Vol.2019-DPS-181 No.5
2019/12/26

WordPress におけるプラグイン透過な SQL インジェクション防御策の提案

向佐裕貴[†1]　中村章人[†1]

概要：Web サイトの構築・管理にコンテンツ管理システム（CMS）が広く利用されている．CMS は HTML や JavaScript などの専門知識がなくても高品質な Web ページを作成でき，管理機能によりサイトの運用を容易にする．CMS は多種多様なものがあるが，WordPress は世界の Web サイトで約 3 割のシェアを持つ．本論文では，Web サイトの基盤システムとして広く用いられる WordPress のセキュリティ対策方法について述べる．情報漏えいや Web ページの改ざん，認証回避などの深刻な損害をもたらす SQL インジェクション（SQLI）攻撃に着目し，その解決策を示す．提案手法は，WordPress の実装言語である PHP の言語処理系を拡張することにより，アプリケーションコードに SQLI 脆弱性があったとしても，危険な SQL 文を無害化してデータベースアクセスを安全に実行する．また，WordPress に限らず，あらゆる PHP のアプリケーションコードにおける SQLI 脆弱性の防御策としても利用できる．

キーワード：SQL インジェクション，脆弱性，CMS，情報漏えい，不正アクセス

Plugin-Transparent SQL Injection Defense Method for WordPress

YUKI MUKASA[†1]　AKIHITO NAKAMURA[†1]

Abstract: Content Management System (CMS) is widely used for construction and management of Web sites. CMS helps both well-experienced and less-experienced persons to create high quality Web pages and manage sites. Among the many different kinds of CMSs, WordPress is used by 35% of all the Web sites. In this paper, we discuss how to defend WordPress against SQL injection (SQLI) attacks in a fundamental way. SQLI vulnerabilities can typically lead to confidentiality and integrity failures: exposure, defacement, or destruction of information. In addition, user authentication and/or authorization could be ruined. The proposed defense method extends PHP runtime system to detoxify dangerous SQL statements even if the application code is vulnerable to SQLI. Our method eliminates the problems of programmer involvement, false negatives or positives, and additional infrastructures, while it is defensible against the most types of SQLI attacks. The method can be also applied to other PHP-based systems.

Keywords: SQL Injection, software vulnerability, CMS, information leakage, unauthorized access

1. はじめに

Web サイトの構築と管理にコンテンツ管理システム（Content Management System: CMS）が広く利用されている．CMS は HTML や JavaScript などの専門知識がなくても高品質な Web ページを作成でき，その管理機能を利用すれば運用を容易にできる．また，拡張機能をプラグインとして組み込める機構が用意されており，プログラミングせずとも容易にカスタマイズできる．本稿執筆時点（2019 年 11 月）で，世界の Web サイトの 56.7%が CMS を利用しており，その中で WordPress[1]の利用率が 61.8%で最大である [2]．WordPress 以外の Joomla![3]や Drupal[4]等の CMS のシェアは数%に留まる．一方，Web サイトにおけるサーバ側のプログラミング言語では，PHP が 78.9%を占める[5]．PHP は WordPress の実装にも用いられている．

CMS は，Web ページ生成方式の違いにより，静的 CMS と動的 CMS とに分類できる[6]．静的 CMS は，データベース（DB）に保存したコンテンツから Web ページを一括で生成する．一方，動的 CMS は，ユーザのリクエストに応じて DB に保存されているコンテンツを取得してプログラムで処理し，動的に Web ページを生成する．前者には Movable Type[7], 後者には WordPress [1]や Drupal [4]等がある．静的 CMS は，応答時間や負荷耐性に優れるが，コンテンツ更新のリアルタイム性や対話性に欠ける．動的 CMS はこれらと反対の特性を持つ．セキュリティの観点から比較すると，動的 CMS はより脆弱になりやすい．なぜならば，実行時にさまざまなプログラムが実行される上に，DB へのアクセスが発生するからである．

SQL インジェクション（SQL Injection: SQLI）は，DB を利用するシステムにおける最も危険な脆弱性の一種である [8,9]．SQLI 脆弱性は，外部入力値を SQL 文の一部に組み込む際の不適切な処理に起因する．SQLI 攻撃が成功すると，不正な SQL 文が DB で実行される．その結果，一般的に機密性または完全性が侵害され，DB に記憶されている情報の漏えい，改ざん，または消去を引き起こす．更に，ユーザ認証を回避されたり，任意のコマンドを実行されたりする可能性もある．

Web アプリケーションにおける最も深刻なセキュリティリスクを列挙した OWASP Top 10 において，SQLI は第 1 位

†1 会津大学
University of Aizu

ⓒ2019 Information Processing Society of Japan

に位置づけられている[10]．同様に最も危険なソフトウェアエラーを列挙した CWE Top 25 においても第 6 位に位置する[11]．また，2017 年以降に CVE ID を付与された公知の脆弱性の内，SQLI に分類されるものは年間約 400 件，全脆弱性数の約 3%を占める[12]．また，PHP を利用したプロダクトの SQLI 脆弱性は，毎年 200 件以上に上る．SQLI の深刻度及び発生数を鑑みると，対策の必要性は明らかである．

本論文では，Web サイトの構築と運用に広く利用されている WordPress を対象とした SQLI 攻撃に対する防御策を示す．提案手法は，WordPress の実装言語である PHP の言語処理系を拡張して，実行時に危険な SQL 文を無害化する．外部入力値を確実に追跡し，必要に応じて SQL 文の実行方法を変更することで，プラグイン等のアプリケーションコードが SQLI 脆弱な場合であっても DB アクセスを安全に実行できる．本手法の実装コードは，わずかな修正により，WordPress 以外の PHP で実装されたシステムでも同様に SQLI 攻撃を防止できる．

本論文の構成は以下のとおりである．まず，第 2 章で SQLI 脆弱性とその攻撃方法及び基本的な防御方法を説明する．第 3 章では，SQLI 防御策に関する関連研究について述べる．第 4 章と第 5 章で，我々の提案手法と WordPress 上での実装方法を述べる．第 6 章に提案手法及び実装の評価結果を示す．最後に第 7 章でまとめとする．

2. SQL インジェクション（SQLI）

本章では，提案手法に係る重要な概念を説明する．

2.1 Web システムと SQLI 脆弱性

典型的な Web システムは，プレゼンテーション（ユーザインタフェース），ドメインロジック（アプリケーション処理），データ管理から成る 3 層アーキテクチャを持つ[13]（図 1）．

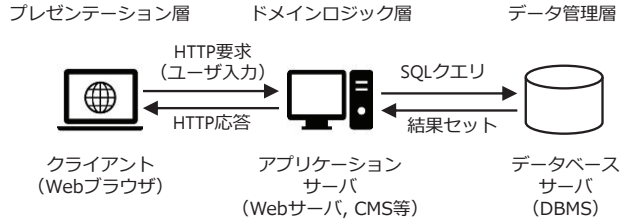

図 1: Web システムの 3 層アーキテクチャ

プレゼンテーション層は，ユーザのデバイス上の Web ブラウザ等で動作し，情報の入出力を行う．ドメインロジック層は，アプリケーションサーバ上で動作し，アプリケーションの機能を制御する．データ管理層は，DB サーバ上で動作し，DB 管理システム（DBMS）によって提供されるデータストレージとそのアクセス機能を持つ．最も普及しているデータモデル及びクエリ言語は関係型と SQL である[14]．

SQLI が起こる原因は，ドメインロジックのコード中で，適切な処理がされていない外部由来の要素を組み込んだ SQL 文を文字列として組み立てることに起因する[8,9]．このように SQL 文を文字列の連結で作成する方法を動的文字列生成または動的 SQL と呼ぶ．SQL 文への外部由来文字列の注入により，意図しない SQL 文を作成してしまう．

ユーザ認証における SQLI 脆弱性の例を図 2 に示す．図の左側はユーザインタフェース，右側(a)はソースコード 1 の 1-3 行目の処理によって生成される SQL 文である．この SQL 文は実行前に以下のように最適化される（図 2 右側）．

(a) 動的文字列生成により SELECT 文が作成される（ソースコード 1，1-3 行目）．リクエストパラメータ `username` と `password` で渡された外部入力文字列 `"a' OR 1=1; --"` と `"dummy"` が SQL 文の断片文字列と連結される．

(b) 連続ハイフン `--` は SQL 構文でコメントの開始を意味するので，それ以降の部分を無視する．

(c) 式 `1=1` の評価は常に真なので，WHERE 句の条件はトートロジにより空とみなす．

この最適化により，実行する SQL 文は(c) `"SELECT * FROM user;"` となる．これはすべてのアカウント情報を検索する．DB に一つでもアカウント情報があれば，6 行目の `if` 文の条件式は偽となり，認証が成功してしまう．

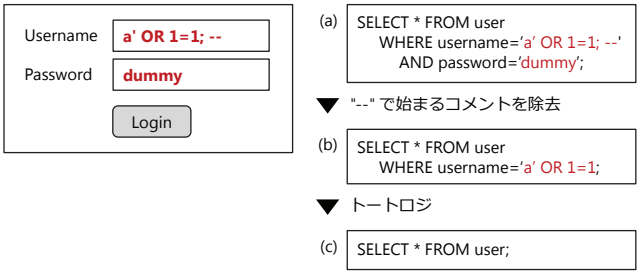

図 2: SQLI による認証回避の例

ソースコード 1: SQLI 脆弱な認証処理（PHP, MySQL）

```
1  $str_sql = "SELECT * FROM user "
2           + "WHERE username='$_GET['username']'"
3           + " AND password='$_GET['password']'";
4  $response = mysqli_query($con, $str_sql);  //動的 SQL
5  $result = mysqli_fetch_array($response);
6  if (is_null($result))
7      throw new Exception('Login Failed');
8  return TRUE; /* Login Succeeded */
```

2.2 プリペアードステートメントによる SQLI 防御

SQLI 攻撃に対する防御策の一つがプリペアードステートメント（Prepared Statement: PS）の利用である[9,15]．PS は SQL 文の構造を定義し，外部入力との組み合わせ後もその構造を変化させない働きを持つ．すなわち，想定する SQL

ⓒ2019 Information Processing Society of Japan

文の構造を変化させるような外部入力値の注入を確実に防止できる．

PS を用いてソースコード 1 を書き直したものをソースコード 2 に示す．1-3 行目の SQL 文は PS のテンプレートである．疑問符 `"?"` は，プレースホルダ（またはパラメータ）を表す．4 行目の `mysqli_prepared` 関数はテンプレートを DB サーバに送信し，DB サーバはこれをコンパイルしておく．次の `mysqli_stmt_bind_param` 関数の実行で，外部入力値がプレースホルダにバインドされる．つまり，二つの入力値 `username` と `password` が DB サーバに送信され，コンパイル済みの SQL 文テンプレートにはめ込まれ，完成した SQL 文が実行される．

プレースホルダにはリテラル値のみがバインド可能で，SQL 構文として解釈されない．例えば，前例の入力文字列 `"a' OR 1=1; --"` がプレースホルダにバインドされた場合，論理和 `OR` ともコメント開始 `--` とも解釈されず，全体で一つの文字列として解釈される．このようにコードとデータとを分離することで，PS は予期しない SQL 文の作成と実行を防止する．

ソースコード 2: SQLI 防御した認証処理（PHP, MySQL）

```
1  $str_sql = "SELECT * FROM user "
2             + "WHERE username=?"
3              + " AND password=?";
4  $ps = mysqli_prepare($con, $str_sql); //PS
5  mysqli_stmt_bind_param($ps, "ss",
6             $_GET['username'], $_GET['password']);
7  mysqli_stmt_execute($ps);
8  $response = mysqli_stmt_get_result($ps);
9  $result = mysqli_fetch_array($response);
10 if (is_null($result))
11     throw new Exception('Login Failed');
12 return TRUE; /* Login Succeeded */
```

3. 関連研究

SQLI 防御手法は，防御的コーディング，脆弱性検出，実行時防止の 3 種類に分類できる[16,17].

防御的コーディングは，セキュアコーディングとも言い，不注意により脆弱なコードを書いてしまうことを防止するためのソフトウェア開発方法である. SQLI に対する防御的コーディングには，プリペアードステートメント，ストアードプロシージャ，文字エスケープ，ホワイトリストフィルタなどの種類がある[16]. 特に最初の二つは SQLI を確実に防止できる．この手法の欠点は，見逃し等のヒューマンエラーや低スキルプログラマによるコーディング誤りを避けられないことである．

脆弱性検出は，ソースコード中の SQLI 脆弱性を検出する手法である. 静的コードテスト[18], 攻撃生成[19,20,21], テイントに基づく情報フロー解析[22,23,24,25]等の方式がある．コードテスト方式はパス数爆発によるスケーラビリティの問題がある．また，アプリケーションロジックや脆弱性パターンのモデリングの精度，複雑なコードや外部ライブラリの扱いの困難さが原因で，擬陽性（false positive）または偽陰性（false negative）を起こしやすい．

実行時防止手法は，正当な SQL 文と実行時 SQL 文とを比較して，SQLI 攻撃を防止する．正当なクエリの規定[26,27]または推論[28]，動的 SQL 文解析[29,30,31]，ホワイトリスト生成[18,31]等の方式がある. この手法の問題点は，すべての外部入力を補足することが困難なことである[16]. また，ホワイトリスト生成方式は，すべての可能性のある SQL 文をテストしなければならない．

我々の提案手法は，実行時防止の一種で，プログラミング言語の処理系レベルで外部入力の追跡を行い，すべての SQL 文は必ずプリペアードステートメントで実行される．そのため, 不正な SQL 文の誤認や見逃し（擬陽性と擬陰性）がなく，SQLI 攻撃を確実に防止する．また，プログラマの知識やスキルへの依存もなく，専用のサブシステムを追加する必要もない．

4. 透過的 SQL インジェクション防御策

本章では，我々が提案する SQLI 防御手法を示す．ドメインロジックの実装（アプリケーションコード）が SQLI 脆弱かどうかに係わらず，またプログラマが SQLI 対策を意識する必要なく, SQLI 攻撃を防止できるという意味で透過的である．言語処理系レベルで外部入力を確実に追跡し，動的文字列生成 SQL 文に注入された場合は自動的に PS に変換して実行する．以下では，特定のプログラミング言語に依存しないよう，提案手法を抽象的に記述する．次章で具体的に WordPress 上の PHP での実装方法を示す．

4.1 外部入力の追跡

まず，プログラミング言語における文字列型オブジェクト及び関連操作を拡張した抽象的な言語要素を仮定する．この言語要素を拡張文字列システム（Extended String System: ESS）と呼ぶ．ESS はドメインロジック層で動作し，以下の二つの機能を持つ．

- 外部入力マーキング: 外部入力の文字列にマーク（目印）を付ける．マークは，文字列中の外部入力部分の位置を表す．
- マーク伝搬: 文字列どうしの連結を行って新しい文字列が作られるときに，マークを伝搬させる．

4.1.1 外部入力マーキング

ESS オブジェクトは，文字列とマークの二つのデータで構成される．マークは二つの整数の組（スライス索引）のリストである．文字列は，第 1 文字目を 1 とする整数の索引を持つ．文字列 S，索引 a と b（a < b）について，スライス S[a,b]は索引 a から b - 1 の区間の部分文字列を表す．例

えば，スライス `"abcdefg"[2,4]` は `"bc"` である．

図 3 に外部入力マーキングと ESS オブジェクトの例を示す．前述の例と同じユーザ認証のシナリオで，ユーザ名とパスワードに入力された二つの文字列に対してそれぞれマーキングを行い，図右側の二つの ESS オブジェクトが生成される．上段が文字列，下段がマークを表す．この時点では文字列全体が外部入力であるため，それぞれのマークは[1,14]と[1,6]となる．

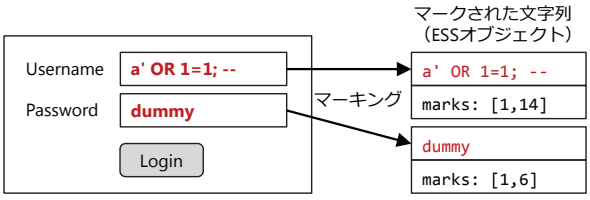

図 3: 外部入力マーキングと ESS オブジェクトの例

4.1.2 マーク伝搬

既に述べたように，SQLI は，適切な処理をされていない外部入力値と SQL 文の断片文字列とを連結して SQL 文を作成する場合に起きる．文字列どうしの連結が行われる際に，後で外部入力値がどの位置に含まれているかを判定できるように，連結後の文字列にマークを正しく伝搬させる必要がある．この機能をマーク伝搬と呼ぶ．

図 4 にマーク伝搬の例を示す．図 3 に示した二つの外部入力値を使って SQL 文が組み立てられるとする．文字列連結操作をプラス記号（`"+"`）で表す．まず，(a)で SELECT 句とユーザ名の文字列が連結される．前者は外部入力ではないのでマークを持たず，後者はマーク [1,14]を持つ．連結の結果，(b)の ESS オブジェクトが作成され，マークは[36,49]となる．すなわち，索引 36 から 48 の部分文字列が外部由来であることを表す．次に，もう一つの外部入力であるパスワードの文字列が連結され，最終的に(c)の ESS オブジェクトが作られる．二つの外部入力の位置を示すマーク [36,49], [65,70]を持つ．

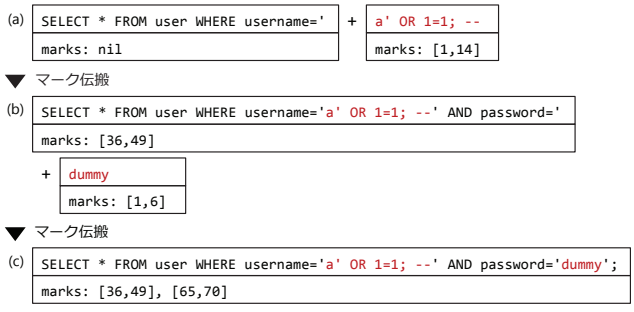

図 4: 文字列連結とマーク伝搬の例

4.2 SQLI 防御手順

ESS を用いて SQLI 攻撃を防止する手順を示す（図 5）．

Web アプリケーションインタフェース（Web IF）はクライアントからの HTTP リクエストを受信し，それをドメインロジックに引き渡す．ドメインロジックの中で SQL 文を生成する部分をここでは SQL 生成ロジックと呼ぶ．SQL 文は DB ドライバを通じて DB サーバに送られて実行される．

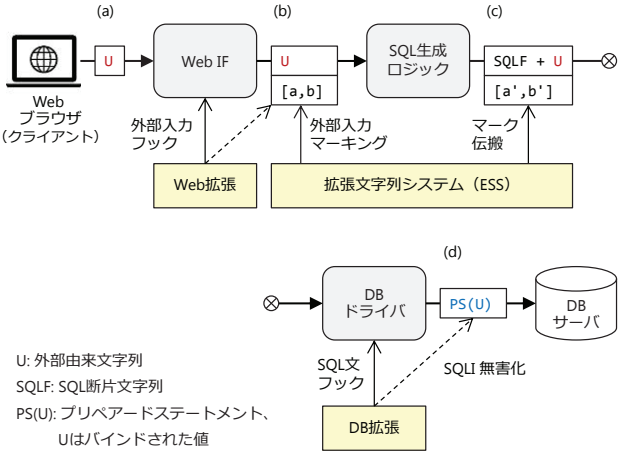

図 5: SQLI 防御手順

システムの本来の処理に割り込むために，二つの拡張機能を用意する．Web 拡張は，Web IF を拡張して外部入力をフックする．DB 拡張は，DB ドライバを拡張して SQL 文の送信をフックする．

以下に SQLI 防御手順を示す（図 5 参照）．

(a) 外部入力フック: Web IF モジュールは，外部入力を受信したら，値をパースして文字列オブジェクトを生成する．Web 拡張は，ESS の外部入力マーキング機能を呼び出す．

(b) 外部入力マーキング: ESS は各外部入力値に対してマーキングを行い，ESS オブジェクトを生成する．

(c) マーク伝搬: 文字列の連結操作により新しい文字列が生成されるとき，ESS はマークを統合して更新する．

(d) SQLI 無害化（detoxification）: DB 拡張は，SQL 文が DB ドライバに渡されるのをフックして，文字列がマークされているならば PS を生成してパラメータ値（外部入力値）をバインドし，SQL 文を実行する．SQL 文の文字列がマークされていないならば，何もせず元のアプリケーションコードに戻る．

SQLI 無害化の手順を図 6 に示す．SQL 生成ロジックにより SQL 文 S が作成されたと仮定する．前述の例と同じく，不適切なユーザ名とパスワードが入力され，S の一部に注入されている．これら外部入力の位置は ESS オブジェクト内に[36,49], [65,70]とマークされている．

(d1) パラメータ化: ESS は外部入力スライスをプレースホルダ（`"?"`）に置換する．結果のプリペアー

ドステートメントを PS とする．

(d2) パラメータバインド: DB 拡張は，DB ドライバが提供する関数を呼び出して PS を DB サーバに送りコンパイルする．同じく DB ドライバが提供する関数を用いて各外部入力スライスを PS にバインドする．

(d3) 実行: DB 拡張は，DB ドライバが提供する SQL 実行関数を呼び出して，パラメータをバインドした PS を実行する．

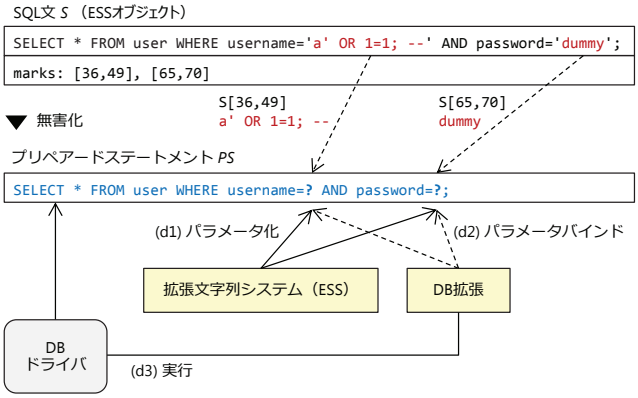

図 6: SQLI 無害化手順

5. WordPress での実装

本章では，前章で示した提案手法を WordPress でどのように実装するか述べる．

5.1 PHP 処理系への SQLI 拡張モジュールのインストール

PHP は言語拡張のしくみを元来持っており，拡張機能を実装したコードを拡張モジュールと呼ぶ[32]．拡張モジュールは C 言語で実装する．前章で述べた ESS，Web 拡張，DB 拡張を PHP 用に実装したものをここでは SQLI 拡張モジュールと呼ぶ．

まず，WordPress またはその他の PHP 処理系に以下の手順で SQLI 拡張モジュールをインストールする．

(1) ソースコードをコンパイルして，ダイナミックリンクライブラリ（.so ファイル）を作成する．

(2) PHP の開始時に拡張モジュールをロードするように，実行時設定ファイル php.ini の extension オプションに(1)のダイナミックリンクライブラリを指定する[33]．

5.2 SQLI 拡張モジュールのライフサイクルと処理内容

SQLI 拡張モジュールの処理内容と実行のタイミングを説明する．PHP の拡張モジュールは，各 HTTP リクエストについて 5 相から成るライフサイクルで動作する[32]．図 7 の左上にこの 5 相を示す．第 3 相以外の各相では，拡張モジュール内に定義された特定の関数を呼び出すことが決められている．

SQLI 拡張モジュールの処理内容（図 7 右側）をライフサイクルに沿って示す．

(1) Module startup: PHP の関数テーブルを書き換えて，PHP 本来の処理をフックするよう設定する．文字列連結オペレータと，文字列の SQL 文を引数とする SQL 実行関数の二つのフックを設定する（ここでは MySQL または Maria DB の利用を仮定する）．

(2) Request startup: SQLI 防御手順で述べた外部入力フックの役割を果たす（図 5）．SQLI 拡張モジュール内の Web 拡張の実装を呼び出して，クライアントから受信した HTTP リクエストのすべての外部入力をマークする．すなわち，GET/POST のパラメータ及び Cookie の値をマークする．

(3) PHP's life: PHP のアプリケーションコード内で文字列連結オペレータまたは SQL 実行関数が呼び出されると，それぞれ(1)で設定した関数にフックされる．文字列連結では，マーク伝搬の処理が実行される．SQL 文の実行では，SQLI 無害化手順（図 6）に従い，SQL 文のパラメータ化による PS の作成，外部入力スライスの取得，パラメータバインド，PS の実行，最後に結果の取得を行う．

(4) Request shutdown: 何もしない．

(5) PHP shutdown: 何もしない．

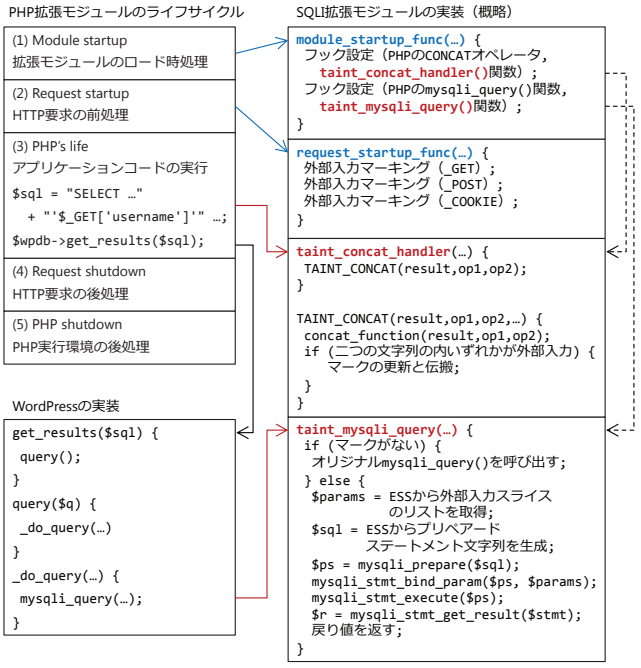

図 7: SQLI 拡張モジュールのライフサイクルと処理内容

ESS オブジェクトは HTTP のリクエストコンテキスト内に HashTable オブジェクトの要素として保持する．キーは文字列，値はマークを格納する 2 次元配列である．

6. 評価

本章では，提案手法及び実装の評価について述べる．

6.1 正確性

まず，提案手法が SQLI 攻撃を防御できるかどうかを評価した．SQLI 脆弱なプラグイン "Server Status by Hostname/IP"（バージョン 4.6）[34]を導入した WordPress 実行環境を構築し，種々の SQLI 攻撃を実際に実行して，防御できるかどうかを確認する．本評価では，SQLI 脆弱性の検出及び攻撃のプロセスを自動実行するペネトレーションツール sqlmap[35]を使用した．sqlmap が検出できる SQLI の種類は表 1 に示す 6 種類である．

Sqlmap で評価した結果，SQLI 脆弱な実行環境では sqlmap が数個の SQLI 脆弱性を検出したが，提案手法を組み込んだテスト環境では SQLI 脆弱性は検出されなかった．すなわち，提案手法により SQL 攻撃を防御できたといえる．

表 1: 正確性の評価に用いた SQLI 攻撃の種類

1	Boolean-based blind SQLI
2	Error-based queries SQLI
3	Inline queries SQLI
4	Stacked queries SQLI
5	Time-based blind SQLI
6	UNION query SQLI

6.2 性能

提案手法の導入に伴う性能の変化を調べるために， 表 2 に示す実行環境を構築した．測定用のクライアントプログラム， WordPress が動作する Web サーバ，DB サーバは同一ホスト上で動作する．

表 2: 評価環境

Platform	Google Compute Engine, n1-standard-1 CPU: Intel Xeon 2.20GHz 1Core, RAM: 3.75GB
OS	Debian GNU/Linux 9, kernel 4.9.0-9-amd64
WordPress	WordPress 5.3 ja, on PHP-FPM (FastCGI)
PHP	PHP 7.0.33
Web server	Nginx 1.10.3
DBMS	Maria DB 10.1.38

6.2.1 応答時間

提案手法の導入に伴う応答時間の変化を調べた．ここで応答時間は，クライアントが HTTP リクエストを送信してからレスポンスを受信するまでの時間である．10,000 回測定して平均を求めた結果を表 3 に示す．

提案手法の導入により，応答時間が約 14%増大した．しかしながら，絶対値は約 4ms と小さく，人が利用する対話的なシステムでは問題とならない．

表 3: 応答時間 (ms)

対策なし	29.3	
提案手法	33.5	
差	4.2	(14.3%)

6.2.2 メモリ使用量

同様に，メモリ使用量を計測した結果を表 4 に示す．ここで計測したメモリ使用量は，WordPress プロセスの実メモリサイズ（Resident Set Size: RSS）と仮想メモリサイズ（Virtual Set Size: VSZ）である．それぞれについて，システム起動後でクライアントのリクエスト前（添字 i の項目）と，1,000 回のリクエストを処理した後（添字 r の項目）で計測した．

いずれの値も，提案手法の導入による増加は約 0〜2.4K バイト，約 0〜3.86%に留まっている．この増加分は，ESS オブジェクト及び拡張モジュールのコードである．この結果から，標準的な WordPress 実行環境におけるメモリ使用量の負荷は無視可能な程度に小さいと言える．

表 4: メモリ使用量 (Kbytes)

	RSSi	RSSr	VSZi	VSZr
対策なし	6.78	32.41	248.85	255.79
提案手法	6.77	33.66	250.94	258.20
差	-0.01	1.25	2.09	2.41
	(-0.15%)	(3.86%)	(0.84%)	(0.94%)

7. おわりに

本論文では，Web サイトの構築と運用に広く利用されている WordPress を対象として，SQLI 攻撃に対する防御策を示した．提案手法は，WordPress の実装言語である PHP の言語処理系を拡張して，実行時に危険な SQL 文を無害化する．SQL 文に注入される外部入力を確実に追跡し，必要に応じて SQL 文の実行方法を変更することで，プラグイン等のアプリケーションコードが SQLI 脆弱な場合であっても DB アクセスを安全に実行できる．また，性能評価の結果，本手法による性能劣化は小さく，対話的な Web ページ操作では無視可能な程度である．本手法の実装コードは，わずかな修正により，PHP で実装された WordPress 以外のシステムにも適用できる．なお，実装コードは本論文発表後にオープンソースで公開する予定である．

ⓒ2019 Information Processing Society of Japan

参考文献

[1] WordPress. https://wordpress.org/
[2] W3Techs: Usage of content management systems. https://w3techs.com/technologies/overview/content_management
[3] Joomla!. https://www.joomla.org/
[4] Drupal. https://www.drupal.org/
[5] W3Techs: Usage of server-side programming languages for websites. https://w3techs.com/technologies/overview/programming_language
[6] CMS 2.0, *Web Designing*, vol.195, マイナビ出版, 2019.
[7] Movable Type. https://movabletype.com/
[8] MITRE Corporation: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). https://cwe.mitre.org/data/definitions/89.html
[9] Clarke-Salt, J.: *SQL Injection Attacks and Defense* (*2nd ed*.), Syngress, 2009.
[10] OWASP Top 10, 2017. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
[11] MITRE Corporation: 2019 CWE Top 25 - Most Dangerous Software Errors. https://cwe.mitre.org/top25/archive/2019/2019_cwe_ top25.html
[12] NIST: National Vulnerability Database (NVD). https://nvd.nist.gov/
[13] Eckerson, W. W.: Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications, *Open Information Systems*, Vol.10, No.1, 1995, pp.1–20.
[14] Date, C. J., Darwen, H.: *A Guide to SQL Standard* (*4th ed*.), Addison Wesley, 1996.
[15] IPA: 安全なウェブサイトの作り方, 改訂第 7 版, 2015.
[16] Shar, L. K., Tan, H. B. K.: Defeating SQL Injection, *IEEE Computer*, Vol.46, No.3, 2013, pp.69-77.
[17] Steiner, S., et al.: A Structured Analysis of SQL Injection Runtime Mitigation Techniques, *Proc. of the 50th Hawaii International Conference on System Sciences* (*HICSS-50*), HI, USA, 2017, pp. 2887-2895.
[18] Shin, Y., et al.: SQLUnitGen: Test Case Generation for SQL Injection Detection, *TR-2006-21*, 2006, North Carolina State University.
[19] Alshahwan, N., Harman, M.: Automated Web Application Testing Using Search Based Software Engineering, *Proc. of the 26th Int'l Conf. on Automated Software Engineering* (*ASE*), KS, USA, 2011, pp.3-12.
[20] Fu, X., Li, C.-C.: A String Constraint Solver for Detecting Web Application Vulnerability, *Proc. of the 22nd Int'l Conf. on Software Engineering and Knowledge Engineering* (*SEKE*), CA, USA, 2010, pp.535-542.
[21] Kieyzun, A., et al.: Automatic Creation of SQL Injection and Cross-Site Scripting Attacks, *Proc. of the 31st Int'l Conf. on Software Engineering* (*ICSE*), BC, Canada, 2009, pp.199-209.
[22] Livshits, V. B., Lam, M. S.: Finding Security Vulnerabilities in Java Applications with Static Analysis, *Proc. of the 14th USENIX Security Symposium*, MD, Canada, 2005, pp.271-286.
[23] Pietraszek, T., Berghe, C. V.: Defending Against Injection Attacks Through Context-Sensitive String Evaluation, *Proc. of the Int'l Workshop on Recent Advances in Intrusion Detection* (*RAID*), LNCS 3858, Springer, WA, USA, 2005, pp.124-145.
[24] Wassermann, G., Su, Z.: Sound and Precise Analysis of Web Applications for Injection Vulnerabilities, *Proc. of the 28th SIGPLAN Conf. on Programming Language Design and Implementation* (*PLDI'07*), CA, USA, 2007, pp.32-41.
[25] Xie, Y., Aiken, A.: Static Detection of Security Vulnerabilities in Scripting Languages, *Proc. of the 15th USENIX Security Symposium*, BC, Canada, 2006, pp.179-192.
[26] Huang, Y.-W., et al.: Securing Web Application Code by Static Analysis and Runtime Protection, *Proc. of the 13th International Conference on World Wide Web* (*WWW'04*), NY, USA, 2004, pp.40–52.
[27] Kemalis, K., Tzouramanis, T.: SQL-IDS: A Specification-based Approach for SQL-injection Detection, *Proc. of the Symposium on Applied Computing* (*SAC'08*), 2008, Ceara, Brazil, 2008, pp.2153–2158.
[28] Halfond, W. G. J., Orso, A.: Combining Static Analysis and Runtime Monitoring to Counter SQL-Injection Attacks, *Proc. of the 3rd Int'l Workshop on Dynamic Analysis* (*WODA'05*), MO, USA, 2005, pp.1–7.
[29] Buehrer, G., et al.: Using Parse Tree Validation to Prevent SQL Injection Attacks, *Proc. of the 5th Int'l Workshop on Software Engineering and Middleware* (*SEM'05*), Lisbon, Portugal, 2005, pp.106-113.
[30] Halfond, W. G. J., et al.: WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation, *IEEE Trans. on Software Engineering*, Vol.34, No.1, 2008, pp.65-81.
[31] Su, Z., Wassermann, G.: The Essence of Command Injection Attacks in Web Applications, *Conf. Record of the 33rd SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (*POPL'06*), SC, USA, 2006, pp.372-382.
[32] PHP Wiki: Extensions. https://wiki.php.net/internals/extensions
[33] PHP Manual: Description of core php.ini directives. https://www.php.net/manual/en/ini.core.php
[34] NVD: CVE-2019-12570. https://nvd.nist.gov/vuln/detail/CVE-2019-12570
[35] sqlmap. http://sqlmap.org/

ⓒ2019 Information Processing Society of Japan