Web Application Security Forum

2005 年 9 月 23 日

製品評価分科会
Working Draft

SQL Injection Defense Tree (revision 1.0)

エディタ：門林雄基 (奈良先端科学技術大学院大学)
コントリビュータ：賀戸大輔 (日本 IBM), 澤崎裕喜 (TechMatrix),
高木浩光 (産業技術総合研究所), 中村穣 (McAfee),
早津優美 (三井物産セキュアディレクション), 藤本真樹 (Techstyle),
藤原礼征 (Solution crew), 松尾竹純 (日商エレクトロニクス),
政本憲蔵 (Macnica Networks)

1 はじめに

1.1.本ドキュメントのねらい

本ドキュメントでは、設計から運用にいたるプロセスのなかで、SQL Injection 対策として活用可能な開発手法や、製品、運用手法などのマップを作成することを狙いとしています。これにより SQL Injection 対策として今とりうるアクションが何であるかを分かりやすく提示することを目的としています。

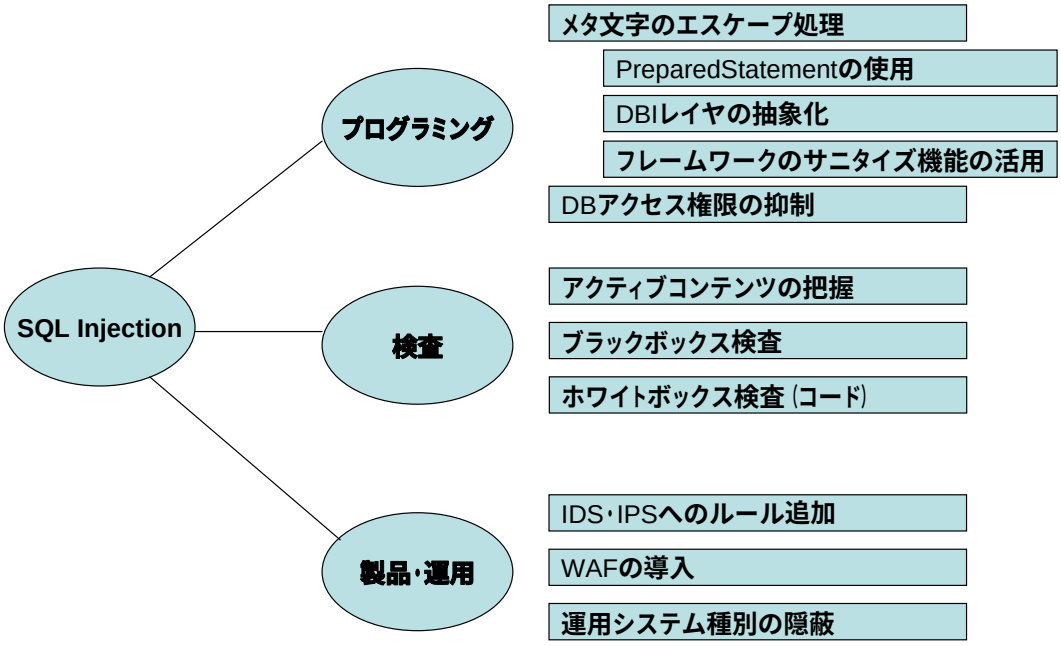
1.2.SQL Injection とは

SQL Injection とは、データベースの情報を窃取、改竄されたり、アクセス制御機能を回避されるなどの被害をもたらしうる、Web アプリケーションの脆弱性のひとつです。プログラム中の SQL 文に対して外部から不正な SQL 命令を注入して攻撃が行われることから、このように呼ばれます。

SQL Injection 攻撃は手法としては古くから知られていましたが、最近になって SQL Injection に特化した攻撃ツールが出回り始めたため、Web アプリケーション開発・運用の現場における対策の重要度が大きく増しています。

1.3.SQL Injection Defense Tree

Defense Tree（防御木）は Attack Tree（攻撃木）の逆をさす言葉で、WASF 製品評価分科会にて発案されました。攻撃木がある目的を達成するための攻撃手法を系統的に示しているのにたいし、防御木ではある脆弱性にたいしてとりうる対策手法を系統的に示すことを目的とします。次節以降ではこの防御木に沿って対策手法を説明します。



2. プログラミングでの対策

一般論として、コードレビューを行う際に、セキュリティ上の問題がないかのチェックもあわせて行うことが理想的です。また、エラー処理において格段の注意が求められることも念頭におくべきでしょう。以下では、特に SQL Injection においてとりうる対策を示します。

2.1. メタ文字のエスケープ処理

SQL Injection ではデータ中に不正にメタ文字を挿入することで、本来データとして解釈すべき部分がデータベースによって命令として解釈されてしまうという点が問題となります。このため一般的には、入力されたデータ中のメタ文字それぞれについて正しくエスケープ処理をおこない、命令として解釈されないよう処理する必要があります。実装言語やフレームワークによってとりうる対策が異なりますが、以下では主なものを列挙します。

2.1.1. PreparedStatement の使用

Web アプリケーションの実装言語とデータベースがバインドメカニズムに対応している場合にはこれを用いるとよいでしょう。Java 言語では PreparedStatement として提供されています。バインドメカニズムを用いると、パラメータに「'」などが含まれていても、特殊文字ではなく単なる文字として扱われます。その結果、不正な SQL の挿入を防止できます。

ただし、バインドメカニズムを用いた場合でも、Like 句における「%」(任意の数の任意の文字にマッチ)と「_」(1 つの任意の文字にマッチ)に関しては注意が必要です。これらの文字は、バインドメカニズムを用いていても Like 句において特殊文字として扱われます。したがって、このような場合にはエスケープ処理が必要となります。なお、特殊文字はデータベース製品によって異なるので、対象データベース製品における特殊文字の定義を確かめる必要があります。

2.1.2. DBI レイヤの抽象化

Web アプリケーションの実装言語として PHP, Perl などを用いている場合には、SQL 命令文を文字列をつなぎ合わせて DBI レイヤに渡すという実装が一般的です。しかしこのような実装では SQL Injection が生じやすいため注意が必要です。より進んだ対策手法として、DBI レイヤを抽象化したフレームワークを用いることも考えられるでしょう。このようなフレームワークでは SQL 命令文と SQL 命令文中のデータをはっきり区別することができ、SQL Injection が生じないよう工夫を凝らしているものがあります。さらには O/R(Object/Relational)マッピングを利用することで SQL 文の記述自体を抑制し、開発効率とセキュリティの両側面で改善をはかることも可能です。

2.1.3. フレームワークのサニタイズ機能の活用

Web アプリケーション・フレームワークの中にはメタ文字がデータ中に挿入された場合にこれを無効化する機能を備えるものがあります。このようなフレームワークの機能を用いて、データ中に不正にメタ文字を挿入された場合においても SQL Injection 攻撃が有効にならないよう工夫することもできるでしょう。ただし、既存のフレームワークのサニタイズ機能は本来 SQL Injection 等の攻撃を防ぐためにつくられたものではないので、対策手法としての有効性を確保するためには、あらかじめ使い手が限界を知っておく必要があります。

また一般論として、コーディング・スタイルにおいて脆弱性を持ち込まないためのスタイルを定義し、運用する必要があるでしょう。例えば、変数名に安全でないデータであることを識別できるプレフィックスをつけるなどの対策は必須だと考えられます。

2.2. DB アクセス権限の抑制

ある種の SQL Injection 攻撃では、UNION などを用いて別のテーブルの情報などを引き

出そうとする場合があります。このような攻撃を未然に防ぐために、Web アプリケーションからデータベースに接続するユーザの権限は最小限のものにする方法があります。なお、これは万が一のための安全対策であり、2.1 節で述べたような対策によって SQL Injection が完全に排除されている場合は必要ないといえます。しかし現実的には、Web アプリケーション中でのメタ文字チェックに漏れがあっても事故がおきることのないよう安全サイドに倒した設定が望ましいでしょう。

Web アプリケーションによってはコネクションプーリングなどの技法により、1 つのユーザでのコネクションを使いまわしていることがあります。このような場合には、アプリケーション全体としてどのような権限が必要かを検討する必要があります。

3. 検査での対策

まず一般論として、運用に入るまえに、セキュリティを中心としたテストを実施することが望ましいでしょう。従来のシステム開発では単体テスト、統合テストといったテスト手法が代表的ですが、それらに加えて、セキュリティテストを行うことが望まれます。

3.1. アクティブコンテンツの把握

まず自サイト内でどのような Web アプリケーションが動作しているのかを把握する必要があります。このために後述するブラックボックス検査が活用できる場合や、Web サーチエンジンが活用できる場合があります。

また、アクセス権限を明確化しておく必要があります。具体的には、1) それぞれの Web アプリケーションがどのようなユーザ権限でアクセスされるのか、また 2) 個々の Web アプリケーションがデータベースなど周辺のシステムに対してどのようなアクセス権限をもつのか、という 2 点です。

3.2. ブラックボックス検査

ソースコードを公開せずに検査を行う方法です。ブラックボックステストでは、Web アプリケーションに対して、あらかじめ想定される不正リクエストを送信し、 アプリケーションがどのような応答をするかによって脆弱かどうかを判定します。

実際に検査を実施する場合には、SQL エラーを誘発させるデータを送信し、その応答で判断します。この結果、少なくとも SQL エラーが発生した場合は、脆弱性があると考えられます。

このように、SQL インジェクションの脆弱性があるかどうかの判定については SQL とプログラムについての詳しい知識が必要になります。また、SQL Injection 攻撃の脆弱性を検査する場合、検査ターゲットを的確に把握することが重要です。

検査ターゲットとして、DB と直接連携しているアクティブコンテンツを把握するだけでなく、間接的に DB 上で処理される値が指定されるパラメータや cookie 等を把握するこ

とも必要です。アプリケーションによっては、HTTP Header の値も DB の検索条件に利用している場合があります。

テスト手法としては、手動による検査と Web アプリケーション専用のテストツールを利用した自動検査があります。

手動による検査では、検査パターンを柔軟に作成することが出来るので、SQL エラーを誘発させるテストだけでなく、よりリアルなテストを実施することが出来ますが、

・アプリケーションの動作から、検査対象の機能をどれだけ把握できるか

・不正リクエストおよび不正データのパターンをどれだけ知っているか

といった点で、検査員の技術レベルに依存することが懸念されます。また、テスト対象やテスト項目が多い場合、非常に手間がかかり、多くの工数を費やしてしまうことも懸念されます。

テストツールを利用した自動検査では、最初にサイトを巡回することで、そのサイトで発生する正当な HTTP リクエストを把握し、収集した HTTP リクエストをベースに、その中に含まれる HTTP Header, cookie,パラメータ等の値を網羅的に不正データに変化させてテストパターンを自動生成します。

テストツールの利用は、一定のレベルで短期間に網羅的に検査結果を得られるといった利点がありますが、HTTP Header, cookie,パラメータ等複数の要素を組み合わせてはじめて効果のある攻撃については、テストパターンとして自動生成できないことが懸念されます。

手動、自動、それぞれの欠点を補うには、Web アプリケーション専用のテストツールにより、短期間で網羅的にテストを行い、テストツールで収集した情報やテスト結果を元に、手動による複雑なテストやよりリアルなテストを行うといった 2 段階の検査が、脆弱性を見つけ出すのにより効果的な検査手法といえます。

なお e-commerce サイトでは複数のページにまたがる複雑なページ遷移をおこなう場合があるため、検査サービスの専門家と Web アプリケーションの設計者、開発者に参画を依頼し、アプリケーションロジックを見極めてテストする必要があるでしょう。

3.3. ホワイトボックス検査

アプリケーションのソースコードを開示して検査を行う方法です。入力に対する挙動を確実に追えるので、正確かつ効果的な検査を行うことができます。専門知識を有する検査員によるアプリケーション検査の対象となるユーザ入力を受けうるすべての箇所（認証や検索などのフォーム、アドレスバーからの入力がその代表例です）を、ユーザインタフェースの動きとソースコードの両方で確認することにより、個々のアプリケーションの目的に合わせたソースコードの安全性向上が容易になります。

4. 製品・運用での対策

一般論として、Web アプリケーションに対してもセキュリティマネジメントの基本的な

方法論があてはまることはいうまでもありません。それらに加えて、SQL Injection に対してとりうる対策を以下に示します。

4.1. IDS・IPS へのルール追加

IDS/IPS での防御を考えると基本的には用意されているシグネチャの活用や、無い場合には独自にシグネチャを追加することである程度の検知が可能となります。

より正確なシグネチャを作成する場合にはアプリケーションの作りや使用している DB の種類、PHP などの言語特性を意識する必要はありますが、基本的には SQL Injection で使用されることの多いキーワードが HTTP や HTTPS 上でリクエストとして発行された時に検知するようなシグネチャを作成します。これらのキーワードは固定位置ではなく間にスペース、カンマ、OR などが挿入もしくは併用されることも考慮して作成する必要があります。

キーワードの例：

- UNION
- INSERT
- DROP
- SHUTDOWN
- EXEC
- XP_
- SP_
- CREATE TABLE

また、SQL コマンド以外にも、'OR　1=1'のような条件を追加するものをキーワードに含めることも検討できますが、OR 'ABCD' = 'AB' + 'CD' など限りがなく全てをシグネチャで列挙するのは困難であり、全てを対称にするのは現実的でないため対象は限定する事になります。

製品の機能／特性に依存する部分もありますがサーバ上で実行されるコマンドやコードのチェック、環境によっては極端に長いＵＲＬなどを監視することも有効な手段となります。

ネットワーク上での検知／防御である事を考えると重要データのやり取りに使用される傾向が高い HTTPS 通信上での検知は、SSL の復号機能を持つものでなければ対応はできません。

4.2. Web Application Firewall の導入

Web Application Firewall(以下、WAF)は Web Application に特化した Firewall で、レイヤ７まで確認しアクセスコントロールを実施出来る点が通常の Firewall と異なります。

その為、WAF は SQL Injection 攻撃と思われる HTTP リクエストを Web Server の手前で止める事が可能です。

アプリケーションの開発者がいなくなった為完全なチェックが難しい、チェック項目が膨大すぎてとても全てを確認しきれない、本番環境しか無い為全てのチェックが出来ない、アプリケーションの作り直しが困難であると言う様な場合に WAF の導入は有効であると言えるでしょう。

ただし WAF はすべての脆弱性に対し有効であるとは限らないので、盲目的に製品を導入するだけでは危険です。Web アプリケーションの実装方式として、一般に「使うべきでない」とされているような脆弱な実装方式をとっているケースでは WAF を導入しても脆弱性をカバーできないことがあります。検査ツール・検査サービスと併用するなどして有効性を確保する必要があるでしょう。

また一般論として、ネットワークの設計に関して、アプリケーションの開発者が参加することにより、アプリケーションの機能とネットワーク設計の間に不整合が発生していないかチェックする必要があるでしょう。

4.3. 運用システム種別の隠蔽

攻撃者は Web サーバのバージョンや Web アプリケーションの実装言語、データベースの種別をもとに攻撃手法を変えてくることがあります。このための手がかりとして、レスポンスヘッダやエラーページをはじめ、求人サイトなどに掲載されている情報が用いられるようです。

このような手がかりとなる情報を公表しないよう、システムの設定や運用手順を点検したほうがよいでしょう。エラーページは Web サーバと Web アプリケーションが生成するので、Web サーバと Web アプリケーションコンテナの設定、ならびに Web アプリケーション中の例外処理部のそれぞれにおいて対策が必要です。

例えば Web サーバの情報を隠蔽したい場合、具体的には、Apache の場合、httpd.conf に下記のような指示子を追加することで、レスポンスヘッダやエラーページ内における Apache バージョンの表示を防ぐことができます。

```
ServerTokens ProductOnly
ServerSignature Off
```

またプログラムや設定での対策だけでなく、ネットワーク機器レベルでの対策が求められる場合もあります。nmap のフィンガープリントオプションを使うと、レイヤ 4 以下の挙動を見てターゲットの OS を割り出すことができますが、設定だけでこれらの挙動を完全に隠蔽するのは困難です。サーバ負荷分散装置や Web Application Firewall などの、リバースプロキシ型のサーバを手前に配置することで、全レイヤにて Web サーバの OS を隠蔽することが可能な場合があります。

5. 本ドラフトについて

本ドラフトは Web Application Security Forum 製品評価分科会のメンバーによって作成されました。当フォーラムについてのお問い合わせは、フォーラム事務局までお願いします。

Web Application Security Forum 事務局
(日商エレクトロニクス株式会社内)
TEL 03-3544-8333　FAX 03-3544-7642
E-mail: info-wasf@wasf.net
URL: http://www.wasf.net/