Faculdade de Tecnologia de Americana – Ministro Ralph Biasi

Curso Superior de Tecnologia em Segurança da Informação

Segurança em Aplicações Web: Um estudo do SQL Injection

Leonardo Buck de Godoy, Lucas da Silva Costa, Juliane Borsato Beckedorff Pinto

leobuck12@gmail.com, lucascosta98@hotmail.com, julianeb@gmail.com

Abstract. *The purpose of this article is to introduce information security in Web applications and to import secure development against SQL Injection. The attack will be exposed through a case study, using as scenario a login system developed in PHP programming language with connection to MySQL database and analyzing the risks present in the application, based on the results, will be demonstrated ways to prevent code to attack.*

Resumo. *O objetivo deste artigo é apresentar a segurança da informação em aplicações Web e a importação do desenvolvimento seguro contra o SQL Injection. O ataque será exposto por meio de um estudo de caso utilizando como cenário um sistema de login desenvolvido na linguagem de programação PHP com conexão ao banco de dados MySQL e analisando os riscos presentes na aplicação, com base nos resultados, será demonstrado formas de prevenção de código ao ataque.*

1. Introdução

Na atualidade são desenvolvidas inúmeras aplicações na plataforma Web, como por exemplo, sistemas de vendas e compras, sistemas bancários, sistemas de comunicação, entre outros sistemas *online*. Isso torna a segurança um ponto muito importante, pois a aplicação fica propensa a diversos tipos de ataques. A infraestrutura da própria aplicação é muitas vezes a principal questão que a torna vulnerável e os seus desenvolvedores e arquitetos são os principais responsáveis.

O presente trabalho tem como objetivo apresentar um dos mais famosos ataques utilizados em uma aplicação Web, o SQL *Injection*, que consiste em digitar instruções SQL em entradas de dados de um sistema sob o intuito de descobrir informações confidenciais ou causar danos ao negócio ou à organização.

2. Segurança da Informação

Compreende-se a segurança da informação como a proteção de uso e do acesso autorizado à informação, que atualmente é um dos bens mais inestimáveis para uma organização. Um dos objetivos cruciais da segurança é a proteção dos dados, através da redução dos riscos de vazamentos, fraudes ou perdas de dados, interrupção de serviços, fatores humanos, dentre outras ameaças, as quais possam causar prejuízos a corporação.

Sêmola (2014, p. 68) define segurança da informação como "[...] área do conhecimento dedicada à proteção de ativos da informação contra acessos não autorizados, alterações indevidas ou sua indisponibilidade [...]".

Podemos considerá-la também como responsável por preservar a continuidade do negócio, mantendo as informações disponíveis, íntegras e com a garantia de sua autenticidade, identificando, documentando e combatendo as ameaças aos sistemas, a infraestrutura e os dados.

Existem três pilares principais e fundamentais da área de segurança da informação, que são:

Confidencialidade: Esse conceito se relaciona com a privacidade dos dados da organização e está diretamente ligado às ações tomadas para certificar que informações confidenciais e críticas não sejam furtadas através de ataques cibernéticos, espionagem digital, entre outros meios. Para Sêmola (2014, p. 69): "toda informação deve ser protegida de acordo com o grau de sigilo de seu conteúdo, visando a limitação de seu acesso e uso apenas a pessoas a quem é destinada".

Integridade: Corresponde à preservação das informações pela empresa ao longo dos processos ou de seu ciclo de vida, garantido a exatidão, consistência e confiabilidade das mesmas. Segundo Sêmola (2014, p. 69): "toda informação deve ser mantida na mesma condição em que foi disponibilizada pelo seu proprietário, visando protegê-la contra alterações indevidas, intencionais ou acidentais".

Disponibilidade: está associada ao tempo e à acessibilidade que se tem informações da organização, isto é, se elas podem ser consultadas a qualquer momento pelos colaboradores que possuem autorização. Sêmola (2014, p. 69): define que "toda informação gerada ou adquirida por um indivíduo ou instituição deve estar disponível aos seus usuários o momento em que eles necessitem delas para qualquer finalidade".

3. Aplicação Web

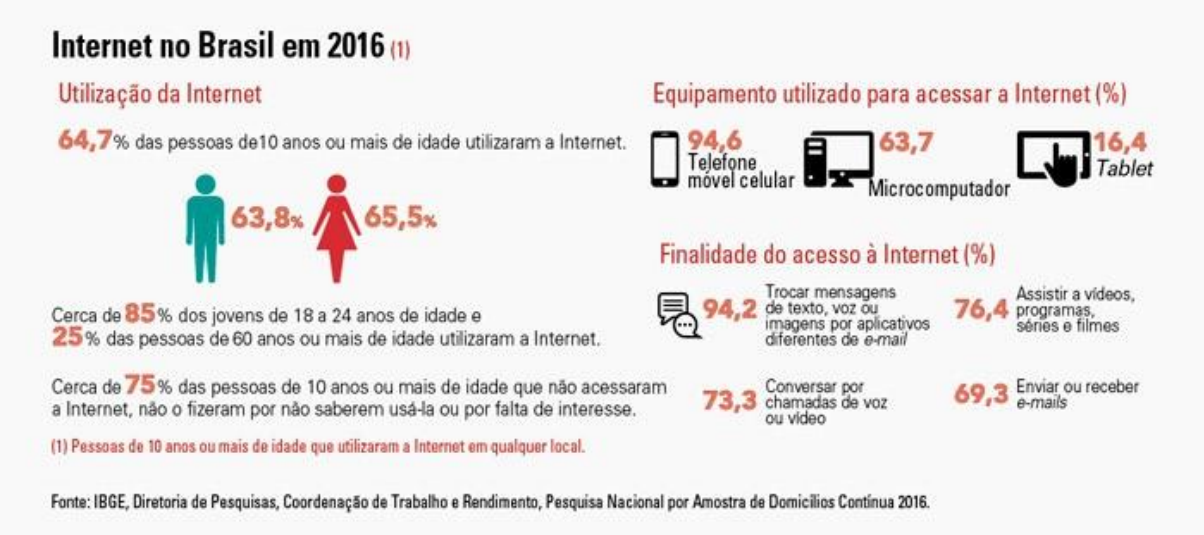
Hoje muitas pessoas acessam serviços *online* e dependem da Internet para realizar compras, acessar a sua conta bancária, conversar por meio de mensagens de texto, vídeo ou voz, entre outras atividades que foram facilitadas pela tecnologia e podem ser utilizadas principalmente por um *smartphone*.

Segundo consta no resultado da Pesquisa Nacional por Amostra de Domicílios Contínua (PNAD Contínua) realizada pelo IBGE (Instituto Brasileiro de Geografia e Estatística) no ano de 2016, 116 milhões de brasileiros fazem utilização da Internet. O meio mais utilizado para conexão foi o *smartphone* com um percentual de 94,6% de uso pelas pessoas. A atividade mais realizada na Internet pela população, com uma porcentagem de 94,2%, foi a troca de mensagens de texto, voz ou imagens por

aplicativos que não fossem por e-mail, sendo o aplicativo WhatsApp como mais popular.

As informações coletadas na pesquisa do instituto podem ser conferidas na Figura 1.

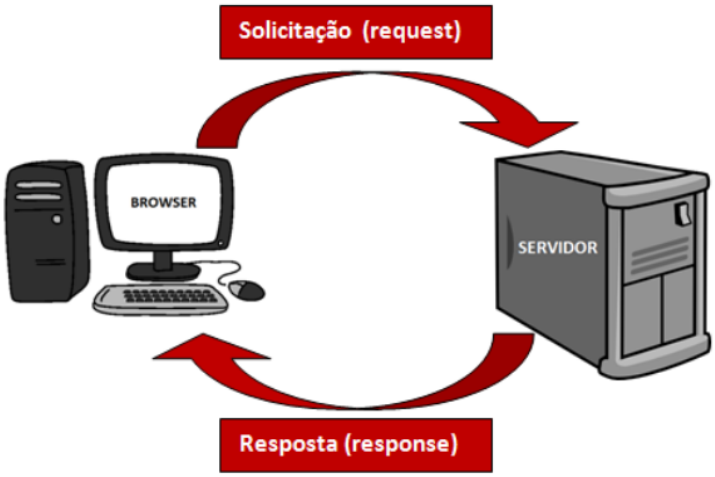
Figura 1. Utilização da Internet no Brasil



Fonte: Instituto brasileiro de geografia e estatística (2018)

A aplicação Web é um sistema executado em um servidor que pode ser acessado por meio de um navegador de Internet denominado cliente. O servidor tem como função receber as requisições enviadas pelo cliente, processar as informações recebidas, armazenar essas informações e responder de volta para o navegador. O fluxo do funcionamento de uma aplicação Web pode ser visualizado na Figura 2.

Figura 2. Fluxo de requisição e resposta entre cliente e servidor



Fonte: Devmedia (2012)

Nesse tipo de funcionamento, as aplicações Web são construídas utilizando tecnologias fundamentais no lado do cliente e no lado do servidor. As tecnologias do

lado do cliente são reconhecidas pelo navegador e são responsáveis por criar a interface e experiência com o usuário da aplicação e tem como principais exemplos:

- HTML (HyperText Markup Language - Linguagem de Marcação de Hipertexto): é uma linguagem de marcação que define a estrutura de uma página Web.
- CSS (Cascading Style Sheets - Folha de Estilo em Cascata): é um mecanismo declarativo para a definição de estilo e visual de uma página Web.
- JS (JavaScript): é uma linguagem de programação de script para interatividade e usabilidade das páginas Web com o usuário.

Já as tecnologias do lado do servidor são executadas no servidor e tem como função a implementação das ações e processamento das requisições enviadas pelo lado cliente da aplicação, podendo acessar um sistema de banco de dados.

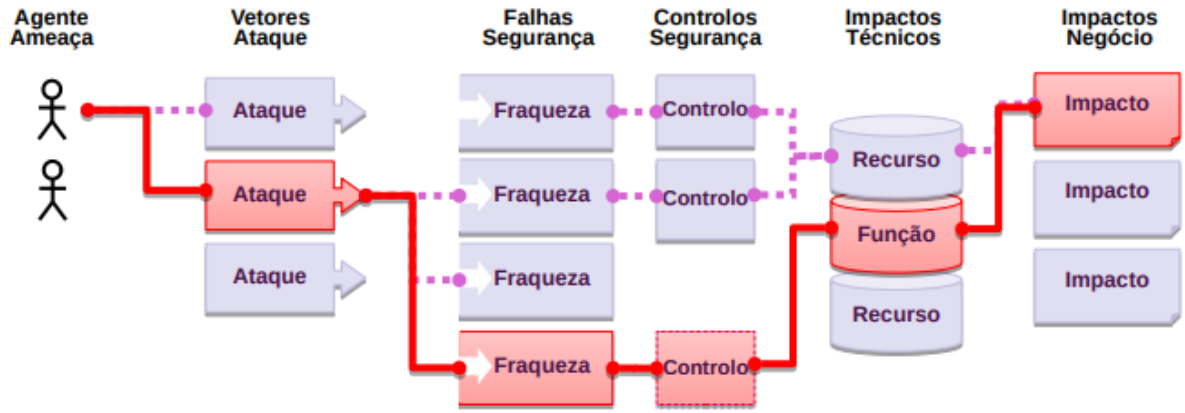
4. Riscos de Segurança em Aplicações Web

Iniciada em 2001, a OWASP (Open Web Application Security Project - Projeto Aberto de Segurança em Aplicações Web) é uma comunidade *online* que disponibiliza de forma gratuita ferramentas e normas de segurança aplicacional, livros e artigos sobre testes de segurança aplicacional, desenvolvimento de código seguro e revisão de código visando a segurança da aplicação.

Com todos as novas tecnologias voltadas para a plataforma Web, os riscos de segurança envolvendo as aplicações aumentaram conforme o tempo. A OWASP define os riscos de segurança aplicacional como as diferentes vulnerabilidades de uma aplicação que podem ser exploradas pelos atacantes para afetar um negócio ou uma organização.

Na Figura 3 é possível verificar o fluxo da tentativa de exploração de uma vulnerabilidade de um sistema. Os atacantes (agente ameaça) utilizam ataques, explorando as fraquezas, falhas ou vulnerabilidades de um sistema, tomando acesso e permissão de controle, que irá causar impactos ao negócio.

Figura 3. Fluxo de requisição e resposta entre cliente e servidor



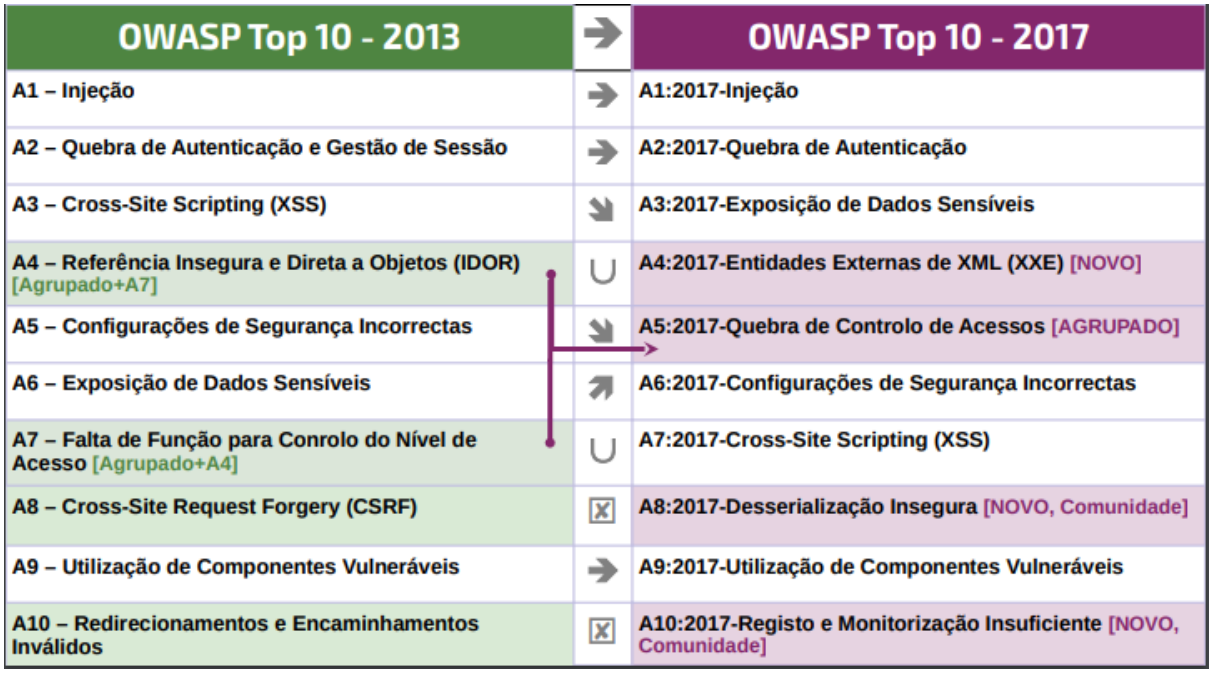
Fonte: OWASP (2017)

A OWASP desenvolveu e disponibilizou o documento "OWASP Top 10" que tem como objetivo a conscientização no desenvolvimento seguro de aplicações Web, em

que são apresentadas as mais importantes falhas de segurança em sistemas para esta plataforma. A documentação acabou tornando-se referência na área da segurança em aplicações Web.

A Figura 4 apresenta a comparação da posição das falhas de segurança publicadas no documento de 2013 e na publicação de 2017. É possível visualizar que nos dois anos, o ataque de injeção encontra-se em primeiro lugar do ranking.

Figura 4. Comparativo do OWASP Top 10 dos anos 2013 e 2017



Fonte: OWASP (2017)

5. SQL *Injection*

O SQL *Injection* é um dos mais antigos e famosos ataques contra uma aplicação Web em que são exploradas as entradas de dados dos usuários em formulário sem validação ou tratamento que podem ser injetados instruções SQL, levando a enganar o compilador ou interpretador para executar comandos maliciosos ou permitir acesso não autorizado aos dados do sistema. Alguns exemplos de entradas de dados de usuários são campos de um formulário de cadastro como nome, data de nascimento, endereço, etc. um formulário de *login* com campos de usuário e senha, um campo de pesquisa em um *site*, entre outros.

Segundo Ferreira (2017, p.4):

> O SQL Injection é um ataque bem simples de ser realizado, pois basicamente a ideia dele consiste em digitar comandos SQL nos inputs de formulários da aplicação. Se os valores digitados pelos usuários nos campos forem concatenados diretamente nos comandos SQL, sem ser realizada uma validação ou tratamento antes, certamente ela estará vulnerável a esse tipo de ataque.

Um dos cenários mais comuns para verificar se uma aplicação Web está vulnerável ao SQL *Injection* é através da tela de acesso ao sistema ao tentar realizar o *login* para autenticação.

6. Estudo de caso

Para analisar o ataque SQL *Injection*, será utilizado como estudo de caso um sistema de *login* desenvolvido pelos autores deste artigo, aplicando a linguagem de programação PHP com conexão ao banco de dados MySQL e demonstrar as formas de prevenção.

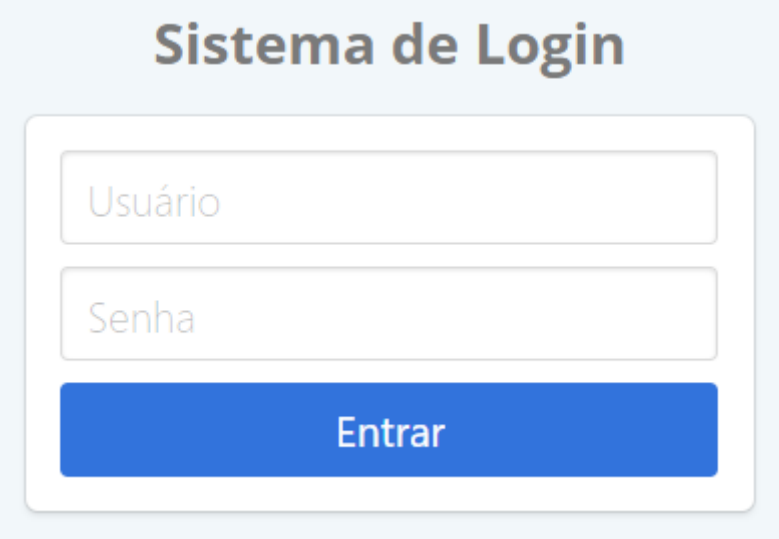
O PHP é um acrônimo para PHP: *Hypertext Preprocessor* (Pré-processador de Hipertexto), sendo uma linguagem de programação de scripts para desenvolvimento Web para processamento do lado do servidor, tendo como principais usos receber informações de formulários e gerando páginas dinâmicas.

O MySQL é um sistema gerenciador de banco de dados (SGBD) relacionais que utiliza linguagem SQL (Structure Query Language – Linguagem de Consulta Estruturada) para gerenciamento das informações em um banco de dados.

Para fins didáticos, não foi utilizada criptografia ao salvar a senha no banco de dados e o campo de senha está em texto simples para que seja possível visualizar a instrução, deixando a aplicação vulnerável.

Conforme mostrado na Figura 5, a página de *login* do sistema, possui um formulário com dois campos de entrada de dados para serem digitados, "Usuário" e "Senha", respectivamente, e o botão "Entrar", que tem como função realizar o processo de *login* com os dados informados.

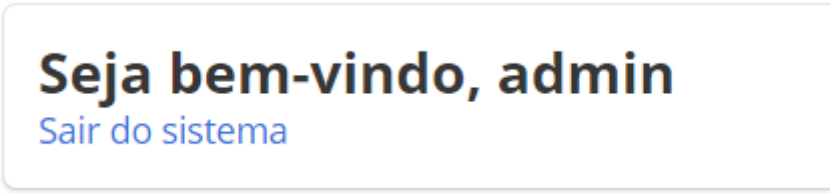
Figura 5. Tela de *Login* do Sistema



Fonte: Autoria própria (2019)

Caso seja encontrado um usuário cadastrado no sistema, o sistema será redirecionado para o painel interno com o usuário logado, conforme mostrado na Figura 6.

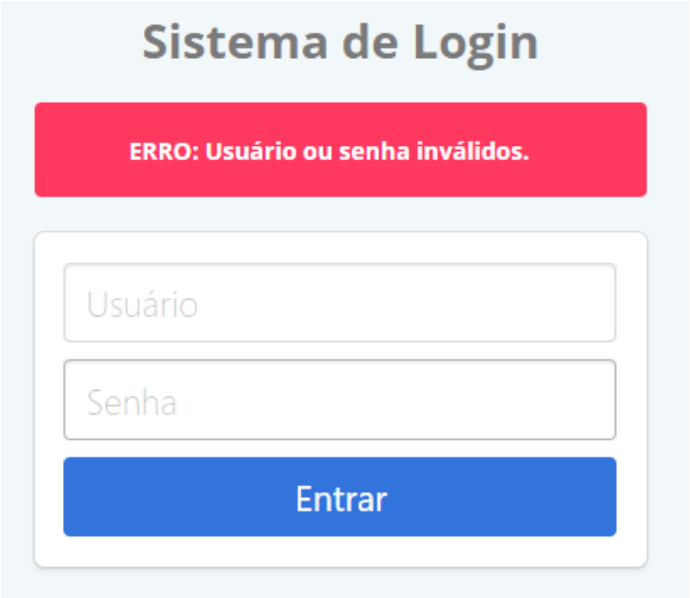
Figura 6. Tela de Painel do Sistema



Fonte: Autoria própria (2019)

Caso não exista nenhum registro cadastrado no sistema, a tela de entrada será apresentada novamente, porém será exibido uma mensagem informando que o usuário ou senha estão inválidos, assim bloqueando o acesso ao sistema e possibilitando o usuário tentar novamente o acesso, essa situação pode ser visualizada na Figura 7.

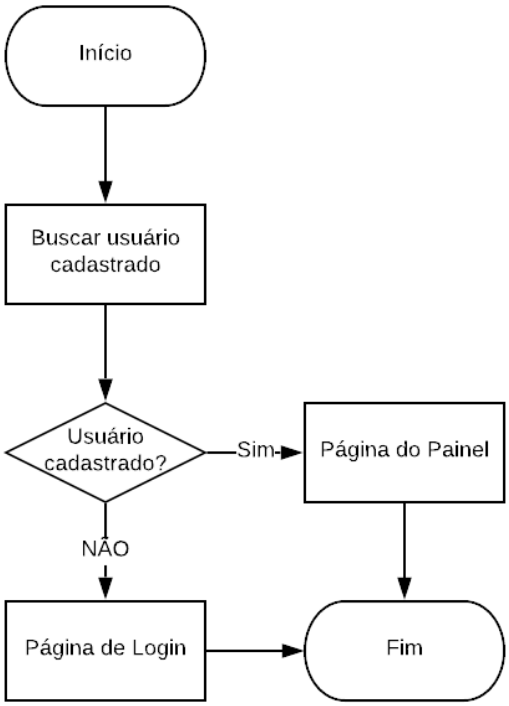
Figura 7. Tela de Painel do Sistema



Fonte: Autoria própria (2019)

Na Figura 8, é possível visualizar o fluxograma do processo de *login* do sistema.

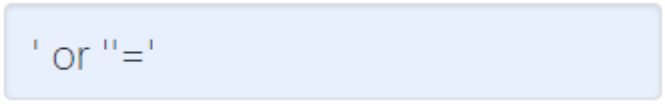
Figura 8. Fluxograma do processo de *login* do sistema



Fonte: Autoria própria (2019)

Para realizar o SQL *Injection* na aplicação basta ser digitado qualquer valor no campo de usuário, neste exemplo “sql”, e no campo de senha é preciso digitar a instrução SQL que pode ser visualizada na Figura 9.

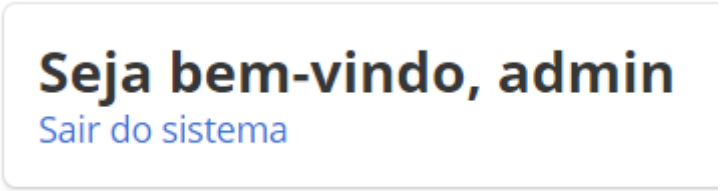
Figura 9. Campo de senha



Fonte: Autoria própria (2019)

Após clicar no botão “Entrar”, o usuário será redirecionado para a página de painel, conforme pode ser verificado na Figura 10. O atacante terá acesso ao sistema e estará autenticado como um usuário cadastrado com permissões.

Figura 10. Tela de Painel do Sistema após o SQL Injection



Fonte: Autoria própria (2019)

No teste realizado acima, a instrução SQL que buscou os registros no banco de dados foi a seguinte:

```
SELECT * FROM usuario WHERE usuario = 'sql' and senha = '' or '' = ''
```

O comando acima irá buscar todos os campos (SELECT *) da tabela "usuario" (FROM usuario) em que o campo "usuario" for igual "sql" e o campo "senha" seja vazio ou se vazio ('') for igual a vazio ('') (WHERE usuario = 'sql' and senha = '' or '' = ''). Nesse caso ele irá retornar registros, pois a condição vazio ('') igual a vazio ('') retornará verdadeira, mesmo que as condições "usuario" igual a 'sql' e "senha" igual a vazio ('') retornem falso. A análise das condições da cláusula *WHERE* pode ser verificada na Figura 11.

Figura 11. Análise da instrução SQL

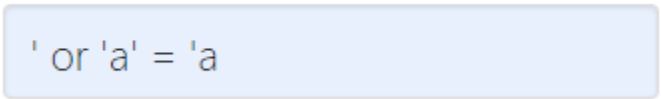
```
-- SELECT 1) COMPARAÇÃO DE VAZIOS

SELECT *
FROM usuario
WHERE usuario = 'sql'  ---> FALSO
  AND senha = ''       ---> FALSO
   OR '' = '';         ---> VERDADEIRO
```

Fonte: Autoria própria (2019)

Um outro exemplo que pode ser usado para testar o SQL *Injection* na aplicação é digitando no campo de senha o comando SQL igual da Figura 12 e no campo de usuário qualquer valor, como por exemplo "*injection*".

Figura 12. Campo de senha



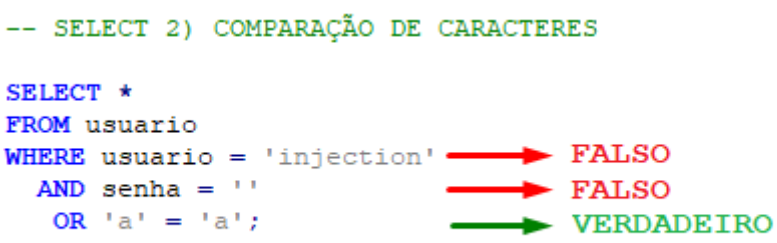
Fonte: Autoria própria (2019)

O resultado será o mesmo do teste realizado anteriormente, o usuário conseguirá acessar o sistema mesmo não possuindo uma autenticação válida. Para esse caso, o seguinte comando SQL que será executado será:

```
SELECT * FROM usuario WHERE usuario = ‘sql’ and senha = ‘’ or ‘a’ = ‘a’
```

A instrução acima buscará todos os campos (SELECT *) da tabela “usuario” (FROM usuario) onde o campo “usuario” for igual “*injection*” e o campo “senha” for vazio ou se ‘a’ for igual a ‘a’ (WHERE usuario = ‘sql’ and senha = ‘’ or ‘a’ = ‘a’). Como no exemplo anterior, as condições “usuario” igual “*injection*” e “senha” igual a vazio (‘’) retornaram falso, porém a condição ‘a’ igual ‘a’ retornará verdadeira. A análise das condições da cláusula *WHERE* pode ser verificada na Figura 13.

Figura 13. Campo de senha



Fonte Autoria própria (2019)

Para entender melhor os resultados do ataque do SQL *Injection*, é necessário analisar o código fonte da aplicação que irá realizar o processo de *login*, que pode ser visualizado na Figura 14. O fluxo da operação é iniciado com a execução do comando SQL que irá buscar os registros na tabela “usuario” com base nos valores passados para os campos “usuario” e “senha”. O resultado será armazenado em uma variável (espaço reservado na memória do computador para guardar informações que serão utilizadas durante o código) para uso futuro. Após isso, será verificado se foram retornados registros e caso encontrado um usuário, ele será armazenado e o sistema redireciona para a página de painel. Caso não sejam encontrados usuários com os valores informados, o sistema irá redirecionar de volta para a página de *login*.

Figura 14. Código fonte do processo de *login* em PHP

```php
// Conexão com banco de dados MySQL

$usuario = $_POST['usuario'];
$senha   = $_POST['senha'];

// Instrução SQL para buscar usuário
$query = "select * from usuario where usuario = '$usuario' and senha = '$senha'";

// Executa o SQL e armazena o resultado
$result = mysqli_query($conexao, $query);

// Armazena o usuario encontrado
$usuario_logado = mysqli_fetch_array($result);

// Armazena o número de registros retornados
$row = mysqli_num_rows($result);

// Redirecionamento do sistema

if ($row > 0) {    // Verifica se retornou registros
    // Armazena o usuário encontrado
    $_SESSION['usuario'] = $usuario_logado['usuario'];
    // Redireciona para a página de painel do sistema
    header('Location: painel.php');
    exit();
} else {
    // Armazena que não foram encontrados registros
    $_SESSION['nao_autenticado'] = true;
    // Redireciona para página de login
    header('Location: index.php');
    exit();
}
```

Fonte: Autoria própria (2019)

Uma boa prática para proteger uma aplicação contra o SQL *Injection* é sempre realizar a validação e o tratamento das informações que são informadas pelos usuários, evitando assim um risco aplicacional que poderá ser explorado por atacantes.

O PHP possui a função “mysqli_real_escape_string” para tratamento dos caracteres de uma sequência de texto que será utilizado em uma instrução SQL especialmente para conexão com o banco de dados MySQL. É necessário apenas

informar para a função, os parâmetros de conexão ao banco de dados e o texto em que será realizado o tratamento.

Para garantir que o usuário não esteja sendo autenticado no sistema por meio do ataque, é possível realizar uma comparação para que os campos de usuário e senha informados tenham os mesmos valores dos campos do registro que está sendo retornado na busca do banco de dados. Caso os valores sejam os mesmos, será autenticado o usuário, caso contrário, a página de *login* será apresentada novamente.

O processo de *login* deverá verificar se foi retornado apenas um registro de usuário na busca do banco de dados fundamentado nos dados digitados.

Outra forma de evitar o SQL *Injection* é o uso de *Prepared Statements* (Comandos Preparados), que tem como função a execução de uma instrução SQL sem concatenar os valores diretamente a ela e sim passando os valores como parâmetro para o comando. Os próprios métodos possuem mecanismos de segurança ao ataque. No PHP, é possível utilizar a classe de comandos preparados "MySQLi_STMT". O processo de *login* do sistema desenvolvido como cenário para o presente artigo foi reestruturado para utilizar comandos preparados, protegendo-o assim do SQL *Injection*. O novo código fonte pode ser visualizado na Figura 15 já com a implementação da validação dos dados digitados pelo usuário com a função "mysqli_real_escape_string" mencionada anteriormente.

Figura 15. Código fonte do processo de *login* em PHP com *Prepared Statements*

```php
// Conexão com banco de dados MySQL
// Validação dos caracteres dos dados digitados
$usuario = mysqli_real_escape_string($conexao, $_POST['usuario']);
$senha   = mysqli_real_escape_string($conexao, $_POST['senha']);

// Instrução SQL para buscar usuários
$query = "SELECT * FROM usuario WHERE usuario = ? and senha = ?";

// Prepara a instrução SQL
if ($stmt = mysqli_prepare($conexao, $query)) {
    // Define os parâmetros dos valores
    mysqli_stmt_bind_param($stmt, "ss", $usuario, $senha);

    // Executa a instrução SQL com Prepared Statements
    mysqli_stmt_execute($stmt);

    // Armazena o resultado
    $result = mysqli_stmt_get_result($stmt);

    // Armazena o usuário encontrado
    $usuario_logado = mysqli_fetch_array($result);

    // Redicionamento do sistema
    // Verifica se retornou apenas um usuario e se os dados digitados são iguais do registro retornado
    if ((mysqli_num_rows($result) == 1) && ($usuario == $usuario_logado['usuario'])
        && ($senha == $usuario_logado['senha'])) {
        $_SESSION['usuario'] = $usuario_logado['usuario'];
        header('Location: painel.php');
        exit();
    } else {
        $_SESSION['nao_autenticado'] = true;
        header('Location: index.php');
        exit();
    }
}
```

Fonte: Autoria própria (2019)

7. Considerações Finais

Assim, seguindo as formas de prevenção apresentadas no decorrer do estudo de caso, a aplicação Web será menos vulnerável ao SQL *Injection*. Neste artigo foi abordado apenas como cenário o processo de *login* do sistema, porém todas as páginas da aplicação que fazem conexão e operação ao banco de dados com valores informados pelo usuário fazem-se necessários utilizar métodos seguros contra esse tipo de ataque.

Referências

AUTRAN, Felipe. Brasil tem 116 milhões de pessoas com acesso à internet, diz pesquisa. In: Tecmundo, 21/02/2018. Disponível em: https://www.tecmundo.com.br/mercado/127430-brasil-116-milhoes-pessoas-acesso-internet-pesquisa-ibge.htm. Acesso em: 29 set. 2019.

BENTO, Evaldo Junior. **Desenvolvimento web com PHP e MySQL**. São Paulo: Casa do Código, 2013.

FERREIRA, Rodrigo**. Segurança em aplicações web**. São Paulo: Casa do Código, 2017.

HYPERTEXT PREPROCESSOR. O que o PHP pode fazer?. In: Hypertext Preprocessor (PHP), 2019. Disponível em: https://www.php.net/manual/pt_BR/intro-whatcando.php. Acesso em: 07 out. 2019.

MARCUS, Daniel. PHP MySQLi prepared statements tutorial to prevent SQL Injection. In: WebSiteBeaver, 08/11/2017. Disponível em: https://websitebeaver.com/prepared-statements-in-php-mysqli-to-prevent-sql-injection. Acesso em: 07 out. 2019.

MEDEIROS, Higor. SQL Injection em múltiplas plataformas. In: Devmedia, 2014. Disponível em: https://www.devmedia.com.br/sql-injection-em-multiplas-plataformas/31389. Acesso em: 14 set. 2019.

MELHORHOSPEDAGEMDESITES. O que é front end e back end: entenda esses termos significam. In: Melhorhospedagemdesites, 11/07/2019. Disponível em: https://www.melhorhospedagemdesites.com/dicas-e-ferramentas/front-end-e-back-end/. Acesso em: 05 out. 2019.

OPEN WEB APPLICATION SECURITY PROJECT. OWASP Top 10 - 2017: the ten most critical web application security risks. In: Open Web Application Security Project, 04/2017. Disponível em: https://www.owasp.org/images/0/06/OWASP_Top_10-2017-pt_pt.pdf. Acesso em: 17 set. 2019.

ORACLE. O que é o MySQL?. In: MySQL, 27/07/2018. Disponível em: https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html. Acesso em: 07 out. 2019.

PALMEIRA, Thiago Vinícius Varallo. Como funcionam as aplicações web. In: Devmedia, 2012. Disponível em: https://www.devmedia.com.br/como-funcionam-as-aplicacoes-web/25888. Acesso em: 05 out. 2019.

PISA, Pedro. O que é e como usar o MySQL?. In: Tecmundo, 17/04/2012. Disponível em: https://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html. Acesso em: 05 out. 2019.

RENAUX, Pedro. Nove entre dez usuários de Internet no país utilizam aplicativos de mensagens. In: Agência de notícias (Instituto brasileiro de geografia e estatística), 10/04/2018. Disponível em: https://agenciadenoticias.ibge.gov.br/agencia-noticias/2012-agencia-de-noticias/noticias/20077-nove-entre-dez-usuarios-de-internet-no-pais-utilizam-aplicativos-de-mensagens. Acesso em: 29 set. 2019

SÊMOLA, Marcos. **Gestão da segurança da informação**. 2. ed. Rio de Janeiro: Elsevier, 2014.

SOUZA, Kalil K. Silva. Evitando SQL Injection em aplicações PHP. In: Devmedia, 2013. Disponível em: https://www.devmedia.com.br/evitando-sql-injection-em-aplicacoes-php/27804. Acesso em: 14 set. 2019.

TUTORIALREPUBLIC. PHP MySQL prepared statements. In: TutorialRepublic, 2019. Disponível em: https://www.tutorialrepublic.com/php-tutorial/php-mysql-prepared-statements.php. Acesso em: 07 out. 2019.

VIANA, Daniel. O que é front-end e back-end?. In: Treinaweb, 30/01/2017. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-front-end-e-back-end. Acesso em: 05 out. 2019.

Fatec Americana
Ministro Ralph Biasi

Centro Paula Souza 50 anos 1969 - 2019

SÃO PAULO
GOVERNO DO ESTADO

Faculdade de Tecnologia de Americana

Leonardo Buck de Godoy
Lucas da Silva Costa

SEGURANÇA EM APLICAÇÕES WEB: UM ESTUDO DO SQL INJECTION

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Segurança da Informação pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana.
Área de concentração: Segurança da Informação.

Americana, 06 de dezembro de 2019.

Banca Examinadora:

Juliane Borsato Beckedorff Pinto (Presidente)
Especialista
Fatec Americana

Edson Roberto Gaseta (Membro)
Mestre
Fatec Americana

Benedito Luciano Antunes De França (Membro)
Mestre
Fatec Americana