Vol.2018-CSEC-81 No.24
Vol.2018-IOT-41 No.24
2018/5/18

Webアプリケーションテストを用いたSQLクエリのホワイトリスト自動作成手法

野村 孔命[1,a)]　力武 健次[1,2]　松本 亮介[1]

概要：
データベースの情報を利用して動作するWebアプリケーションでは，入力検証やクエリ発行処理の脆弱性により，開発者の想定していない不正クエリがデータベースに発行され機密情報を窃取される攻撃が発生する．このような攻撃に対して，Webアプリケーションがデータベースに発行するクエリのホワイトリストを作成し不正クエリの検知を行う方法がとられてきたが，Webアプリケーションの大規模化や実装言語の多様化に伴いホワイトリストの作成が難しくなっている．そのため，Webアプリケーション解析や運用時のクエリを用いた学習により生成を用いてホワイトリスト作り，検知する手法が提案されている．しかし，手法が実装言語依存の問題やWebアプリケーションの仕様変更の頻度が高いことによるホワイトリストの管理が難しい問題がある．本稿では，Webアプリケーションの動作を保証するためのテストがあり，Webアプリケーションの更新に追従してテストの更新が行われる開発プロセスが採用されていることを前提とし，テスト時に発行されるクエリからホワイトリストを自動作成する手法を提案する．Webアプリケーションの運用時には作成されたホワイトリストを用いて不正クエリを検知する．提案手法は，Webアプリケーションの複雑性や実装言語に依存せずにクエリのホワイトリストを自動作成することができ，新クエリが実装された場合もテストの更新に伴いホワイトリストが更新される．また，検知されたクエリはテストされていないクエリもしくは不正クエリであり，これらを早期に発見することで原因となるWebアプリケーションの脆弱性が長期化することを防ぐことができる．

Automatic Whitelist Generation for SQL Queries Using Web Application Test

Komei Nomura[1,a)]　Kenji Rikitake[1,2]　Ryosuke Matsumoto[1]

Abstract:
Database-driven Web applications are vulnerable to the attacks by the malicious queries which are not expected by the developers and are not correctly processed due to the incomplete input processing and the query issuing process. Defining the whitelist of the database queries issued by the Web applications has been conducted to detect the malicious queries. However, as the Web applications themselves become more complex and the implementation programming languages become more diverse, the whitelisting approach becomes more challenging to implement. In this paper, we propose a method to automatically generate a whitelist of database queries from those generated by the Web applications during the testing phase, provided that the application development process includes the operation testing. In the proposed method, the malicious queries are detected using the generated whitelist during the Web application is executed. Our method can automatically generate the whitelist of the queried regardless of the Web application complexity and independent of the implementation language and can update the whitelist as the test cases are updated when new queries are implemented. The detected queries are either untested or malicious and detecting the queries in the early stage of the attack may help an early removal of the respective Web application defects.

© 2018 Information Processing Society of Japan

Vol.2018-CSEC-81 No.24
Vol.2018-IOT-41 No.24
2018/5/18

1. はじめに

Web アプリケーションの脆弱性を利用した攻撃は後を絶たず [8]，Web サービスが保有する個人情報やサービス特有の機密情報を漏洩させるようなセキュリティインシデントが発生している [10]．このような攻撃は Web アプリケーションが利用するデータベースに不正クエリを発行することで行われ，場合によっては 1 つの不正クエリが大規模な情報漏洩を引き起こす．そのため，不正クエリはデータベースで実行される前に検知し，実行を停止する必要がある．また，PHP や Ruby など様々なプログラミング言語が実装に用いられる Web サービスにおいては，Web アプリケーションの実装に依存せず汎用的に利用できる不正クエリの検知方法が求められている．

ネットワークの攻撃検知方法として，異常検知と不正検知が一般的に用いられる [11]．異常検知では，統計的な手法や学習的な手法を用いて，正しい利用状況を表すプロファイルを作成し，不正なパターンの検知を行う．統計的な手法や学習的な手法を用いることから，プロファイルの作成に時間がかかり，即時に検知を行うことができない．また，プロファイルの正確性によっては，誤検知率が高くなってしまう．そのため，不正クエリ検知に用いた場合，検知漏れにより不正クエリを実行されてしまうことや，Web アプリケーションの正常なクエリ発行の実行を妨げてしまう問題が発生する．一方で，不正検知には，既知の不正なパターン（ブラックリスト），もしくは既知の正常なパターン（ホワイトリスト）を定義して，パターンマッチングを行い検知する方法がある．不正クエリをブラックリストを用いて検知する場合は，既知の不正クエリのパターンを全て定義できたとしても，未知の不正クエリを検知することができない．これ対して，ホワイトリストを用いる場合は，Web アプリケーションが発行するクエリを定義することで，未知の不正クエリにも対応可能である．

Web アプリケーションが発行するクエリのホワイトリストを定義することは，Web アプリケーションの大規模化や複雑化などの理由で，ホワイトリストの作成が困難となっている．そのため，Web アプリケーションが発行するクエリのホワイトリストを自動で作成する手法が提案されている [2][7]．しかし，これらの手法は，ホワイトリスト作成のための学習期間が必要であることによる検知の即時性の課題や，Web アプリケーションの実装言語に依存してしまう課題がある．

[1] GMO ペパボ株式会社 ペパボ研究所
Pepabo R&D Institute, GMO Pepabo, Inc., Tenjin, Chuo ku, Fukuoka 810-0001 Japan
[2] 力武健次技術士事務所
Kenji Rikitake Professional Engineer's Office, Toyonaka City, Osaka 560-0043 Japan
[a)] komei.nomura@pepabo.com

様々な言語で実装され，改修が頻繁に行われる Web サービスにおいて，不正クエリへの対策は重要であるが，対策を行うことでサービスの運営を妨げることは避けたい．そのため，導入時のシステムの構成や開発プロセスへの影響を抑えつつ，不正クエリへの対策を行うことが求められる．このことから，Web アプリケーションの実装言語に依存せず，既存のシステム構成や開発プロセスに大きく影響を与えない不正クエリへの対策が必要となる．

本研究では，Web アプリケーションの実装に依存せず，Web アプリケーションの運用前に，自動でホワイトリストを作成するための手法について述べる．提案手法は，開発プロセスに自動テストが採用されている状況において，Web アプリケーション運用前のテスト実行時に Web アプリケーションが発行するクエリを収集してホワイトリストを自動で作成する．テスト時のクエリを収集することよって，開発者が想定する Web アプリケーションの動作の過程で生じるクエリがホワイトリストに登録される．提案手法によって検知されたクエリは開発者が想定できていないクエリであり，そのようなクエリはテスト時に発行されていないもしくは不正クエリに限定され，テストカバレッジを向上させることが不正クエリの検知精度向上につながる．

Web アプリケーションの実装に依存しないようにするために，提案手法はデータベースの前段にデータベースプロキシを配置することでクエリの収集を行い，ホワイトリストに定義する方法を検討している．

本稿の構成を述べる．2 章では，Web アプリケーション利用するデータベースへの不正クエリ検知の課題を整理する．3 章では，提案手法の開発プロセスにおける位置付けと提案手法の設計について述べ，提案手法の考察を述べる．4 章でまとめを述べる．

2. Web アプリケーションが利用するデータベースへの不正クエリ検知の課題

Web アプリケーションが利用するデータベースへの不正クエリ検知の課題を整理する．ネットワークの攻撃検知方法として，異常検知と不正検知が一般的に用いられる [11]．異常検知は，統計的な手法や学習的な手法を用いて，正しい利用状況を表すプロファイルを作成し，収集したデータとプロファイルを比較しその差が大きい場合を不正とみなす．統計的な手法や学習的な手法を用いることから，プロファイルを作成するのに時間がかかる課題があり，不正クエリ検知に用いる場合は，検知できない期間が発生し，不正クエリを実行される危険性がある．また，プロファイルの正確性によっては，不正なパターンを正常なパターン，もしくは正常なパターンを不正なパターンと誤検知することが多い．そのため，不正クエリ検知において，不正なパターンを正常なパターンと誤検知した場合は，不正クエリを実行されてしまい，正常なパターンを不正なパターンと

© 2018 Information Processing Society of Japan

誤検知した場合は，Web アプリケーションが発行する正常なクエリの実行を妨げてしまう．一方で，不正検知は，既知の不正を定義し，収集したデータとパターンマッチングを行い，不正なパターンを検知する．不正検知には，不正なパターンをブラックリストとして定義しパターンがマッチしたものを不正とみなす方法や，正常なパターンをホワイトリストとして定義しパターンがマッチしないものを不正とみなす方法がある．ブラックリストを用いて検知する場合，全ての不正のパターンをブラックリストに定義すること難しく，定義できたとしても未知の不正パターンに対応することができない．また，一般的な不正パターンが提供されている場合もあり [5]，この場合，導入は容易になるが，不正パターンの更新は提供側任せになるため，新たな不正パターンへの対応が遅れる場合が考えられる．ホワイトリストを用いて不正クエリを検知する場合，Web アプリケーションが発行するクエリのパターンを正常なパターンとして定義し，それ以外のクエリを不正クエリとみなすことで，未知の不正クエリにも対応できる，ホワイトリストに定義する Web アプリケーションが発行するクエリのパターンには，Web アプリケーションが発行するクエリはユーザからの入力により変化するため，クエリのユーザ入力部分をプレースホルダに置き換えたクエリ構造（クエリダイジェスト）が用いられることがある [1]．ホワイトリストを手動で定義する場合，開発者は Web アプリケーションのソースコードから全てのクエリ発行処理を特定し，そこから発行されるクエリダイジェストを把握しなければならず，開発者への負担は増加する．さらに，Web アプリケーションの以下の特徴から，手動で Web アプリケーションが発行するクエリをホワイトリストに定義することが困難になっており，開発者への負担は増加する傾向にある．

(1) Web アプリケーションの大規模化
(2) Web アプリケーションの複雑化
(3) Web アプリケーションの更新頻度が高い [9]
(4) Web アプリケーションの実装にオブジェクトリレーショナルマッピング（ORM）[6] の利用

（1）（2）により，Web アプリケーションのクエリ発行処理が増加するうえに，条件分岐によってクエリ発行処理が複雑化し，ホワイトリストを作成するための開発者の負担が増大する．（3）により，Web アプリケーションが発行するクエリダイジェストが変化する頻度が向上し，その度にホワイトリストを更新することは開発者の負担を増大させる．これらのことから，Web アプリケーションの開発を行いながら，ホワイトリストを定義することは，開発者にとって大きな負担となる．さらに，（4）により，開発者は Web アプリケーションが発行するクエリを意識することが少なくなる．ORM[6] はオブジェクト指向言語におけるオブジェクトとデータベースのレコードを関連づける．これにより，開発者はデータベースのレコードをオブジェクトとして扱えるため，直接 SQL 文を記述することが少なくなる．例えば，Web アプリケーションの実装に Web アプリケーションフレームワークである Ruby on Rails を用いた場合，ORM として ActiveRecord が利用される [4]．開発者は，ActiveRecord を用いて，データベースのレコードと Ruby のクラスオブジェクトを関連づけることで，クラスオブジェクトを用いてクエリの発行処理を記述できる．クラスオブジェクト User がデータベースの users テーブルのレコードに関連付けされている場合の Ruby コードと発行されるクエリは表 1 のようになる．このように，開発者は Web アプリケーションが発行するクエリを意識する必要がなくなるため，ホワイトリストを手動で定義することは，ORM を用いた開発の妨げとなる．

表 1 Ruby コードと発行されるクエリの対応例
Table 1 Example of Ruby code and issued query

Ruby コード	発行されるクエリ
User.find(1)	SELECT * FROM users WHERE (users.id = 1) LIMIT 1
User.first	SELECT * FROM users ORDER BY users.id ASC LIMIT 1

Web アプリケーションの運用時に発行されるクエリを収集し，構文解析を行いクエリダイジェストに変換することで，ホワイトリストを定義する手法が提案されている [2]．この手法には，クエリを収集しホワイトリストを作る学習モードと作成したホワイトリストを用いて検知を行う検知モードがあり，学習モード中は検知を行うことができない．この手法を用いることで，Web アプリケーションの運用時に自動でホワイトリストを定義することができる．しかし，ホワイトリストの作成には学習期間を要するため，Web アプリケーションの更新頻度が高い特性から，頻繁に再学習を行わなければならず，その都度検知を行えない期間が発生してしまう．そのため，学習期間中に不正クエリを発行された場合は，検知することができず実行されてしまう．これは，Web アプリケーション運用時にホワイトリストを作成していることが原因であり，Web アプリケーションデプロイ後即時に不正クエリの検知を行うためには，運用前の段階でホワイトリストを作成する必要がある．

Web アプリケーションのソースコードからクエリの発行処理を特定し，クエリ発行処理を解析することで，ホワイトリストを自動で作成する手法が提案されている [7]．この手法を用いることで，Web アプリケーション運用前にホワイトリストを作成することができ，Web アプリケーションデプロイ後即時に不正クエリの検知を行える．一方で，この手法では，ソースコードのクエリ発行処理の解析を行うため，Web アプリケーションの実装言語ごとに解析器を実装しなければならない．これは Web アプリケーションの実装言語が多様化していることと，様々な ORM が利用されるようになっていることが原因で課題となる．筆者が所

© 2018 Information Processing Society of Japan

属する GMO ペパボ株式会社では，PHP や Ruby のような様々な言語や Web アプリケーションフレームワークを用いて Web サービスの開発が行われている．このような状況において，不正クエリへの対策をすることは重要であるが，対策を行うことによってサービスの運営を妨げることは避けたい．そのため，導入時のシステムの構成や開発プロセスへの影響を抑えつつ，Web アプリケーションの特性などに依存しない汎用的な不正クエリへの対策を行うことが求められる．このことから，Web アプリケーションの実装言語に依存せず汎用的に利用でき，かつ，現状のシステム構成や開発プロセスに影響の少ない対策が必要となる．

3. 提案手法

2 で述べた課題を解決するために，提案手法は以下の要件を満たす必要がある．

- Web アプリケーションが発行するクエリのホワイトリストを自動で作成できる
- Web アプリケーションの運用前にホワイトリストを作成できる
- Web アプリケーションの実装に依存せずホワイトリストを作成できる

提案手法は，開発プロセスへの影響を抑えつつ，Web アプリケーションのデプロイ後即時に不正クエリの検知を行うために，開発プロセスに自動テストが採用されている場合を想定し，テスト時に発行するクエリを利用してホワイトリストを自動で作成する．また，提案手法は，導入時のシステム構成への影響を抑えつつ，Web アプリケーションの実装に依存せず，ホワイトリスト作成を行うために，データベースの前段にデータベースプロキシを配置しクエリを収集する．

3.1 自動テストを採用した開発プロセスにおける提案手法の位置付け

自動テストを採用した開発プロセスを図 1 に示す．

図 1 で示した開発プロセスについて説明する．まず，開発者は Web アプリケーションの新機能の開発や既存機能の修正を行う．次に，開発した機能が既存の他の機能に影響を与えている場合や，開発した機能に対してのテストが行われていない状態が発生している場合は，開発者はテストコードの記述を行う．ここで，テストコードには，Web アプリケーションをテストするときの実行手順であるテストケースと期待される動作結果が記述されており，Web アプリケーションが仕様通りに動作しているかの検証を行う．そして，自動テストの段階で，全てのテストコードを用いてテストを行う．このとき，テストが失敗した場合は，開発した機能の動作が仕様通りでない，もしくはテストコードの記述，すなわち仕様の定義に誤りがあることが分かる．この場合，開発者は原因を特定し，Web アプリケーション

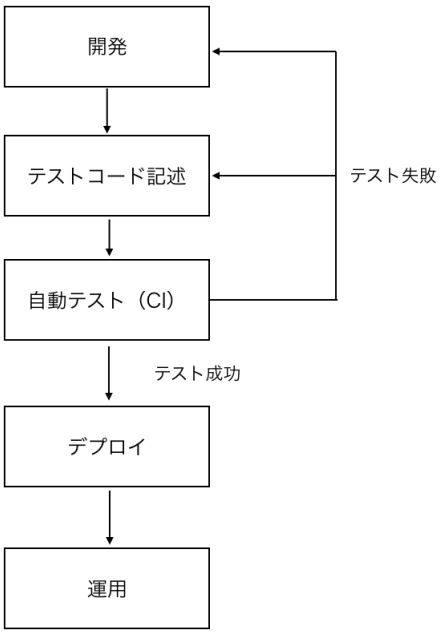

図 1 自動テストを用いた Web アプリケーションの開発プロセス
Fig. 1 Web application development process using Automatic testing

のソースコードもしくはテストコードの修正を行う．テストが成功した場合は，開発した機能が仕様通りに動作していることを確認できたとみなし，本番環境に Web アプリケーションをデプロイし運用する．

この開発プロセスの特徴は，開発された機能に対して，開発者が想定する Web アプリケーションの動作をテストコードに記述していくことであり，これにより Web アプリケーションの動作を保障する．しかし，テストコードに記述されているテストケースには漏れが生じる可能性があるため，機能が定義したテストケースに対して正常に動作していないことは分かるが，機能の動作に欠陥がないこと保証することはできない．このことから，テスト時に発行されるクエリは，開発者が想定した Web アプリケーションの動作の過程で発行されたクエリであり，開発者が想定できているクエリである．そのため，提案手法は Web アプリケーション運用時における開発者の想定できていないクエリを検知することができる．

提案手法は，図 1 の開発プロセスの自動テスト時に Web アプリケーションが発行するクエリのホワイトリストが作成できており，Web アプリケーションのデプロイ時にホワイトリストの更新を行うことができる．これにより，Web アプリケーションの仕様変更により，発行されるクエリダイジェストに変化があったとしても，変化に追従しホワイトリストを更新することができる．

3.2 提案手法の設計

提案手法は，テスト時と Web アプリケーション運用時において，どちらも同じアーキテクチャを用いて，テスト

© 2018 Information Processing Society of Japan

時の Web アプリケーションが発行するクエリの収集と運用時の発行されるクエリの監視を行う．提案手法のテスト時のホワイト作成フローと Web アプリケーション運用時の検知フローを図 2 を用いて説明する．

テスト時のホワイトリスト作成フローを説明する．図 2 において，まず，テストが実行されることにより，テストコードに基づいて Web アプリケーションが動作する．次に，その動作の過程で，データベースにアクセスする処理がある場合，Web アプリケーションはデータベースにクエリを発行する．このとき，データベースの前段に配置されているデータベースプロキシがクエリを受け取り，受け取ったクエリに対して構文解析行いクエリダイジェストに変換し記録する．データベースプロキシが受け取ったクエリはそのままデータベースに渡すことで，テストの実行に支障が出ないようにする．テスト終了後，データベースプロキシに保管されているクエリダイジェストの一覧がホワイトリストとなる．

Web アプリケーション運用時の検知フローについて説明する．図 2 において，まず，Web アプリケーションがユーザからの入力を受けクエリを発行する．次に，発行されたクエリをデータベースプロキシが受け取り，構文解析を行いクエリダイジェストに変換し，ホワイトリストと照合する．このとき，発行されたクエリのクエリダイジェストがホワイトリストに定義されていた場合，そのクエリを正常とみなしデータベースに渡し実行する．発行されたクエリのクエリダイジェストがホワイトリストに定義されていなかった場合は，クエリをデータベースに渡さず実行を停止する．

3.3 提案手法の考察

提案手法によって検知できるクエリについて述べる．提案手法は，テスト時のクエリを用いてホワイトリストを作成しているため，これを用いて検知されるクエリはテスト時に発行されたクエリダイジェストに一致しないものである．つまり，提案手法によって作成されたホワイトリストで検知されたクエリは，テストコードを記述した開発者が想定できていないテストケースが，運用時に実行されたことによって生じた想定できていないクエリである．このようなクエリには，テスト不足により検知されたクエリと不正クエリが含まれている．図 3 にクエリの内包関係の概略図を示す．

図 3 より，Web アプリケーションのテストカバレッジを向上させることによって，開発者が想定できているクエリ領域を Web アプリケーションが発行するクエリ領域に近づけることができ，その結果，不正クエリの検知精度向上に繋がると考えられる．テストカバレッジを向上させるためには，検知されたクエリの中からテスト不足により検知されたクエリであることを判断し開発者に知らせる必要がある．現状，提案手法では，検知されたクエリがテスト不足により検知されたクエリか不正クエリかを判断することはできないため，この判断は開発者に任せるしかない．そのため，今後の課題として，この判断を行い開発者に知らせるための方法の検討が必要である．

テストコードを記述することとホワイトリストにクエリダイジェストを手動で定義することの開発者への負担の差について述べる．テストコードには想定される Web アプリケーション動作であるテストケースを記述していくのに対して，ホワイトリストには Web アプリケーションが動作の過程で発行するクエリダイジェストを定義していく．テストコードを書く場合，開発者は Web アプリケーションの動作を意識する．これに対して，ホワイトリストを作成する場合，開発者は Web アプリケーションの動作を理解したうえで，その過程で発行されるクエリを意識しなければならない．そのため，テストコードを記述していくことの方が開発者への負担が少ないと考えられる．

提案手法で検知できない不正クエリについて述べる．提案手法のホワイトリストにはクエリのユーザ入力部分をプレースホルダーに置き換えたクエリダイジェストが定義されており，ユーザ入力部分が不正かどうかの判断は行なっていない．そのため，クエリダイジェストが同じであるが，ユーザ入力部分が不正であるようなクエリを検知することはできない．例えば，セッションハイジャックにより，あるユーザのアカウントが乗っ取られてしまい，そのユーザの個人情報を取得するようなクエリを発行される状況が考えられる．この場合，Web アプケーションから発行されるクエリは，ホワイトリストに定義されているクエリダイジェストと同じとなるため，提案手法では検知することができない．そのため，このようなクエリに対しては，提案手法とは別に対策を検討する必要がある．

Web アプリケーション運用時の検知の実装について述べる．Web アプリケーション運用時の検知には，発行されたクエリとホワイトリストの照合を行うため，この処理の実装方法が Web アプリケーションとデータベース間のレイテンシーに影響を与えると予想される．Web アプリケーションから発行されたクエリに対してホワイトリストを全探索した場合，ホワイトリストの照合処理の計算量はホワイトリストに定義されているクエリダイジェスト数 n に対して $O(n)$ となる．そのため，Web アプリケーションとデータベース間のレイテンシーが増大することが予想される．これを避けるために，ホワイトリストの作成時には，クエリダイジェストをキーとしてハッシュテーブルに登録しておき，運用時のホワイトリストの照合処理を行うことで，Web アプリケーションとデータベース間のレイテンシーの増大を抑えられると考えられる．

© 2018 Information Processing Society of Japan

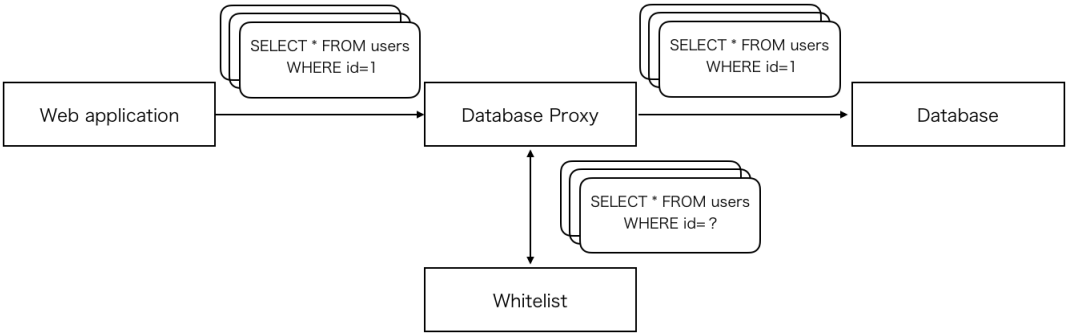

図 2　提案手法のアーキテクチャ
Fig. 2　Propose method architecture

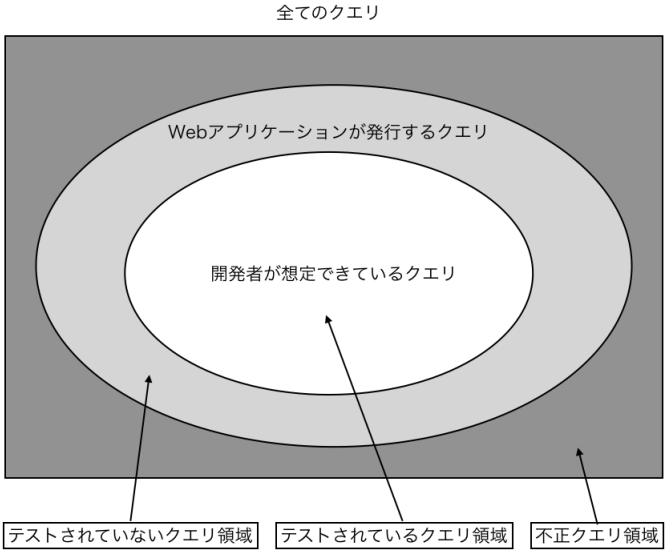

図 3　クエリの内包関係
Fig. 3　Relationship of queries

4. まとめ

本稿では，Web アプリケーションが利用するデータベースに対して個人情報を窃取するような不正クエリを防ぐために，開発プロセスに着目し，テスト時に Web アプリケーションが発行するクエリを収集し，Web アプリケーションが発行するクエリのホワイトリストを自動で作成する手法を提案し，手法の設計について述べた．今後は，提案手法を実装し，テストカバレッジに対する不正クエリの検知精度の検証や導入による Web アプリケーションとデータベース間のレイテンシーへの影響の検証などを進めていく．また，今後の課題として，検知されたクエリがテスト不足により検知されたクエリか不正クエリかを判断する方法や，提案手法では検知できないクエリダイジェストが同一の不正クエリへの対策を検討が挙げられる．

参考文献

[1] D Kar, S Panigrahi, Prevention of SQL Injection attack using query transformation and hashing, Advance Computing Conference (IACC), 2013.
[2] F. Valeur, D. Mutz and G. Vigna, A Learning-Based Approach to the Detection of SQL Attacks In Proceedings of the Conference on Detection of Intrusions and Malware Vulnerability Assessment (DIMVA), July 2005.
[3] Martin Fowler, David Rice, Matthew Foemmel, Edward Hieatt, Robert Mee, Randy Stafford, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.
[4] M Bachle, P Kirchberg, Ruby on rails, 2007.
[5] OWASP ModSecurity Core Rule Set (CRS), https://modsecurity.org/crs/.
[6] Scott W. Ambler, Mapping objects to relational databases What you need to know and why Ronin International, July 2000.
[7] William G.J. Halfond and Alessandro Orso, AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks, In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (ASE '05), ACM, New York, NY, USA, 174-183, 2005.
[8] IPA, 安全なウェブサイトの作り方 改訂第 7 版, 2015 年, 3 月.
[9] 青井 翔平, 坂本 一憲, 鷲崎 弘宜, 深澤 良彰, DePoT:Web アプリケーションテストにおけるテストコード自動生成テスティングフレームワーク, 情報処理学会論文誌 Vol.56 No.3 835846, March, 2015.
[10] 情報セキュリティインシデントに関する調査報告書 個人情報漏洩編 第 1.2 版, NPO 日本セキュリティ協会セキュリティ被害調査ワーキンググループ, 長崎県立大学 情報システム学部情報セキュリティ学科, 2017 年 6 月.
[11] 藤田 直行, 侵入検知に関する誤検知低減の研究動向, 電子情報通信学会論文誌 B Vol.J89B No.4 pp.402411, 2006.

© 2018 Information Processing Society of Japan