center（SOC）と呼ばれる知識を持った人員と体制が必要となるため，攻撃の分析を Web アプリケーション運用者自ら行うのではなく，Managed Security Service Provider（MSSP）と呼ばれる外部の情報セキュリティの専門家に分析を委託することが増加している．

MSSP は SOC を構築し，SOC アナリストは顧客のネットワークに設置された WAF などのセキュリティ製品から送信されるアラートを分析してセキュリティインシデントがあった際は顧客へ通知する．攻撃対象の内部情報を得ることができれば，アラートと突き合わせることで攻撃による被害や影響の有無を把握することが可能である．しかしながら，攻撃対象のシステムの詳細な構成，アプリケーションの設計・実装，データベースのテーブルなどの設計など，攻撃対象のシステムの内部情報は機密性の観点から提供されない場合も多い．また，攻撃対象のシステムに対応した詳細な情報を収集するために，攻撃対象のシステムに手を加えることも可用性の観点から認められない場合が多い．そのため，SOC アナリストはセキュリティ製品からのアラートやトラフィックなど，限られた情報から攻撃の侵害や影響の有無を分析する必要がある．また，Web アプリケーションは攻撃者からのアクセスが容易で大量の攻撃に晒されているため，WAF は大量のアラートを発する．そのため，人手による分析では必要なアラートの取捨選択などを行い，効率的に分析を行うことが求められる．攻撃の成否が判明していれば効率的なアラートの取捨選択が可能であるが，攻撃の成否が分からない場合も多く，別の観点での効率化が必要である．

SQL インジェクションでは，情報漏洩といったシステムに被害をもたらすまでに攻撃が段階的に行われる．まず，攻撃者は攻撃対象のシステムに SQL インジェクションの脆弱性の有無の確認する．攻撃者は攻撃対象の Web アプリケーションがどのようなデータベースの構成になっているかといった情報を持っていないため，次に，システムやデータベーススキーマに関する情報の取得を行う．最後に，データベースの内容の取得する．このような一連のフェーズをたどって段階的に攻撃を仕掛けることが基本である．ただし，オープンソースの Web アプリケーションなど，事前に攻撃者が脆弱性やデータベースの情報を把握している場合には，初期のフェーズが省略され，最終フェーズであるデータベースの内容を取得する攻撃が最初に行われることもあるが，SQL インジェクションに攻撃フェーズが存在することには変わりない．

検知された攻撃がどのフェーズの攻撃であるかが分かれば，どのような被害につながりうる攻撃なのかが明らかになり，対処すべき WAF アラートの優先度付けや被害の調査，必要な対処の判断などに役立てることができる．たとえば，攻撃の初期フェーズである脆弱性の有無を確認する攻撃は情報漏洩などの被害を引き起こさないため，これらの攻撃を識別し，除外することができれば，より後期のフェーズである被害を引き起こす重要な攻撃に絞り込んで対処することができるようになる．また攻撃の数もフェーズが進行するに連れて減少するため，初期フェーズの攻撃を除外し，いかに後期フェーズの攻撃に注力するかが分析効率化の鍵となる．

WAF や既存の攻撃検知手法 [5], [6] では攻撃を検知することが可能だが，その攻撃のフェーズは判断できていない．SQL インジェクションを検知したうえで，攻撃の成否や漏洩したデータの分析を行う手法 [7] も存在するが，攻撃のフェーズに着目したものではない．

本稿では，顧客の機密性や可用性の制約がある MSSP の

[1] 日本電信電話株式会社
NTT, Musashino, Tokyo 180–8585, Japan
[2] 静岡大学
Shizuoka University, Hamamatsu, Shizuoka 432–8011, Japan
[†1] 現在，NTT コミュニケーションズ株式会社
Presently with NTT Communications Corporation
[a)] kotomi.kuroki@ntt.com

表 1 SQL インジェクション例
Table 1 Examples of SQL injection.

攻撃フェーズ	攻撃例
脆弱性偵察フェーズ	a' and SLEEP(5)--
	a' UNION SELECT CONCAT(CHAR(116),CHAR(101),CHAR(115),CHAR(116))--
	a' AND EXTRACTVALUE(1,CONCAT(0x5c73,(SELECT (ELT(1=1,1))),0x61))--
システム情報取得フェーズ	a' UNION SELECT VERSION()--
	a' UNION SELECT table_name FROM information_schema.tables--
DB 内容取得フェーズ	a' UNION SELECT name,password FROM users--

形態に鑑み，攻撃対象のシステムの情報を必要なく SQL インジェクションの攻撃のフェーズを識別する手法を提案する．提案手法では，インストールし手を加えていない初期状態の DataBase Management System（DBMS）を用い，攻撃に含まれる SQL クエリをこの初期状態 DBMS 上で実行した際の挙動を活用することで攻撃のフェーズを識別する．

本研究の貢献は以下のとおりである．

- 顧客の機密性や可用性の観点から監視対象のシステムへ手を加えたり，システムの情報を得ることが困難である MSSP の制約に鑑み，攻撃対象の内部情報と改変を必要としない SQL インジェクションの攻撃フェーズ識別手法を設計し，実装を行った．
- 提案手法を評価し，提案手法が，人工的に作成したデータセットに対して 78.1%，実環境で得られたデータセットに対して 73.8%の精度で SQL インジェクションの攻撃フェーズを正しく識別できることを示した．

2. SQL インジェクションにおける攻撃フェーズ

2.1 攻撃フェーズの分類

本稿では，SQL インジェクションの主な被害である情報漏洩を引き起こす攻撃に着目して，SQL インジェクションの攻撃のフェーズを以下の 3 つに分類する．

- **脆弱性偵察フェーズ**
- **システム情報取得フェーズ**
- **DB 内容取得フェーズ**

攻撃者は Web アプリケーションに利用されている DBMS の情報を持ち合わせていないため，SQL インジェクションを行う際は，まず，URL パラメタや HTTP ヘッダなど複数箇所に SQL クエリを挿入したリクエストを試行することで攻撃可能な脆弱性を探す偵察が行われる．悪用可能な脆弱性が見つかった場合，次に，情報取得を行うため，DBMS の種類やバージョンなどのシステムに関する情報，データベースの内容を取得するために必要なテーブル名やカラム名といったデータベースのスキーマ情報などを取得する攻撃が行われる．最後に，今まで得られた情報をもとに SQL クエリを組み立てて，最終的にデータベースの中のデータを取得する攻撃が行われる．

各攻撃フェーズの SQL インジェクションの例を**表 1** に示す．以下，それぞれの攻撃フェーズについて詳しく解説する．

2.1.1 脆弱性偵察フェーズ

攻撃対象の Web アプリケーションに対して，SQL インジェクションの脆弱性の有無を確認するための攻撃を指す．データベースの内容やシステム情報といった攻撃対象のシステム内に存在する情報が出力されるような攻撃ではなく，攻撃者が指定した文字列や真偽値が出力されるような攻撃となっており，出力が攻撃者の意図したものと一致するか否かで攻撃の成否，すなわち脆弱性の有無が確認される．その性質からインターネット空間上の脆弱性の有無が分からない Web アプリケーションに対して広域に行われる攻撃であるため，SQL インジェクションの中でも最も量が多い傾向にあるが，成功する攻撃の数は少なく，成功しても被害を引き起こさない．そのため，成否が分からない攻撃の中から脆弱性偵察フェーズの攻撃を識別し，対処から除外したり，対処の優先度を下げたりすることで，被害を引き起こす可能性のある攻撃に SOC アナリストを注力させることができる．表 1 の 1 つ目の攻撃例では，“SLEEP(5)” という関数によって HTTP レスポンスを遅延させることで，SQL インジェクションの可否を判断している．2 つ目の攻撃例では，`CONCAT(CHAR(116),CHAR(101),CHAR(115),CHAR(116))` を実行した結果である “test” という文字列が HTTP レスポンスに含まれるか否かで SQL インジェクションの可否を判断している．3 つ目の攻撃例では，特定のエラーが発生するような SQL クエリを挿入し，レスポンスにエラーメッセージが含まれるか否かで SQL インジェクションの可否を判断している．

2.1.2 システム情報取得フェーズ

DBMS のバージョン情報やホスト名，システムのユーザ名，スキーマ情報など，DBMS に関する情報を取得する攻撃を指す．後述の「DB 内容取得フェーズ」の攻撃ほどクリティカルな被害にはつながらないものの，脆弱性偵察フェーズの攻撃とは異なりクエリの実行結果にシステムに関する情報が含まれるため，攻撃の影響について詳細な調査と必要に応じて報告を行う必要がある．表 1 の攻撃例に

あげたような，“VERSION()” という関数によって DBMS のバージョン情報を取得するものや，特定の DBMS にデフォルトで存在する “information_schema.tables” などのシステムテーブルから攻撃対象のデータベースに存在するテーブル名を取得する攻撃などが存在する．

2.1.3 DB 内容取得フェーズ

アプリケーションやユーザが定義したテーブルに保存されている内容を取得する攻撃を指す．DBMS にデフォルトで存在するシステムテーブルの内容を取得する攻撃については**システム情報取得フェーズ**として扱う．このフェーズの攻撃はアプリケーションのユーザ ID やパスワードといった機微情報の漏洩を引き起こす攻撃であるため，このフェーズの攻撃を正しく識別することで，重大な被害をもたらす攻撃に優先的な対処を行うことができる．表 1 の攻撃例では，“users” というテーブルから “name” と “password” というカラムの内容を取得している．

2.2 攻撃フェーズ識別における課題

MSSP では顧客の機密性や可用性の観点から SQL インジェクションの攻撃フェーズを識別することには次の 2 つの課題が存在する．

1 つ目の課題は，攻撃対象のシステムで実行される SQL クエリの全容が把握できないことである．MSSP は顧客の機密性や可用性の制約から監視対象のデータベースのスキーマや内容といったシステムの内部情報の取得や監視対象のシステムの改変が困難である．また，攻撃対象のデータベースで実行される SQL クエリの全容は Web アプリケーションのソースコードに記述されているため，アラートに含まれる HTTP リクエスト情報から DBMS で実行される SQL クエリの全容を把握することはできず，攻撃者が挿入する部分的な SQL クエリのみをもとに攻撃フェーズを識別する必要がある．しかし，攻撃が挿入する部分的な SQL クエリは Web アプリケーションのソースコードに含まれる SQL クエリのテンプレートに組み込まれる形で実行されるため，攻撃者によって挿入される部分的な SQL クエリ単体ではデータベース上で SQL クエリとして実行できるものではない．

2 つ目の課題は，静的に攻撃フェーズ識別するための有用なシグネチャを作成することが難しいことである．顧客の機密性や可用性の観点からシステムの内部情報が得られないため，シグネチャをあらかじめ設計することができない．たとえば，攻撃対象のデータベースのテーブル構成が未知であれば，攻撃がシステム情報を狙ったものなのか，ユーザの情報を狙ったものなのか判別できない．また，脆弱性偵察フェーズの攻撃において送信される SQL クエリは攻撃対象のシステムに依存しないクエリであるが，他のフェーズの攻撃と共通する部分が多く，異なるのはクエリ内にテーブルの指定がないことだけなど，脆弱性偵察の攻撃のみに現れる部分を共通化してシグネチャを作成するというのは難しい．たとえば，「`-1 union select 1,2,3,4,5--`」や「`1 union all select null,concat(0x3a79716f3a,0x3a7a77783a),null,null`」といった脆弱性偵察の攻撃がある一方で，「`-5 union select 1,2,3,4,5,6,concat(admin,0x23,email,0x5D,loginname,0x7E,pass),8,9,0,1,2,3,4,5,6,7,8,9,0 from users--`」のような DB 内容の取得を行う攻撃が存在する．脆弱性偵察の攻撃と他の攻撃を区別できる点は，FROM 句でのテーブルの指定やカラムの指定の有無のみである．脆弱性偵察の攻撃はほかにも「`admin') AND (SELECT * FROM (SELECT(SLEEP(3-(IF(12=88,0,3)))))MBBc) AND ('`」のように FROM 句にサブクエリを入れているものなどもあり，既存のテーブルを指定したものか否か，静的なシグネチャを用いて識別することは困難である．このように，システムの内部情報を利用できない場合に，シグネチャで活用できる情報を整理し，それらの情報の範囲内でフェーズを識別するシグネチャを設計する必要がある．

3. 関連研究

SQL インジェクションについては，攻撃の仕組みや種類，対策など様々な研究が行われている [2], [3], [4], [8]．特に対策のために，脆弱性の検知や攻撃の検知・防御を目的とした多くの研究が存在する．脆弱性を検知する研究としては，たとえば，Web アプリケーションのソースコードを解析することで SQL インジェクションの脆弱性を検出する研究が存在する [9], [10], [11], [12]．攻撃による被害が発生する前に事前に脆弱性を検知して修正を行うことで，SQL インジェクションへの対策が可能となる．しかしながら，実際の Web アプリケーションでは事前に網羅的な検知ができておらず脆弱性が残っていることがあるため，発生した攻撃の検知・分析といった対応も必要となっている．

SQL インジェクションを検知する手法としては，WAF で行っているような正規表現のルールセットを用いたもののほかにも，機械学習を用いた検知手法 [5], [6] や，Web アプリケーションや DBMS に介入して実行される SQL クエリを解析することで SQL インジェクションを検知・防御する研究が行われている [13], [14], [15], [16], [17], [18]．たとえば，AMNESIA [13] は，Web アプリケーションのソースコードをもとに作成した正常な SQL クエリのモデルを用いて，攻撃対象の DBMS 上で実行される SQL クエリを検査することで SQL インジェクションの検知を行う．このような研究では，攻撃対象のシステムの内部情報が利用されている．ほかにも，検知率の向上に向けて，WAF による SQL インジェクションの検知を回避する攻撃を自動生成する研究 [19] や，正規表現で記述された WAF のルールセットを自動で修正する研究 [20] も行われている．これらの研究においては攻撃か否かを判定することに焦点が当

てられている．

SQL インジェクションを検知した後の分析に関わる研究としては，情報漏洩の被害に着目して漏洩したデータの解析を行う DIAVA [7] や，トラフィックから攻撃の成否判定を行う研究 [21] などが存在する．DIAVA は，正規表現を用いて SQL インジェクションの検知と成否判定を行い，さらにデータが漏洩していた場合には漏洩したデータの抽出と，漏洩したデータがハッシュ化されていた場合には解読の容易さを分析する．これらの研究は，本研究と同様にトラフィックベースで攻撃の分析を行うことを目的としている．しかしながら，攻撃のフェーズという観点での分析を行うものではない．

4. 提案手法

2.2 節で述べた課題をふまえて，攻撃者が HTTP リクエストに挿入した部分的な SQL クエリを解析対象とし，初期状態の DBMS をエミュレータとして用いた動的解析によって攻撃フェーズを特定する手法を提案する．

図 1 に提案手法の概要を示す．提案手法では，アラートに関連する HTTP リクエストの情報から抽出した攻撃者が挿入する SQL クエリを入力とする．HTTP リクエストが攻撃か否かの判別は WAF や既存の攻撃検知手法で実現可能である．攻撃者が挿入した SQL クエリは，たとえばパースした HTTP リクエストの要素ごとに既存の検知ルールを適用するという方法で抽出が可能である．提案手法の処理は 1. クエリ抽出，2. 動的解析の 2 つのステップで構成される．攻撃に含まれる部分的な SQL クエリはデータベースで実行される SQL クエリの一部でしかないため，そのままでは実行することができない．そのため，まずステップ 1 で部分的な SQL クエリから実行対象となる SQL クエリを抽出する．その後，実行可能な形に SQL クエリを整形し，ステップ 2 にて初期状態 DBMS でクエリを実行し，その際の挙動から攻撃フェーズを識別する．以下，各節にて各ステップの処理を詳説する．

4.1 クエリ抽出

攻撃者が HTTP リクエストに挿入した攻撃クエリは，攻撃対象のデータベースで実行されるクエリの一部でしかなく，そのままでは実行できないため，動的解析の前に実行可能なクエリの抽出と整形を行う．攻撃クエリの構造から実行可能な範囲のクエリを抽出するために，攻撃クエリに対して構文解析を行い，構文木の部分木単位でクエリを抽出する．既存の SQL の構文では不完全な攻撃クエリを構文解析することができないため，攻撃クエリに対して構文解析を行うための構文ルールは別途定義した．攻撃クエリは実行される SQL 文の全体ではなくあくまで攻撃者が挿入した部分的なクエリであり，括弧や引用符が不足している場合が多い．したがって，構文解析の際には必要に応じて括弧および引用符の補完を行う．

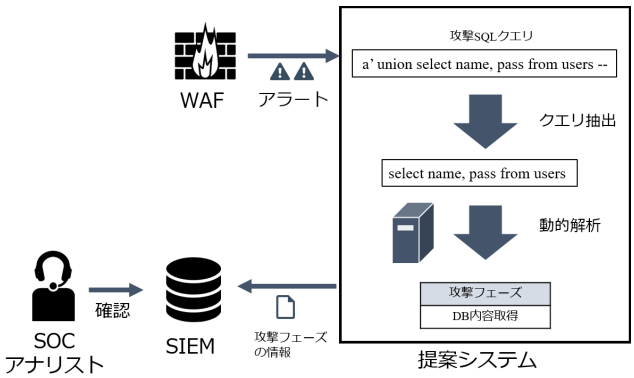


図 1 提案手法の概要図
Fig. 1 Overview of the proposed method.

括弧の補完としては，攻撃クエリ内に対にならない括弧がある場合に対応する括弧を先頭または末尾に追加する．引用符の補完としては，攻撃クエリに単一引用符や二重引用符が含まれており，かつそのままでは正しい構文となっておらず構文解析ができないものに対して先頭と末尾に補完を行う．

SQL インジェクションの攻撃クエリを構文解析するための文法の例を EBNF（Extended Backus–Naur Form）で記述したものを**図 2** に示す．本来の SQL の文法を参考に，部分的に抜き出されたクエリでも適用できるように定義したもので，⟨*sqli-query*⟩ を開始記号とする．図 2 中で，⟨*escaped_string*⟩ は引用符で囲まれた文字列，⟨*non_escaped_string*⟩ はテーブル名などの指定で使われるような引用符で囲まれていない文字列，⟨*number*⟩ は数値を表す．構文解析によって得られた構文木から部分木単位で抽出したクエリに対して動的解析を実施する．抽出する部分木は，図 2 における ⟨*statement*⟩ 以下を根に近い方から順に対象とする．

表 1 であげた攻撃のクエリ抽出例を**表 2** に示す．前述した構文解析によって得られた構文木から部分木単位で抽出することで，クエリの構造に基づいて適切な範囲を抽出することが可能となる．

4.2 動的解析

4.1 節にて抽出したクエリに対してエミュレーションを行い，その挙動から攻撃フェーズを識別する．識別の対象とする攻撃フェーズは 2 章で述べた**脆弱性偵察フェーズ**，**システム情報取得フェーズ**，**DB 内容取得フェーズ**の 3 つとする．

脆弱性偵察フェーズにある攻撃は，関数やサブクエリの実行結果にデータベースの内容などが含まれず，攻撃者の指定した固定の結果になるという特徴を持つ．そのため，実行時にテーブルへのアクセスの有無を確認してテーブルへのアクセスがないこと，“VERSION()” などのシステム

⟨*sqli-query*⟩ ::= ([⟨*base*⟩] ⟨*statements*⟩ | ⟨*items*⟩)
⟨*base*⟩ ::= ⟨*condition*⟩
⟨*statements*⟩ ::= ⟨*statement*⟩ | [⟨*statement*⟩] (';' ⟨*statement*⟩)+
⟨*statement*⟩ ::= ⟨*non-mod-statement*⟩ | ⟨*mod-statement*⟩
⟨*mod-statement*⟩ ::= ⟨*mod-command*⟩ ((⟨*clause-word*⟩ [⟨*items*⟩ [⟨*non-escaped-string*⟩]])+ | ⟨*items*⟩)
⟨*non-mod-statement*⟩ ::= (⟨*clause-word*⟩ [⟨*items*⟩ [⟨*non-escaped-string*⟩]]) [⟨*select-statement*⟩] | ⟨*select-statement*⟩
⟨*select-statement*⟩ ::= 'SELECT' [⟨*items*⟩ [⟨*non-escaped-string*⟩]] (⟨*clause-word*⟩ [⟨*items*⟩ [⟨*non-escaped-string*⟩]])*
⟨*items*⟩ ::= ⟨*condition*⟩ (',' ⟨*condition*⟩)*
⟨*condition*⟩ ::= ⟨*expression*⟩ [⟨*operand*⟩ ⟨*expression*⟩]
⟨*expression*⟩ ::= [⟨*object*⟩] (⟨*num-op*⟩ ⟨*object*⟩)+ | ⟨*object*⟩
⟨*object*⟩ ::= ⟨*function*⟩ | ⟨*sysfunction*⟩ | ⟨*parened-statement*⟩ | ⟨*value*⟩ | ⟨*end*⟩
⟨*value*⟩ ::= ['+' | '-'] (⟨*non-escaped-string*⟩ | ⟨*number*⟩ | ⟨*escaped-string*⟩ | '*' | ⟨*word*⟩ | '(' ⟨*value*⟩ ')')
⟨*parened-statement*⟩ ::= '(' (⟨*items*⟩ [⟨*statement*⟩] | ⟨*statement*⟩) ')'
⟨*function*⟩ ::= ⟨*non-escaped-string*⟩ '(' (⟨*items*⟩ [⟨*statement*⟩] | ⟨*statements*⟩) ')'
⟨*sysfunction*⟩ ::= ⟨*non-escaped-string*⟩ '(' ')'
⟨*num-op*⟩ ::= '*' | '+' | '-' | '/' | ...
⟨*clause-word*⟩ ::= 'AND' | 'BY' | 'FROM' | 'GROUP' | ...
⟨*mod-command*⟩ ::= 'DELETE' | 'GRANT' | 'INSERT' | ...
⟨*word*⟩ ::= 'NULL' | ...
⟨*op*⟩ ::= '!=' | '<>' | '=' | '>=' | '<=' | '>' | '<'

図 2 提案手法における攻撃クエリの構文例
Fig. 2 Syntax rule of attack SQL queries in the proposed method.

表 2 クエリ抽出例
Table 2 Query extraction examples.

攻撃クエリ	抽出クエリ
a' and SLEEP(5)--	SLEEP(5)
a' UNION SELECT CONCAT(CHAR(116),CHAR(101),CHAR(115),CHAR(116))--	SELECT CONCAT(CHAR(116),CHAR(101),CHAR(115),CHAR(116))
a' AND EXTRACTVALUE(1,CONCAT(0x5c73,(SELECT (ELT(1=1,1))),0x61))--	EXTRACTVALUE(1,CONCAT(0x5c73,(SELECT (ELT(1=1,1))), 0x61))
a' UNION SELECT VERSION()--	SELECT VERSION()
a' UNION SELECT table_name FROM information_schema.tables--	SELECT table_name FROM information_schema.tables
a' UNION SELECT name,password FROM users--	SELECT name,password FROM users

情報を取得する関数や環境変数が含まれていないこと，エラーが発生しないことを判断基準とする．

システム情報取得フェーズにある攻撃は，“VERSION()” などのシステム情報を取得する関数や環境変数が含まれていること，DBMS にデフォルトで存在するテーブルへのアクセスが行われるという特徴を持つ．そのため，実行時にテーブルへのアクセスの有無を確認してテーブルへのアクセスがある場合と，クエリ内に “VERSION()” などのシステム情報を取得する関数や環境変数が含まれている場合にシステム情報取得として判断できる．

DB 内容取得フェーズにある攻撃は，ユーザやアプリケーションによって作成されたテーブルへのアクセスが行われるという特徴を持つ．ユーザやアプリケーションによって作成されたテーブルは初期状態 DBMS には存在しないため，初期状態 DBMS での実行時にはテーブルが存在しない旨のエラーが発生する．したがって，初期状態 DBMS での実行時に，エラー番号から存在しないテーブルへアクセスするクエリであることを確認し，DB 内容取得として判断する．

つまり，クエリ実行時に DBMS の以下の挙動を取得することで攻撃フェーズの識別が実現できる．

- テーブルエラーの有無（存在しないテーブルを指定した際に発生するエラー）
- その他のエラーの有無
- テーブルへのアクセスの有無
- システム関数の有無

これらは初期状態の DBMS 上で実行した際に得られる情報を用いるため，攻撃対象のシステムの内部情報を必要とせず，識別することができる．

抽出したクエリを実行可能な SQL 文とするため，実行

表 3 識別ルール
Table 3 Identification rules.

攻撃フェーズ	テーブルエラー有無	その他エラー有無	テーブルアクセス有無	システム関数有無
脆弱性偵察	無	無	無	無
システム情報取得	無	無	有	-
	無	無	-	有
DB 内容取得	有	-	-	-

表 4 識別例
Table 4 Identification examples.

抽出クエリ	テーブル エラー有無	その他 エラー有無	テーブル アクセス有無	システム 関数有無	識別結果
`SLEEP(5)`	無	無	無	無	脆弱性偵察
`SELECT CONCAT(CHAR(116),CHAR(101), CHAR(115),CHAR(116))`	無	無	無	無	脆弱性偵察
`EXTRACTVALUE(1,CONCAT(0x5c73,(SELECT (ELT(1=1,1))),0x61))`	無	有	無	無	識別不可
`1,CONCAT(0x5c73,(SELECT (ELT(1=1,1))), 0x61)`	無	無	無	無	脆弱性偵察
`SELECT VERSION()`	無	無	無	有	システム情報取得
`SELECT table_name FROM information_schema.tables`	無	無	有	無	システム情報取得
`SELECT name,password FROM users`	有	無	無	無	DB 内容取得

時に必要に応じてクエリの修正を行う．具体的には，構文解析によって前後の不要な部分を除いたクエリの抽出を行い，クエリの前後に “SELECT” や “;” を追加することで実行可能な SQL 文とする．ここで SQL クエリの実行に用いるのはインストール後に何もデータベースの追加を行っていない初期状態の DBMS である．

初期状態 DBMS については監視対象のシステムで利用されている DBMS と同じ種類の DBMS を 1 つ用意することを想定している．複数の異なる種類の DBMS を利用している場合，種類ごとに初期 DBMS をそれぞれ用意して提案手法を適用することとなるが，複数の DBMS を利用する場合でも仮想環境やコンテナ技術などで DBMS の種類ごとに DBMS をインストールしただけの空の状態の DB を 1 つずつ用意するだけで済むため，導入・運用のコストは高くないと考えられる．本稿では初期状態 DBMS はオープンソースで広く利用されている MariaDB を利用して実装した．DBMS の種類により SQL クエリの構文や用意されている関数が若干異なるが．容易に対応可能であると考える．

これらの情報をもとに攻撃フェーズを識別するためのルールを**表 3** に示す．表 3 の条件にあてはまらない場合には，クエリ抽出時に得られた構文木をもとに，別の範囲のクエリを抽出して対象とする．これによって，意図的に特定のエラーを引き起こしてエラーメッセージの中に攻撃者が取得したい情報を出力させるような攻撃にも対応することが可能となる．

表 2 で抽出されたクエリの動的解析での識別例を**表 4** に示す．1 つ目と 2 つ目の例では，それぞれ SLEEP 関数と CONCAT 関数を実行しており，テーブルアクセスやエラーなどが発生しないことから脆弱性偵察と判断できる．3 つ目の例はエラーメッセージの中にクエリの実行結果が含まれることを利用した攻撃となっており，このクエリを実行するとエラーが発生し，この時点では識別不可となる．この場合，抽出するクエリの範囲を変更して改めて識別を行う．抽出する範囲を狭めたものが 4 つ目の例となっており，改めて実行することで，正しく脆弱性偵察と識別される．5 つ目の例では VERSION 関数を実行しており，システム関数が含まれること，エラーが発生していないことからシステム情報取得と判断できる．6 つ目の例では，MySQL にデフォルトで存在する information_schema テーブルから，データベースに存在するテーブルの名前を取得するクエリとなっており，テーブルへのアクセスが発生することからシステム情報取得と判断できる．7 つ目の例では，ユーザやアプリケーションが用意した users テーブルの内容を取得するクエリとなっており，用意した初期状態 DBMS で実行することでテーブルが存在しないことを示すエラーが発生することから，DB 内容取得と判断できる．

5. 評価

提案手法の識別精度について評価を行った．評価に用いたデータセットは，検証用の環境で攻撃を再現して作成した人工データセットと，実環境のトラフィックから作成した実環境データセットの 2 種類である．

表 5 人工データセット全体に対する識別結果
Table 5 Results for the artificial dataset.

識別結果	正解ラベル			合計
	脆弱性偵察	システム情報取得	DB 内容取得	
脆弱性偵察	9,381	3,678	8	13,067
システム情報取得	1,104	23,707	31	24,842
DB 内容取得	596	259	7,042	7,897
識別不可	3,581	1,511	502	5,594
合計	14,662	29,155	7,583	51,400
正解率（全体）				78.1%

5.1 人工データセット

実環境で発生する SQL インジェクションでは攻撃手法やフェーズに偏りがあるため，多様な攻撃に対する提案手法の精度を網羅的に評価するために人工データセットを作成した．データセットの作成にはオープンソースの SQL インジェクション脆弱性検査ツールである sqlmap [*1] を用いた．実験環境に用意した Web サーバに対して sqlmap を実行し，用意されている複数の攻撃手法を網羅的に利用して SQL インジェクションを収集した．また，sqlmap はオプションによって攻撃対象の Web サーバから取得する情報の種類を変更できるため，異なるオプションを指定することで攻撃フェーズの種類が異なる SQL インジェクションを生成した．具体的には，**脆弱性偵察フェーズ**に行う攻撃を生成する際は脆弱性の検査のみを行うよう指定した．**システム情報取得フェーズ**に行う攻撃を生成する際はバージョン情報，DBMS のユーザ情報，データベース・テーブル名，サーバのホスト名，テーブルのスキーマ情報をそれぞれ取得するよう指定した．**DB 内容取得フェーズ**に行う攻撃を生成する際は特定のデータベース・テーブルを指定して内容を取得するよう指定した．

5.2 実環境データセット

実環境で発生している攻撃に対する攻撃フェーズの識別精度の評価と，分析の効率化への貢献を明らかにするため，実ネットワーク環境のトラフィックから SQL インジェクションを収集してデータセットを作成した．観測対象は大学の 1 学部内で管理されている 42 個の Web サーバであり，実際に運用・公開されているものである．トラフィックを収集した期間は 2019 年 3 月 28 日から 2021 年 02 月 24 日の約 2 年間である．SQL インジェクションの検知には，OWASP Core Rule Set [*2] の SQL インジェクション用検知ルールを用いた．検知した SQL インジェクションから抽出した 22,289 件の攻撃クエリを実環境データセットとした．その中からランダムサンプリングした 1,500 件に対して目視での判断による正解ラベルとなる攻撃フェーズの付与を行い，攻撃フェーズの識別精度評価に用いた．

*1 http://sqlmap.org/
*2 https://coreruleset.org/

表 6 各攻撃フェーズの識別に対する適合率，再現率および F 値
Table 6 Precision and recall for each intention type.

攻撃フェーズの種類	適合率	再現率	F 値
脆弱性偵察	71.8%	64.0%	67.7%
システム情報取得	95.4%	81.3%	87.8%
DB 内容取得	89.2%	92.9%	91.0%

5.3 評価結果

人工データセットに対して提案手法を適用した結果を**表 5** に示す．各攻撃フェーズの種類について正しく識別された数を下線で示した．表 5 によると全体の 51,400 件のうち 40,130 件を正しく攻撃フェーズを識別し，正解率は約 78.1%だった．全体の 10.9%にあたる 5,594 件の識別不可となっている攻撃は，ステップ 1 のクエリ抽出の際に SQL クエリを構文解析する際にエラーが発生したものである．したがって，クエリ抽出の精度は 89.1%となる．

攻撃フェーズごとの識別精度の確認のため，表 5 の結果から各攻撃フェーズに対して算出した適合率，再現率，F 値を**表 6** に示す．適合率とは，当該フェーズとして識別したデータ数に対する正解データ数の割合である．再現率とは，当該フェーズとして識別されるべきデータ数に対する正解データ数の割合である．表 6 から，DB 内容取得フェーズの識別における適合率と再現率がそれぞれ 89.2%，92.9%とともに 9 割近い精度となっていることが分かる．これによって，特に重大な被害につながる DB 内容の漏洩を引き起こす攻撃の見逃しを抑えることができると考えられる．

次に，正解ラベル付きの実環境データセットに対して提案手法を適用した結果を**表 7**，実環境データセット全体に対して提案手法を適用した結果を**表 8** に示す．正解ラベルのある 1,500 件について適用した結果と，実環境データセット全体に対して適用した結果をそれぞれ示している．正解ラベルが「その他」となっている攻撃については，データベースの内容を書き換える攻撃や攻撃対象のサーバ内のファイルに書き込みを行う攻撃などといった，本研究における攻撃フェーズにあてはまらない攻撃が含まれている．正解ラベル付きのデータセットに対しては 1,500 件のうち 1,107 件を正しく識別でき，全体の正解率は 71.9%という結

表 7 正解ラベル付き実環境データセットに対する識別結果
Table 7 Results for the real-world dataset with label.

識別結果	正解ラベル				合計
	脆弱性偵察	システム情報取得	DB 内容取得	その他	
脆弱性偵察	612	184	0	0	796
システム情報取得	3	490	0	0	493
DB 内容取得	24	2	5	0	248
識別不可	107	66	0	7	180
合計	746	742	5	7	1,500
正解率（全体）					73.8%

表 8 全実環境データセットに対する識別結果
Table 8 Results for the real-world dataset.

攻撃フェーズ	識別結果
脆弱性偵察	11,885
システム情報取得	7,319
DB 内容取得	384
識別不可	2,701
合計	22,289

果となった，全データセットに対しては，全体の 53.3%にあたる 11,885 件が脆弱性偵察となっており，次いでシステム情報取得が全体の 32.8%にあたる 7,319 件，DB 内容取得が全体の 1.7%にあたる 384 件という結果となった．また，全体の 12.1%にあたる 2,718 件については，構文エラーなどにより識別ができなかった．

5.4 考察

表 5，表 7 に示したように，人工データセットにおいては全体の 10.9%にあたる 5,594 件の攻撃，実環境データセットにおいては全体の 12.1%にあたる 2,701 件の攻撃が識別不可となった．人工データセットにおいて識別不可となったもののうちおよそ 5 割は引用符で囲われることを想定している文字列が引用符で囲われずに攻撃クエリに含まれていたり，想定していない記号（“[]” の括弧など）が含まれていることで構文解析ができないことによって識別不可となっていた．残りの識別不可となった攻撃については，今回想定していない特定の DBMS でのみ利用できる構文や関数が含まれていることによってエラーとなったことなどが要因として考えられる．特に脆弱性偵察フェーズの攻撃が識別不可となった件数が多いのは，脆弱性偵察を行う攻撃の中に SQL の構文エラーを引き起こすことを意図した攻撃が含まれているためと考えられる．

また，表 5 によると脆弱性偵察の攻撃とシステム情報取得の攻撃を誤って識別してしまっている場合が多い．これは，システム変数から情報を取得する場合など，システム情報を取得する方法が網羅できていないことや，脆弱性偵察の中にも DBMS のシステムテーブルにアクセスする攻撃が含まれていることが要因としてあげられる．その他，脆弱性偵察の攻撃を誤って DB 内容取得と識別してしまっている場合については，脆弱性偵察の攻撃の中にランダムな文字列をカラム名として指定するような攻撃が含まれていることが要因としてあげられる．

表 7 によると，実環境における SQL インジェクションの約半分が初期の脆弱性偵察フェーズの攻撃であり，攻撃フェーズが進むにつれて攻撃数が減少していることが分かる．この結果から，本研究は SOC アナリストの分析業務において，WAF が検知した SQL インジェクションのうち半分の攻撃は対処不要と判断でき，SOC アナリストの分析稼働を半減することに貢献できると考える．また，検知した攻撃の中で DB 内容取得フェーズにある攻撃については攻撃が成功していれば重大な被害につながる可能性のある攻撃として優先的に対処をすることが可能となり，被害状況の把握や被害の最小化にも寄与すると考える．

6. 制約

SQL インジェクションには Web アプリケーションを不正に動作させる攻撃や，データベースの改竄を行う攻撃などが存在する．Web アプリケーションを不正に動作させる攻撃としては，たとえば ID とパスワードを入力するログインページにおいて，SQL インジェクションによってパスワードの検証を無効化する攻撃などがある．このような攻撃は，アプリケーションで実装されている判定ロジックなどを騙すことでアプリケーションの機能を不正に動作させるものであるため，アプリケーションの機能や実装によって攻撃の影響が大きく異なる．このような攻撃を識別するにはアプリケーションの機能や実装に関する情報が必要となるが，本手法ではこれらの情報が得られない状況を想定しているため対応できない．また，Web アプリケーション側に実装されている，INSERT や UPDATE などデータベースに変更を加える SQL コマンドを利用して改竄を行う攻撃についても同様の理由で本手法では対応できない．

7. おわりに

本稿では，攻撃分析の効率化に向けた，攻撃対象の内部情報に依存しない SQL インジェクションの攻撃フェーズ識別手法を提案した．提案手法では，HTTP リクエスト中

に含まれる部分的なSQL文のみを解析対象として，攻撃対象のシステムの内部情報の利用やシステムの改変を必要としないエミュレーションによる分析によって攻撃フェーズを識別する．識別精度の評価によって，人工データセットに対して78.1%，実環境のデータセットに対して73.8%の精度で正しく識別できたことを示した．また，実環境データセットに対する評価から，実環境における攻撃の多くが脆弱性偵察などの優先度の低い攻撃であり，クリティカルな被害をもたらすDB内容の取得を引き起こす攻撃の数が少ないことが分かった．このことから，SOCアナリストなどによる分析業務において，多くの攻撃を対処不要として絞り込み一部の重大な攻撃を優先的に対処することで，稼働を減らしつつ重大なインシデントへの迅速な対応を行うことへの貢献が期待できる．

参考文献

[1] European Union Agency for Cybersecurity (ENISA): ENISA Threat Landscape 2020 – Web application attacks (2020).

[2] Hu, J., Zhao, W. and Cui, Y.: A Survey on SQL Injection Attacks, Detection and Prevention, *Proc. 2020 12th International Conference on Machine Learning and Computing*, *ICMLC 2020*, pp.483–488, Association for Computing Machinery (2020).

[3] Alwan, Z. and Younis, M.: Detection and Prevention of SQL Injection Attack: A Survey, *International Journal of Computer Science and Mobile Computing*, Vol.68, pp.5–17 (2017).

[4] Dehariya, H., Shukla, P. and Ahirwar, M.: A Survey on Detection and Prevention Techniques of SQL Injection Attacks, *International Journal of Computer Applications*, Vol.137, pp.9–15 (2016).

[5] Sheykhkanloo, N.M.: A Learning-based Neural Network Model for the Detection and Classification of SQL Injection Attacks, *International Journal of Cyber Warfare and Terrorism*, Vol.7, No.2, pp.16–41 (2017).

[6] Kar, D., Panigrahi, S. and Sundararajan, S.: SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM, *Computers & Security*, Vol.60, pp.206–225 (2016).

[7] Gu, H., Zhang, J., Liu, T., Hu, M., Zhou, J., Wei, T. and Chen, M.: DIAVA: A Traffic-Based Framework for Detection of SQL Injection Attacks and Vulnerability Analysis of Leaked Data, *IEEE Trans. Reliability*, pp.1–15 (2019).

[8] William G.J. Halfond, J.V. and Orso, A.: A Classification of SQL Injection Attacks and Countermeasures, *Proc. IEEE International Symposium on Secure Software Engineering*, pp.13–15 (2006).

[9] Xie, Y. and Aiken, A.: Static Detection of Security Vulnerabilities in Scripting Languages, *Proc. 14th Conference on USENIX Security Symposium*, pp.179–192 (2006).

[10] Livshits, V.B. and Lam, M.S.: Finding Security Vulnerabilities in Java Applications with Static Analysis, *Proc. 14th Conference on USENIX Security Symposium*, p.18 (2005).

[11] Dahse, J. and Holz, T.: Simulation of Built-in PHP Features for Precise Static Code Analysis, *Symposium on Network and Distributed System Security* (*NDSS*) (2014).

[12] Dahse, J. and Holz, T.: Static Detection of Second-Order Vulnerabilities in Web Applications, *Proc. 23rd USENIX Conference on Security Symposium*, *SEC'14*, pp.989–1003, USENIX Association (2014).

[13] 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*, *ASE '05*, pp.174–183, ACM (2005).

[14] Buehrer, G., Weide, B.W. and Sivilotti, P.A.G.: Using Parse Tree Validation to Prevent SQL Injection Attacks, *Proc. 5th International Workshop on Software Engineering and Middleware*, *SEM '05*, pp.106–113 (2005).

[15] Bandhakavi, S., Bisht, P., Madhusudan, P. and Venkatakrishnan, V.N.: CANDID: Preventing Sql Injection Attacks Using Dynamic Candidate Evaluations, *Proc. 14th ACM Conference on Computer and Communications Security*, *CCS '07*, pp.12–24 (2007).

[16] Medeiros, I., Beatriz, M., Neves, N. and Correia, M.: SEPTIC: Detecting Injection Attacks and Vulnerabilities Inside the DBMS, *IEEE Trans. Reliability*, Vol.68, No.3, pp.1168–1188 (2019).

[17] Medeiros, I., Beatriz, M., Neves, N. and Correia, M.: Hacking the DBMS to Prevent Injection Attacks, *Proc. 6th ACM Conference on Data and Application Security and Privacy*, *CODASPY '16*, pp.295–306, Association for Computing Machinery (online), DOI: 10.1145/2857705.2857723 (2016).

[18] Jahanshahi, R., Doupé, A. and Egele, M.: You Shall Not Pass: Mitigating SQL Injection Attacks on Legacy Web Applications, *Proc. 15th ACM Asia Conference on Computer and Communications Security*, *ASIA CCS '20*, pp.445–457, Association for Computing Machinery (online), DOI: 10.1145/3320269.3384760 (2020).

[19] Appelt, D., Nguyen, C.D. and Briand, L.: Behind an Application Firewall, Are We Safe from SQL Injection Attacks?, *2015 IEEE 8th International Conference on Software Testing, Verification and Validation* (*ICST*), pp.1–10 (2015).

[20] Appelt, D., Panichella, A. and Briand, L.: Automatically Repairing Web Application Firewalls Based on Successful SQL Injection Attacks, *2017 IEEE 28th International Symposium on Software Reliability Engineering* (*ISSRE*), pp.339–350 (2017).

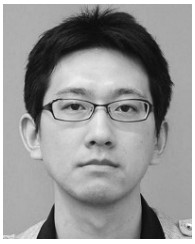
[21] 鐘本 楊，青木一史，三好 潤，嶋田 創，高倉弘喜：攻撃コードのエミュレーションに基づくWebアプリケーションに対する攻撃の成否判定手法，情報処理学会論文誌，Vol.60, No.3, pp.945–955 (2019).

黒木 琴海

2016年静岡大学情報学部卒業．2018年静岡大学総合科学技術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．現在，NTTコミュニケーションズ株式会社勤務．

鐘本 楊

2011 年名古屋大学工学部卒業．2013 年同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．2020 年京都大学情報学研究科博士後期課程修了．博士（情報学）．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 年東北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．

野口 靖浩 （正会員）

2008 年静岡大学大学院理工学研究科博士後期課程単位取得退学．2009 年静岡大学情報学部特任助教，同大学講師の後，2021 年より同大学准教授．自然言語処理，対話システム，知的教育支援システム等に興味を持つ．博士（工学）．人工知能学会，教育工学会，教育システム情報学会，IEEE，ACM 各会員．

西垣 正勝 （正会員）

1990 年静岡大学工学部光電機械工学科卒業．1995 年同大学大学院博士課程修了．日本学術振興会特別研究員（PD）を経て，1996 年静岡大学情報学部助手．同講師，助教授の後，2010 年より同創造科学技術大学院教授．博士（工学）．情報セキュリティ全般，特にヒューマニクスセキュリティ，メディアセキュリティ，ネットワークセキュリティ等に関する研究に従事．2013～2014 年情報処理学会コンピュータセキュリティ研究会主査，2019～2020 年情報環境領域委員長，2020 年調査研究運営委員長．2015～2016 年電子情報通信学会バイオメトリクス研究専門委員会委員長．2016～2020 年日本セキュリティマネジメント学会編集部会長，2021 年より副会長．本会フェロー．