SQL INJECTION: ENTENDENDO E EVITANDO

MAGALHÃES, Felipe. B. [1], BASTOS, Rafael. R[2]

[1] Faculdade IDEAU – Bagé – RS – Brasil – magalhaesbg@gmail.com
[2] Faculdade IDEAU – Bagé – RS – Brasil – rafaelrodriguesbastos@gmail.com

RESUMO

Este artigo aborda o conceito e métodos de evitar o SQL Injection, que é um ataque que pode ocorrer em sistemas locais ou web. Serão abordadas três maneiras de evitar a injeção SQL, sendo elas, a criptografia dos dados, a validação dos dados via replace e a utilização de PHP PDO.

Palavras-chave: SQL Injection, criptografia de dados, validação dos dados via replace, PHP PDO.

1 INTRODUÇÃO

Nos dias atuais, os sistemas computacionais são frequentemente alvos de ataques não autorizados. Segundo Cristina Deluca (2018), os ataques com maior incidência são em servidores web por meio de inserção e manipulação de scripts e também inserção de código no banco de dados, também conhecido como *SQL injection*. Para Clarke (2012), *SQL injection* é uma vulnerabilidade que permite ao invasor alterar as consultas SQL que a aplicação envia ao banco de dados. Quando o invasor pode influenciar as sentenças que serão executadas no banco de dados ele pode fazer uso dos recursos do SQL propriamente dito.

Segundo Macêdo (2016), *SQL injection* é uma das vulnerabilidades mais prejudiciais, pois ela pode ser utilizada para ler, atualizar, adicionar ou eliminar informações de qualquer tipo contidas no banco de dados.

Para Souza (2013), existem várias formas de evitar os ataques de *SQL injection,* dentre as quais destacam-se criptografia, filtro de entradas e utilização do *PHP Data Objects* (PDO).

De acordo com Clarke (2009) ainda afirma que todas as informações consideradas de risco devem ser criptografadas na hora do armazenamento e na sua transmissão, a fim de evitar que os dados se mantenham em texto puro. Outra forma de prevenir o ataque é criando uma validação dos dados recebidos pela aplicação. Existem dois tipos de validação: *whitelist* e a *blacklist*. *Whitelist* é o processo de entrada que testa os dados recebidos pela aplicação com um padrão definido. Essa validação permite apenas a entrada dos dados que correspondam a

padrões pré-estabelecidos, como valor a partir de uma lista pré-definida, tipo de dado e tamanho dos dados. Já a *blacklist* é a prática de rejeitar caracteres especificados. Os dados recebidos são verificados e caso haja um caractere que se encontre na *blacklist* esses dados são invalidados, ou esses caracteres são removidos.

Segundo Garret (2012), criptografia é uma técnica a qual transforma uma informação legível em um conjunto de caracteres incompreensíveis. A criptografia protege os dados, pois em caso de interceptação dos dados enviados em um acesso, serão vistos apenas caracteres aleatórios, sendo necessário uma chave para decifrá-los.

A criptografia dos dados deve ser empregada sempre que possível para dificultar a visualização não autorizada. Na figura 1 apresenta-se um exemplo de criptografia de dados recebidos por um *form* em HTML utilizando do método de submissão POST e sendo criptografado pelo servidor PHP.

A criptografia sozinha não é o método muito eficaz para evitar ataques, devendo ser realizado também o emprego de filtros nos dados recebidos pelos formulários. Assim, podem ser removidos partes de código malicioso ou indevido, que eventualmente sejam inseridos juntamente com os dados dos formulários.

```
<html>
    <input type="password" name="senha">
</html>
<?php
    $senha=md5($_POST['senha']);
?>
```

Figura 1 - Exemplo de criptografia dos dados recebidos

A figura 2 mostra um exemplo de filtro para remoção de caracteres indesejáveis no conteúdo da variável *$senha*.

```
<?php
    $senha = "Um teste de or '1='1;";
    $resultado = preg_replace('/[^[:alnum:] ]/', '',$senha);
?>
```

Figura 2 - Exemplo de substituição de caracteres indesejáveis

O terceiro método abordado é a utilização do PDO. Segundo Rocha (2008), PDO é uma extensão do PHP a qual fornece uma interface segura, leve e rápida e permite a manipulação e conexão ao banco de dados. O PDO também tem métodos de controle de transações utilizando o *commit* e o *rollback*. Um ponto forte do PDO que amplia a velocidade de produção é o fato de não trabalhar com erros e sim com

exceções. O PDO faz uso de parâmetros para passar os dados recebidos fazendo assim que os dados sejam tratados e adicionados no local apropriado na sentença SQL, sendo assim, evitando SQL maliciosos. A figura 3 mostra um exemplo de código PHP com uso do PDO.

```php
<?php
    $pdo = new PDO('mysql:host=localhost;dbname=crud', 'root', '');
    $stmt = $pdo->prepare('select * from agenda where nome = :nome');
    $stmt->bindValue(':nome', 'kalil');
    $run = $stmt->execute();
?>
```

Figura 3 - Consulta em banco de dados com PDO

2 METODOLOGIA

Para realização do presente trabalho, inicialmente foram investigadas formas de evitar ataques de *SQL injection.* Após foram elaborados três cenários de sistemas para web com formulários para submissão de dados.

Para a elaboração das páginas dos formulários utilizou-se a linguagem de marcação HTML versão 5. Para o processamento das requisições, foi empregada a linguagem de programação PHP versão 7.2.6. O servidor web utilizado foi o servidor Apache versão 2.4.33, em ambiente linux com sistema operacional Ubuntu 18.04. Os testes foram realizados em 5 computadores, todos utilizando sistema operacional Windows 10, com uso do navegador Google Chrome versão 69.

Inicialmente foram elaborados formulários para submissão de dados, nos quais foram implementados mecanismo de criptografia, filtro de validação e PDO para proteção ao *SQL injection* e foram realizados três testes. O primeiro teste foi direcionado para evitar um ataque de injeção SQL usando a técnica de criptografia dos dados de entrada. O segundo teste foi direcionado para evitar a injeção SQL, usando a técnica de filtro de validação dos dados. O terceiro teste foi direcionado para evitar a injeção SQL, usando o PDO.

3 RESULTADOS E DISCUSSÃO

Para o primeiro teste foi utilizado o mecanismo de criptografia para dados enviados pelo usuário. A variável $senha receberá os dados enviados via formulário após ser criptografado com o método md5.

```php
<?php
    $senha=md5($_POST['senha']);
    $sql = "SELECT * FROM usuario WHERE senha = '$senha'";
?>
```

Figura 4 – Criptografando o campo senha.

Como pode ser visto na Figura 4, independente do valor enviado via método $_POST, a variável $senha receberá o valor criptografado, assim, possíveis caracteres maliciosos serão removidos em razão do emprego da função md5. O md5 é um método de criptografia que gera um *hash* de 32 caracteres independente do texto criptografado.

Para o segundo teste foi utilizado o método de filtragem de validação para os dados enviados pelo usuário. A variável $senha receberá os dados enviados via formulário após passar pelo filtro de validação. O método que consiste em validar todos os dados com replace pode ser visto na figura 5.

```php
<?php
    $senha = "Um teste de or '1='1;";
    $resultado = preg_replace('/[^[:alnum:] ]/', '',$senha);
    echo $resultado;

    //$resultado = Um teste de or 11
?>
```

Figura 5 – Validando os dados com preg_replace.

Nesse método o preg_replace valida os dados recebidos conforme os argumentos os quais foram passados, permitindo apenas a permanência de caracteres de a até z, A até Z, números e espaço em branco, sendo assim todos os caracteres maliciosos serão removidos das strings.

Para o terceiro teste foi utilizado o PDO para a manipulação do banco de dados. A manipulação ocorre por meio de parâmetros, sendo assim o PDO apenas permite a inserção dos valores em seus respectivos lugares na sentença SQL. A figura 6 apresenta um exemplo de pesquisa passando um parâmetro.

```php
<?php
    $pdo = new PDO('mysql:host=localhost;dbname=crud', 'root', '');
    $stmt = $pdo->prepare('select * from agenda where nome = :nome');
    $stmt->bindValue(':nome', 'kalil');
    $run = $stmt->execute();

    //PDO coloca cada elemento no seu lugar na sentença SQL
?>
```

Figura 6 – Utilizando PHP PDO

A função “bindValue” passa para o parâmetro um valor que é inserido na sentença SQL, sendo assim, cada parâmetro vai em seu lugar e todo e qualquer código anormal, que não esteja no seu lugar na sentença retorna um erro ao PDO.

4 CONCLUSÃO

O presente trabalho abordou métodos de prevenção ao *SQL Injection*. Após implementações dos métodos, verificou-se que o método mais eficaz é o uso do PDO, por empregar técnicas de filtragem e encapsulamento voltadas para a manipulação de bancos de dados. Os outros métodos são importantes para tarefas de recebimento dos dados enviados, contudo não evitam totalmente a injeção de SQL. Com a implementação do PDO, partir de testes de injeção de SQL, observou-se que o emprego dessa técnica é adequado para minimizar possíveis ataques de *SQL injection*.

REFERÊNCIAS

BARBOSA, E. D. e Castro R. O. (2015). Desenvolvimento de Software Seguro: Conhecendo e Prevenindo Ataque SQL Injection e Cross-site Scripting (XSS). Revista Tecnologias, Infraestrutura e Software.

CLARKE, J. SQL Injection Attacks and Defense. 1 ed. USA: Elsevier, 2009.

CLARKE, J. SQL Injection Attacks and Defense. 2 ed. USA: Elsevier, 2012.

DELUCA, C. (2018), Hacking ético: por que ele pode ser a solução para os seus problemas. Disponível em <http://cio.com.br/gestao/2018/09/27/hacking-etico-por-que-ele-pode-ser-a-solucao-para-os-seus-problemas/> Acessado em: 11/10/2018.

GARRET, F. (2012), O que é criptografia? Disponível em <https://www.techtudo.com.br/artigos/noticia/2012/06/o-que-e-criptografia.html> Acessado em: 13/10/2018.

MACÊDO, D. (2016), SQL Injection (SQLi): Entendendo e identificando a vulnerabilidade em aplicações. Disponível em <https://www.diegomacedo.com.br/entendendo-e-identificando-a-vulnerabilidade-sql-injection-sqli-em-aplicacoes/> Acessado em: 14/10/2018

ROCHA, A. S. (2008), PDO - Introdução e conceitos. Disponível em <https://www.vivaolinux.com.br/artigo/PDO-Introducao-e-conceitos> Acessado em: 13/10/2018.

SOUZA, K. K. S. (2013), Evitando SQL injection em aplicações PHP. Disponível em <https://www.devmedia.com.br/evitando-sql-injection-em-aplicacoes-php/27804> Acessado em: 13/10/2018.