FACULDADE DE TECNOLOGIA DE SÃO PAULO

Gabriela Whitaker Visani

Segurança contra Injeção de SQL: Análise do Ataque, Estratégias de Proteção e Avaliação do Nível de Preparo dos Programadores na Prevenção.

SÃO PAULO

2023

FACULDADE DE TECNOLOGIA DE SÃO PAULO

Gabriela Whitaker Visani

Segurança contra Injeção de SQL: Análise do Ataque, Estratégias de Proteção e Avaliação do Nível de Preparo dos Programadores na Prevenção.

Trabalho submetido como exigência parcial

para a obtenção do Grau de Tecnólogo em

Análise e Desenvolvimento de Sistemas

Orientador: Professor Mestre Edson Ceroni

SÃO PAULO

2023

Agradecimentos

Gostaria de expressar minha profunda gratidão ao Professor Mestre Edson Ceroni, que dedicou seu tempo, conhecimento e orientação na condução deste trabalho de conclusão de curso. Sua orientação foi fundamental para o desenvolvimento deste projeto, e estou extremamente grata.

Quero estender meus sinceros agradecimentos, também, à minha esposa, Natália Dalibera. Durante toda a minha jornada acadêmica, ela foi meu pilar de apoio, minha fonte de incentivo e motivação. Seu apoio e compreensão foram essenciais para que eu pudesse alcançar este marco em minha vida.

Por fim, quero agradecer a todos que responderam o questionário que deu base a esse TCC.

Resumo

Este estudo tem como objetivo aprofundar a compreensão do ataque de injeção de SQL (Structured Query Language), sua execução, estratégias contemporâneas de prevenção e a conscientização da comunidade de programadores em relação a essas práticas preventivas. Por meio de uma análise abrangente, o trabalho examina o panorama do ataque de injeção de SQL e avalia a eficácia das medidas preventivas em prática, revelando o nível de preparo do programador para mitigar essa ameaça de segurança.

Palavras chaves: Injeção de Código, SQL, Segurança da informação.

Abstract:

This study aims to deepen the understanding of SQL injection attack, its execution, contemporary prevention strategies, and the awareness of the programmer community regarding these preventive practices. Through a comprehensive analysis, the work examines the landscape of SQL injection attack and assesses the effectiveness of preventive measures in practice, revealing the level of preparedness of the programmer to mitigate this security threat.

Keywords: Code Injection, SQL, Information Security.

Sumário

1. Introdução

O campo da segurança da informação está em constante evolução, e a crescente dependência de sistemas de informação e aplicativos web torna crucial o estudo e compreensão das ameaças cibernéticas que permeiam esse ecossistema digital. Dentro dessa paisagem complexa, a injeção de SQL (Structured Query Language), conhecida pela abreviação SQLi, emerge como uma das vulnerabilidades mais comuns, representando uma ameaça constante para a integridade e confidencialidade dos dados armazenados e processados por sistemas de bancos de dados.

O ataque de injeção de SQL existe desde o final dos anos 90, como observado por Sharma e Bhatt: "A evolução do ataque de injeção de SQL foi observada pela primeira vez por volta de 1998, conforme relatado na Phrack Magazine" (Sharma; Bhatt, 2019, p. 494). Desde então, essa técnica maliciosa tem sido utilizada em inúmeras ocasiões. Conforme Joseph Cox mencionou em um artigo publicado no site Vice, a técnica de injeção de SQL foi empregada para extrair informações da Organização Mundial da Saúde, do Wall Street Journal e até comprometer os sites de agências federais dos Estados Unidos (Cox, 2015).

Mais recentemente, em 2021, um hacker conseguiu atacar repetidamente o site do Datasus, como ilustrado na Figura 1. Na segunda investida, o invasor não apenas comprometeu a segurança do Datasus, mas também adicionou um texto ao site, destacando as vulnerabilidades identificadas, incluindo brechas específicas para injeção de SQL (SQLi) (Ramon de Souza, 2021).

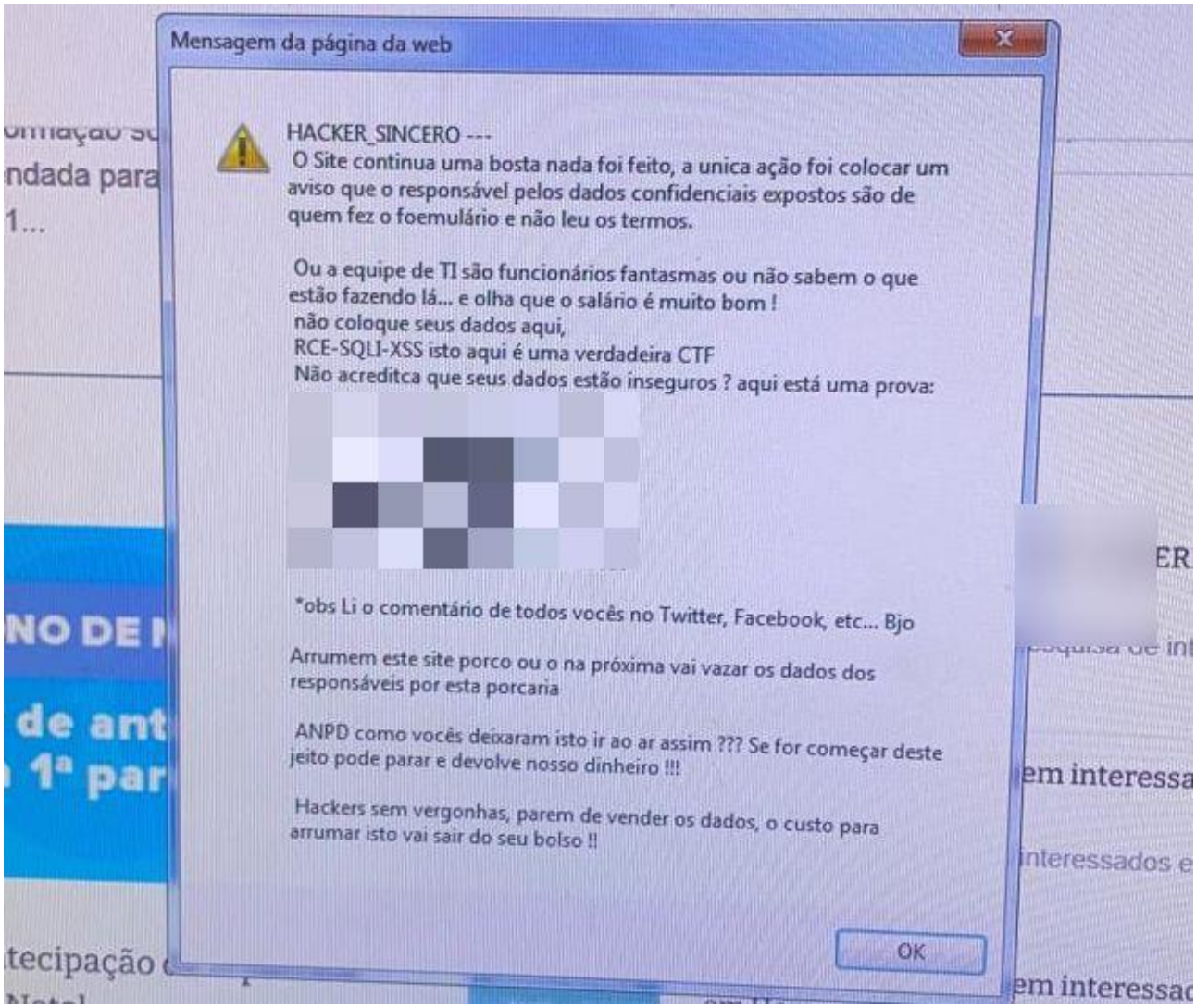

Figura 1 - Mensagem adicionada por hacker no site do Datasus. Fonte:https://canaltech.com.br/seguranca/invasor-volta-a-atacar-datasus-e-zomba-da-seguranca-do-site-do-governo-179179/

A Open Web Application Security Project (OWASP), em 2021, publicou a lista das 10 vulnerabilidades mais comuns, ataques de injeção ficaram em terceiro lugar. De acordo com a organização “94% das aplicações foram testadas para alguma forma de injeção”, distinguindo pelos 33 tipos de ataques de injeção mapeados. As taxas de incidência variaram entre 3,37% e 19% e essa categoria possui o segundo maior número de ocorrências em aplicações, com 274.228,00 ocorrências no total. Esses dados mostram o quanto comum ataques de injeção ainda são, mesmo com todo o avanço em técnicas de prevenção.

A Common Weakness Enumeration (CWE) é gerenciada pelo Instituto de Engenharia e Desenvolvimento de Sistemas de Segurança Interna (HSSEDI), operada pela The MITRE Corporation (MITRE) e patrocinada pelo Departamento de Segurança Interna

dos Estados Unidos (DHS), Agência de Segurança Cibernética e Infraestrutura (CISA). A CWE é uma tipificação, enumeração e ordenamento de acordo com a gravidade das vulnerabilidades de software, firmware, hardware ou serviço. Em seu site oficial aparece que: “A lista "2023 CWE Top 25 Most Dangerous Software Weaknesses" foi calculada analisando dados públicos de vulnerabilidade no U.S. National Vulnerability Database (NVD) para suas causas raiz por meio de mapeamentos CWE.” (Mitre, 2023, tradução minha). Nessa lista de 2023 em terceiro lugar apareceu “Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')”. O fato de a injeção de SQL ocupar uma posição tão destacada nessa lista ressalta a gravidade dos ataques de SQL.

1.1 Objetivo

Este trabalho tem como objetivo investigar e aprofundar o entendimento sobre a injeção de SQL, uma técnica maliciosa utilizada por cibercriminosos para explorar vulnerabilidades em aplicações web e obter informações ou até ganhar acesso não autorizado a sistemas de banco de dados. A injeção de SQL é uma ameaça que persiste, apesar dos avanços na segurança cibernética, em especial nas técnicas de evitar esse tipo de ataque. Sua prevalência serve como um lembrete constante da necessidade de compreender, prevenir e mitigar os riscos associados a esse tipo de ataque.

Neste estudo, serão explorados os princípios fundamentais da injeção de SQL, suas diferentes formas de execução e as consequências devastadoras que podem resultar de um ataque bem-sucedido. Além disso, serão abordadas as estratégias contemporâneas de prevenção e detecção desse tipo de ameaça, bem como as melhores práticas para proteger sistemas e aplicativos web contra ataques de injeção de SQL. Por fim, será analisado o nível de preparo de programadores em relação às técnicas de prevenção a esse tipo de ataque.

1.2 Justificativa

A opção por abordar a injeção de SQL como foco principal deste Trabalho de Conclusão de Curso (TCC) parte da clara necessidade de compreender e enfrentar os desafios representados por um dos ataques cibernéticos mais perigosos e persistentes da atualidade. A injeção de SQL figura como uma ameaça constante à integridade e confidencialidade dos dados em sistemas conectados a bancos de dados, destacando-se como uma vulnerabilidade significativa em um cenário digital cada vez mais complexo. Surge, portanto, a importância de revisar e analisar estratégias de prevenção existentes.

Adicionalmente, este trabalho busca avançar na compreensão dos motivos que perpetuam a ocorrência da injeção de SQL, investigando o nível de conhecimento e preparo dos profissionais vinculados ao desenvolvimento de software diante dessa ameaça. A análise desses aspectos é crucial para identificar lacunas e promover ações educativas e preventivas eficazes.

Dessa forma, este estudo aprofunda a compreensão da injeção de SQL, não apenas como uma investigação técnica, mas como uma contribuição para a comunidade de tecnologia. Ao estudar os mecanismos desse tipo de ataque, a pesquisa visa fornecer conhecimentos práticos que possam ser aplicados no desenvolvimento de estratégias mais eficazes no combate à injeção de SQL, contribuindo assim para a segurança cibernética e a resistência efetiva contra essa ameaça persistente.

1.3 Metodologia

A metodologia adotada neste trabalho consiste em uma revisão bibliográfica abrangente sobre o tema, seguida pela aplicação de um questionário direcionado a profissionais nas áreas relacionadas ao desenvolvimento de sistemas. A revisão bibliográfica será dividida em duas seções para proporcionar uma compreensão mais aprofundada.

A primeira seção, intitulada "O que é Injeção de SQL", aborda definições, tipos e exemplos de ataques de injeção de SQL. Destaca as potenciais consequências e ameaças que esse tipo de ataque representa para os sistemas de informação. A segunda seção, denominada "Métodos de Prevenção", examina as melhores práticas e medidas recomendadas para prevenir a injeção de SQL, incluindo o uso de instruções preparadas, validação de entrada, sanitização, entre outros métodos de proteção.

Além da revisão bibliográfica, conduziu-se à coleta de dados sobre o nível de conhecimento e conscientização de profissionais da área de tecnologia em relação à injeção de SQL e seus métodos de prevenção. Essa coleta foi realizada por meio de um questionário estruturado (anexo 1), abordando tópicos como o perfil do respondente, conhecimento básico sobre Injeção de SQL e familiaridade com práticas de prevenção e mitigação. O questionário foi distribuído entre profissionais da área de tecnologia em grupos de facebook como: “Programadores JAVA”, “Clube dos Programadores” e “Programação Web”; grupos de Whatsapp como: “DevHack”, “Análise e desenvolvimento de Sistemas”, “ADS UNOPAR/ANHANGUERA”, “Grupos de T.I / Linguagens de Programação”, “Fatec ADS 2019/01 Noturno”, “TI Em Prática”, “Mainframe FATEC”, “Developers Brasil”, “DEV’S - GLOBAL”, “Fatec ADS - Turma 281/282”, “Programadores sem emprego” e “ADS”; grupo de Telegram: “Grupy-SP” e o chat “chat-geral” do servidor do Discord “Let’s Code”.

A coleta de dados online foi realizada com o compromisso de preservar a privacidade dos participantes, garantindo a confidencialidade de suas respostas. Essa abordagem metodológica visa não apenas aprofundar o entendimento da injeção de SQL, mas também contribuir para a conscientização e aprimoramento da segurança cibernética nas organizações, considerando os crescentes desafios no cenário de ameaças digitais.

2. Definição e tipificação de ataques de injeção de SQL

SQL, ou Structured Query Language (Linguagem de Consulta Estruturada), é uma linguagem de programação projetada para gerenciar, manipular e consultar bancos de dados relacionais. Ela é amplamente utilizada em sistemas de gerenciamento de bancos de dados (DBMS) como o MySQL, Oracle, SQL Server, PostgreSQL e outros. SQL fornece um conjunto de comandos e instruções que permitem aos usuários realizar diversas operações em bancos de dados, tais como selecionar (select), inserir (insert), modificar dados (update), exclusão de dados (delete).

A injeção de código é uma vulnerabilidade de segurança que ocorre quando dados não confiáveis são incorporados a um programa e executados como parte do código. Essa prática é explorada por atacantes para inserir comandos maliciosos em um sistema, geralmente por meio de entradas de usuário não validadas. Existem vários tipos de ataques de injeção como injeção de script (XSS), injeção de comando, injeção de LDAP e injeção SQL.

Um ataque de injeção de SQL é uma técnica de exploração cibernética que visa manipular sistemas de gerenciamento de banco de dados (DBMS) por meio da inserção de comandos SQL maliciosos em entradas de dados não sanitizadas ou inadequadamente validadas. De acordo com Elshazly *et al.* "É um método pelo qual os parâmetros de uma aplicação baseada na web são modificados a fim de alterar as declarações SQL que são enviadas a um banco de dados de backend". Nesse tipo de ataque o invasor executa operações não autorizadas no banco de dados, como consulta, modificação ou exclusão de dados sensíveis. Os ataques de injeção de SQL podem ter consequências devastadoras, incluindo a exposição de informações confidenciais, corrupção de dados e comprometimento da integridade do sistema.

Para teste, a OAWSP separa os ataques de injeção de SQL de acordo com onde eles apresentam o resultado: Inband (os dados são extraídos usando o mesmo canal), Out-of-band (os dados são recuperados usando um canal diferente) e *Inferencial* ou *Blind* (não há transferência real de dados, mas o testador é capaz de reconstruir as informações). A organização também distingue o ataque em 5 categorias (Union Operator, Booleana, Baseada em erro, Out-of-band, e Atraso de tempo). Não aparece nessa separação da OAWSP, mas alguns altores como Singh (2016) também ultilizam do conceito Injeção de SQL Clássica, principalmente para diferenciar de outros tipos mais complexos como

ataques cegos (*blind* SQLi) e ataques de SQL que se misturam com outros tipos de ataques como SQLi com elementos de Ddos (*Distributed Denial Of Service*).

Injeção de SQL Clássica: Nesse tipo de ataque, o invasor insere comandos SQL diretamente em campos de entrada de dados de uma aplicação web. Esses comandos podem ser inseridos em formulários da web, campos de pesquisa ou parâmetros de URL. Como exemplo, no seguinte código de um formulário de login de um sistema fictício, se uma pessoa ao invés de colocar um usuário e senha colocasse no campo de usuário e senha ’ or ‘1’=’1’ ela conseguiria entrar sem ter um cadastro no sistema.

```
import sqlite3

def login_sistema(username, password):
    conn = sqlite3.connect("database.db")
    cursor = conn.cursor()

    query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"
    cursor.execute(query)

    user = cursor.fetchone()
    conn.close()

    return user

username = input("Digite seu nome de usuário: ")
password = input("Digite sua senha: ")

user = login_sistema(username, password)
```

Union Operator: A injeção de SQL por *Union Operator* é uma técnica usada em ataques de injeção de SQL, na qual o invasor explora vulnerabilidades em uma aplicação web para injetar instruções SQL maliciosas que incluem a cláusula "UNION" em uma consulta SQL. A cláusula "UNION" é usada em SQL para combinar os resultados de duas consultas diferentes em uma única tabela resultante.

Boolean: o invasor explora vulnerabilidades em uma aplicação web ou sistema para manipular consultas SQL usando lógica booleana. Ao usar essa técnica, o invasor tenta inferir informações sensíveis do banco de dados, verificando se certas condições são verdadeiras ou falsas, com base nas respostas do sistema.

Blind SQL Injection: é uma variação do ataque de injeção de SQL na qual o invasor explora vulnerabilidades em uma aplicação web, mas, ao contrário da Injeção de SQL clássica, não recebe diretamente os resultados da consulta ou dados do banco de dados. Em vez disso, o invasor realiza perguntas binárias para determinar se uma condição é verdadeira ou falsa, usando a aplicação como um canal de comunicação indireto. Esse tipo de ataque é "cego" porque o invasor não obtém resultados visíveis da consulta, mas consegue inferir informações sensíveis com base nas respostas do sistema. Um exemplo disso seria em um site que tivesse uma pesquisa que retorne resultados de acordo com o nome do usuário. Um invasor pode injetar uma consulta como esta:

http://www.siteexemplo.com/pesquisa?nome=qualquer_nome' AND 1=2 UNION SELECT NULL, password FROM usuarios--

Se a aplicação não estiver protegida contra Blind SQL Injection, ela pode responder de maneira diferente com base na condição 1=2. Se o resultado da pesquisa não retornar nenhum resultado, o invasor pode inferir que a consulta aninhada está correta e que a tabela "usuarios" existe.

Time-Based Blind SQL Injection: nessa variação do ataque cego, o invasor utiliza atrasos de tempo para determinar se uma condição é verdadeira ou falsa. Nesse tipo de ataque, o invasor cria consultas que fazem o servidor de banco de dados aguardar por um período específico antes de responder. Com base no tempo que o servidor leva para responder, o invasor pode inferir se a condição da consulta é verdadeira ou falsa. Essa técnica é especialmente eficaz quando o servidor de banco de dados não divulga mensagens de erro que possam ser usadas para determinar a validade da consulta. Para exemplificar esse tipo de ataque, suponha que um site tenha um campo de pesquisa e um invasor deseja determinar se o nome de usuário "admin" tem uma senha que começa com a letra "a." Para fazer isso, o invasor pode injetar uma consulta que introduza um atraso de 5 segundos se a condição for verdadeira. A consulta injetada pode se parecer com isso:

http://www.siteexemplo.com/pesquisa?nome=admin' AND IF(1=1, SLEEP(5), 0)--

Nesse exemplo, o invasor está verificando se a condição 1=1 é verdadeira. Se for verdadeira, o servidor de banco de dados será instruído a "dormir" por 5 segundos usando SLEEP(5). Se o atraso de tempo ocorrer, o invasor pode concluir que a condição é verdadeira e que a senha do usuário "admin" começa com a letra "a."

Injeção de SQL baseada em erro: Nesse cenário, o invasor explora vulnerabilidades em uma aplicação web para obter informações sensíveis do banco de dados, aproveitando mensagens de erro geradas pelo sistema de gerenciamento de banco de dados (DBMS). Quando a aplicação não trata adequadamente erros do DBMS, mensagens de erro detalhadas podem ser exibidas, revelando informações úteis, como a estrutura da tabela ou dados sensíveis. Como exemplo desse ataque, pode ser imaginado um site tenha uma página de pesquisa onde os usuários podem pesquisar produtos por nome com a seguinte URL:

http://www.siteexemplo.com/pesquisa?nome=produto

Se um invasor inserir uma consulta maliciosa no campo de pesquisa, como:

http://www.siteexemplo.com/pesquisa?nome=' UNION SELECT 1, table_name FROM information_schema.tables--

A aplicação pode executar uma consulta SQL que tente unir os resultados com uma tabela do sistema de informações do banco de dados, como a tabela information_schema.tables. Se a consulta for bem-sucedida, a aplicação poderá exibir uma mensagem de erro que inclui detalhes sobre a estrutura da tabela, como os nomes das tabelas. O invasor, então, pode analisar essas mensagens de erro para extrair informações confidenciais.

Por exemplo, a aplicação pode gerar um erro que inclui uma mensagem como:

Erro de SQL: a tabela 'usuarios' não existe.

A partir dessa mensagem, o invasor pode inferir que a tabela 'usuarios' existe no banco de dados, obtendo assim informações sobre a estrutura do banco de dados, mesmo sem ver os resultados reais da consulta.

Injeção de SQL Out-of-Band (OOB): Esse tipo de ataque explora vulnerabilidades em uma aplicação web para extrair dados do banco de dados, mas em vez de obter os resultados diretamente, o invasor usa canais alternativos, como solicitações DNS ou HTTP,

para exfiltrar as informações. Essa técnica é especialmente útil quando a aplicação web bloqueia ou não permite diretamente a exibição dos resultados da consulta. Para filtrar os dados, o invasor configura um servidor controlado por ele e faz com que o servidor da aplicação web faça solicitações a esse servidor. Isso pode ser feito de várias maneiras:

Solicitação DNS: O invasor pode configurar um servidor DNS malicioso e inserir comandos na consulta que farão com que o servidor da aplicação faça solicitações DNS para o servidor controlado pelo invasor. Essas solicitações DNS conteriam informações filtradas, como parte dos resultados da consulta.

Solicitação HTTP: O invasor pode configurar um servidor HTTP malicioso e fazer com que a aplicação web faça solicitações HTTP para esse servidor, incluindo dados filtrados como parâmetros na URL ou no corpo da solicitação.

Outros Canais: Além de DNS e HTTP, outras formas de canais alternativos, como SMTP (email) ou qualquer protocolo que permita a transferência de informações, também podem ser usadas para filtrar dados.

3. Métodos de prevenção a ataques de injeção de SQL

A prevenção de ataques de injeção de SQL é crucial para garantir a segurança de aplicativos web e sistemas que interagem com bancos de dados. Esses ataques representam uma ameaça significativa à integridade e à confidencialidade dos dados. De acordo com Sadeghian *et al.* algumas das boas práticas para programação, que contribuem na prevenção desse tipo de ataque é utilizar: validação e sanitização de entrada de dados, parametrização, procedimentos armazenados, e filtros por whitelisting.

Validação e Sanitização de Entrada de Dados: A validação e a sanitização de entrada de dados são processos fundamentais no desenvolvimento de aplicativos para garantir a segurança e a integridade dos dados manipulados. Ambos têm como objetivo impedir a inserção de dados maliciosos ou inválidos em um sistema. No entanto, eles desempenham papéis ligeiramente diferentes.

A validação de entrada de dados refere-se ao processo de verificar se os dados fornecidos atendem a critérios específicos e são aceitáveis antes de serem processados ou armazenados. Isso envolve garantir que os dados estejam no formato correto, atendam aos requisitos de tamanho e não contenham caracteres inaceitáveis.

A sanitização de entrada de dados é o processo de limpar ou "desinfetar" os dados de entrada, removendo qualquer conteúdo malicioso, caracteres especiais ou código potencialmente prejudicial que possa ser usado para explorar vulnerabilidades no sistema. A sanitização tem como objetivo tornar os dados seguros e inofensivos, mas não necessariamente validá-los. Ao combinar a validação e a sanitização de entrada de dados é possível reduzir significativamente a superfície de ataque e minimizar as chances de ataques de injeção de SQL bem-sucedidos.

Instruções Preparadas (Prepared Statements) e Parametrização: Instruções preparadas são um mecanismo fundamental para prevenir ataques de injeção de SQL em aplicativos que interagem com bancos de dados. Essas instruções separam a consulta SQL da entrada do usuário, garantindo que os dados de entrada não sejam interpretados como parte da consulta. O mecanismo de instruções preparadas e a parametrização também podem lidar com o escapamento automático de caracteres especiais nos parâmetros. Isso impede que os caracteres especiais sejam interpretados como comandos SQL, o que pode fazer parte de uma estratégia de sanitização dos campos.

Além disso, as instruções preparadas evitam a concatenação de dados de entrada, uma vez que, os dados de entrada não são concatenados diretamente na consulta. O seguinte código em PHP traz um exemplo de uso de instruções preparadas:

```
// Conexão com o banco de dados

$pdo = new PDO('mysql:host=localhost;dbname=seubanco', 'seuusuario', 'suasenha');

// Consulta preparada

$stmt = $pdo->prepare("SELECT nome, email FROM usuarios WHERE username = :username");

// Parâmetros

$username = $_POST['username'];

// Atribuição dos parâmetros e execução da consulta

$stmt->bindParam(':username', $username, PDO::PARAM_STR);

$stmt->execute();

// Obtenção dos resultados

$resultado = $stmt->fetch();
```

Neste exemplo, a consulta SQL é preparada com um marcador de posição :username, que é posteriormente vinculado ao valor fornecido pelo usuário ($username). A consulta é executada de forma segura, pois o mecanismo de instruções preparadas trata automaticamente o escapamento e evita que a entrada do usuário seja interpretada como parte da consulta SQL. Como resultado, mesmo se o usuário fornecer um nome de usuário malicioso, isso não levará a uma injeção de SQL bem-sucedida.

Procedimentos Armazenados (Stored Procedures): Implementação lógica de banco de dados em procedimentos armazenados, que são pré definidos e armazenados no banco de dados. Essas instruções podem ser chamadas e executadas pelo aplicativo ou por outros procedimentos, permitindo que as operações no banco de dados sejam encapsuladas em uma unidade lógica reutilizável.

Em procedimentos armazenados, as consultas SQL podem ser parametrizadas, o que significa que os parâmetros passados para o procedimento são tratados de maneira segura e não são diretamente interpretados como código SQL. Isso evita a necessidade de concatenar dados de entrada diretamente em consultas, reduzindo assim o risco de injeção de SQL. Os valores dos parâmetros são tratados como dados e não como comandos SQL.

Uso de Listas Brancas (Whitelisting): Trata-se de uma estratégia de segurança que envolve a especificação explícita e autorização de elementos ou ações que são considerados seguros, em vez de tentar identificar e bloquear elementos ou ações que são considerados inseguros. No contexto de prevenção de ataques de injeção de SQL, as Listas Brancas são usadas para determinar quais entradas ou valores são permitidos em vez de tentar filtrar ou bloquear entradas prejudiciais. As Listas Brancas exigem que todos os dados de entrada estejam alinhados com critérios específicos, como tipos de dados, formatos ou valores aceitáveis. Como resultado, possíveis invasores terão maior dificuldade para injetar comandos maliciosos, pois suas entradas provavelmente não estão na Lista Branca.

Em um aplicativo web, por exemplo, que tenha um campo de entrada em que os usuários devem inserir nomes de cidades. A Lista Branca poderia conter apenas nomes de cidades válidas, e qualquer entrada que não correspondesse a esses nomes seria rejeitada. Assim, se um usuário tentasse inserir '"; DROP TABLE usuarios --", essa entrada seria automaticamente rejeitada porque não corresponde a uma cidade válida na Lista Branca.

Além dessas técnicas citadas por Sadeghian e colegas, no site da CWE também é destacado como métodos de prevenção: o uso de bibliotecas e frameworks confiáveis, política de acesso mínimo ao banco de dados, restrição de informações em mensagens de erro, utilização de Firewalls de Aplicativos Web (WAF) e revisão/testes do sistema.

Bibliotecas ou frameworks confiáveis: Ao utilizar bibliotecas e frameworks, a quantidade de código personalizado que um desenvolvedor precisa escrever é reduzida. Isso, por sua vez, reduz a chance de erros de codificação que poderiam levar a vulnerabilidades de injeção de SQL. O uso de bibliotecas e frameworks confiáveis fornece, assim, uma camada de proteção e ajuda a automatizar a implementação de práticas de segurança recomendadas, tornando mais difícil para os desenvolvedores introduzirem vulnerabilidades de injeção de SQL em seus aplicativos.

Acesso Mínimo ao Banco de Dados: O princípio de "Acesso Mínimo ao Banco de Dados" refere-se a uma prática de segurança na qual as contas de usuário que interagem com um banco de dados são concedidas apenas às permissões estritamente necessárias para realizar suas funções específicas. Essa abordagem visa limitar o acesso e a capacidade de modificação do banco de dados somente ao que é essencial para o funcionamento adequado da aplicação, reduzindo assim a superfície de ataque e o risco de ataques, incluindo ataques de injeção de SQL. Nessa abordagem também é desaconselhável o uso de contas com privilégios de administrador (como root ou sa) para interações diárias com o banco de dados, a menos que seja absolutamente necessário. Essas contas de superusuário devem ser reservadas para tarefas de administração e manutenção e não devem ser usadas em operações regulares da aplicação.

Restrição de informações em mensagens de erro: é uma prática importante para prevenir ataques de injeção de SQL, uma vez que limita o conhecimento disponível para os atacantes, tornando mais difícil a exploração de vulnerabilidades. Mensagens de erro detalhadas podem revelar informações sensíveis sobre a estrutura do banco de dados, esquemas de tabelas, nomes de colunas e até mesmo a lógica de consulta SQL. Essas informações são extremamente úteis para um potencial invasor, pois fornecem insights valiosos sobre o ambiente do banco de dados. O site da CWE, nesse sentido, adiciona sobre as mensagens de erro:

> As mensagens precisam equilibrar-se entre ser muito crípticas (o que pode confundir os usuários) e ser muito detalhadas (o que pode revelar mais do que o desejado). As mensagens não devem revelar os métodos usados para determinar o erro. Atacantes podem utilizar informações detalhadas para aprimorar ou otimizar o ataque original, aumentando assim suas chances de sucesso. (Mitre, 2023, tradução minha).

Firewalls de Aplicativos Web (WAF): Os WAFs funcionam como uma barreira entre os usuários da web e os servidores de aplicativos, filtrando o tráfego e identificando e bloqueando atividades maliciosas antes que elas atinjam o aplicativo. Os WAFs são configurados com regras e políticas de segurança que definem o comportamento permitido e proibido. Eles podem detectar atividades que violam essas regras e identificar tentativas de injeção de SQL. Quando uma solicitação parece ser maliciosa, o WAF a bloqueia ou a redireciona, impedindo que a solicitação atinja o servidor do aplicativo. Além de regras personalizadas, muitos WAFs usam bancos de dados de assinaturas para identificar padrões conhecidos de ataques. Eles comparam os padrões no tráfego com as assinaturas conhecidas e bloqueiam solicitações que correspondem a ataques de injeção de SQL bem documentados. Além da prevenção de ataques, os WAFs geralmente registram todas as

atividades de tráfego, permitindo a análise e investigação de eventos de segurança. Isso é útil para a identificação de padrões de ataque, auditorias de segurança e conformidade regulatória. No entanto, é importante destacar que há limites para o que os WAFs conseguem detectar e impedir, nesse sentido, no site da CWE aparece a nota

> Um firewall de aplicativos pode não cobrir todos os vetores de entrada possíveis. Além disso, técnicas de ataque podem estar disponíveis para contornar o mecanismo de proteção, como o uso de entradas malformadas que ainda podem ser processadas pelo componente que recebe essas entradas. Dependendo da funcionalidade, um firewall de aplicativos pode inadvertidamente rejeitar ou modificar solicitações legítimas. Por fim, pode ser necessário algum esforço manual para personalização. (Mitre, 2023, tradução minha).

Revisão e Auditoria de Códigos: A incorporação da revisão de códigos, testes de segurança e auditorias de códigos é essencial como parte integrante de uma estratégia de prevenção contra diversos tipos de ataques, incluindo a injeção de SQL. Seja por desatenção ou desconhecimento, é possível que programadores cometam erros de segurança ao longo de suas carreiras, como falhas na correta validação de campos. Portanto, é crucial implementar mecanismos de detecção para identificar esses erros, garantindo que a segurança não dependa exclusivamente do nível alcançado por uma equipe de programadores na versão inicial de um software.

Ferramentas de Identificação de Vulnerabilidades de Injeção de SQL: Atualmente, os atacantes não se dedicam mais a realizar manualmente variações de comentários e inserções de código SQL em cada campo de entrada e URL. A automação desse tipo de ataque por meio de softwares especializados tornou-se uma prática comum, acelerando a identificação dessas vulnerabilidades de forma mais eficiente. Embora essas ferramentas possam ser usadas de maneira ilegal para perpetrar ataques, também desempenham um papel crucial como instrumentos de identificação de vulnerabilidades. Ao utilizá-las de forma ética, essas ferramentas contribuem para evitar que versões de software com tais falhas de segurança avancem nos testes de segurança e sejam implementadas.

Monitoramento de Logs: O monitoramento de logs é o processo de rastrear, analisar e registrar eventos e atividades do sistema, como logs de servidor web, logs de aplicativos, logs de segurança e outros registros de atividade. Essa prática ajuda a identificar e responder a eventos anômalos ou suspeitos que podem ser indicativos de tentativas de ataques, incluindo ataques de injeção de SQL. O monitoramento de logs em tempo real permite que as equipes de segurança reajam rapidamente a tentativas de injeção de SQL.

Alertas imediatos podem acionar ações de resposta, como o bloqueio de endereços IP ou a implementação de regras de firewall para evitar danos.

Em muitos casos, os ataques de injeção de SQL podem resultar em erros e exceções no aplicativo. O monitoramento de logs pode ajudar a identificar essas exceções e rastrear a origem dos erros. Isso é útil para entender a natureza de uma possível injeção de SQL e tomar medidas corretivas. Em caso de um ataque de injeção de SQL bem-sucedido, os logs podem fornecer evidências forenses cruciais que podem ser usadas na investigação do incidente, na identificação de sua origem e na mitigação de seus efeitos.

Criptografia: a criptografia, em si, não previne ataques de SQLi, mas pode diminuir a quantidade de informação vazada se um atacante conseguir acesso ao banco de dados. De acordo com Gilberto Sudre (2015), Vice-Presidente da Associação Nacional de Peritos em Computação Forense,

> “O ataque do SQL injection pode ser muito ampliado com os maus hábitos que alguns programadores (..) têm de armazenar, por exemplo, senhas em texto claro. Ou seja, se o cara tem acesso ao banco de dados (..) aumenta (..) o resultado do ataque, ele consegue ter acesso também às informações das tabelas e aí tem acesso a email, nome ou as informações pessoais, perfil do usuário e inclusive a senha.” (Sudre, 2015)

4. Resultados

A pesquisa (anexo 1) esteve disponível online no período de 25/10/2023 a 15/11/2023, e obteve a participação de 101 respondentes. No entanto, uma dessas pessoas indicou, na primeira pergunta, que nunca havia estudado qualquer tipo de programação. Por essa razão, suas respostas foram desconsideradas, uma vez que não se enquadraram no perfil desejado para a pesquisa. Assim, todas as análises subsequentes foram baseadas nas respostas de 100 participantes. Vale ressaltar que nem todas as perguntas eram obrigatórias, e em algumas delas, os respondentes podiam assinalar mais de uma alternativa.

Perfil dos respondentes

Até pelo perfil dos grupos em que a pesquisa foi postada, o que inclui vários grupos de alunos de cursos ligados à programação, vemos que boa parte daqueles que responderam ainda não concluíram o curso ligado a programação. Conforme pode ser visto na figura 2, 39% dos entrevistados começaram, mas não concluíram o ensino superior, 3% dos entrevistados não concluíram o ensino técnico e 5% não concluíram o curso livre, totalizando 47% daqueles que responderam o questionário. Esses dados apontam para uma um grande número de pessoas que ainda estão começando suas carreiras em uma área ligada à tecnologia.

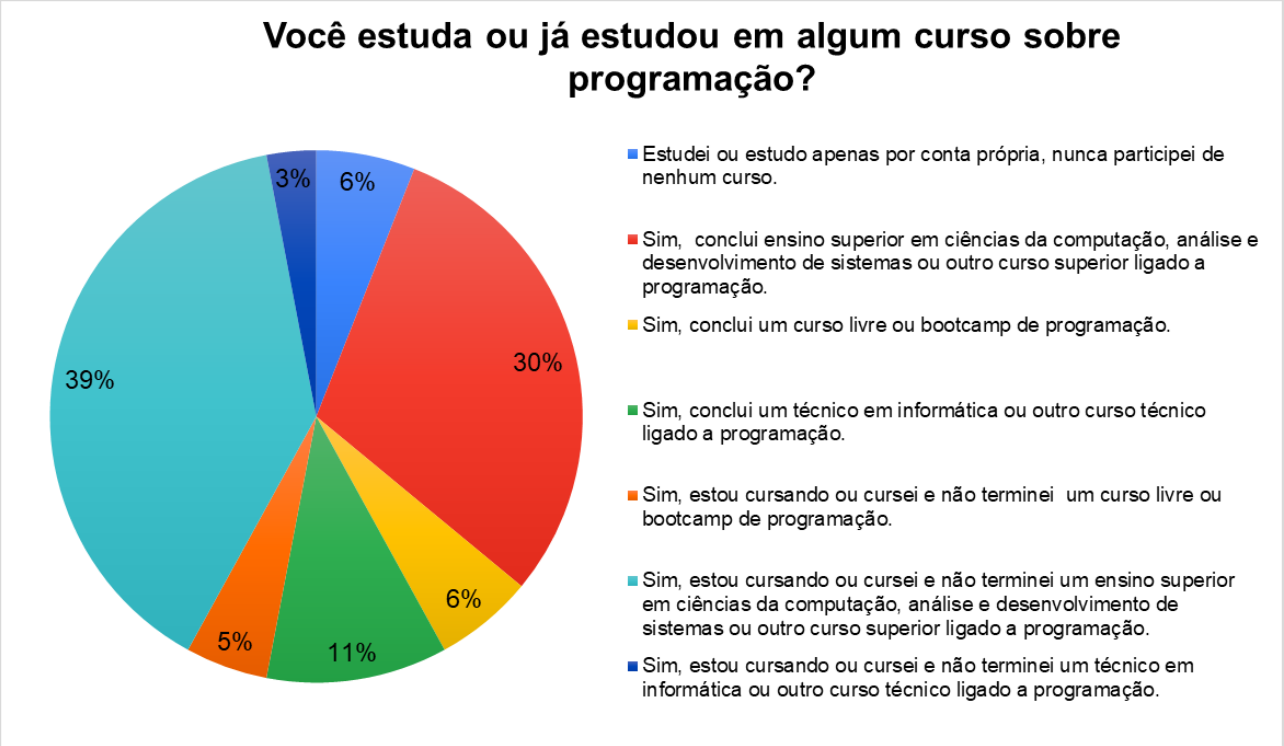

Figura 2 - Resultados da primeira questão da pesquisa.

Ainda sobre onde os participantes aprenderam programação, a instituição mais comum com 33 respostas foi a Fatec. Já as instituições que apareceram em 2 lugar como resposta mais comum, com apenas 7 respostas cada foram Estácio e Uninter. O resto das 48 respostas foram distribuídas em 38 instituições.Já em relação ao curso mais comum, 59% cursa ou cursou análise e desenvolvimento de sistemas (ADS). O fato da FATEC ser a instituição mais citada e o curso de ADS ser preponderante nas respostas pode ser explicado por eu estar me formando na Fatec e conhecer e ter mais acessos à grupos do curso de ADS da Fatec.

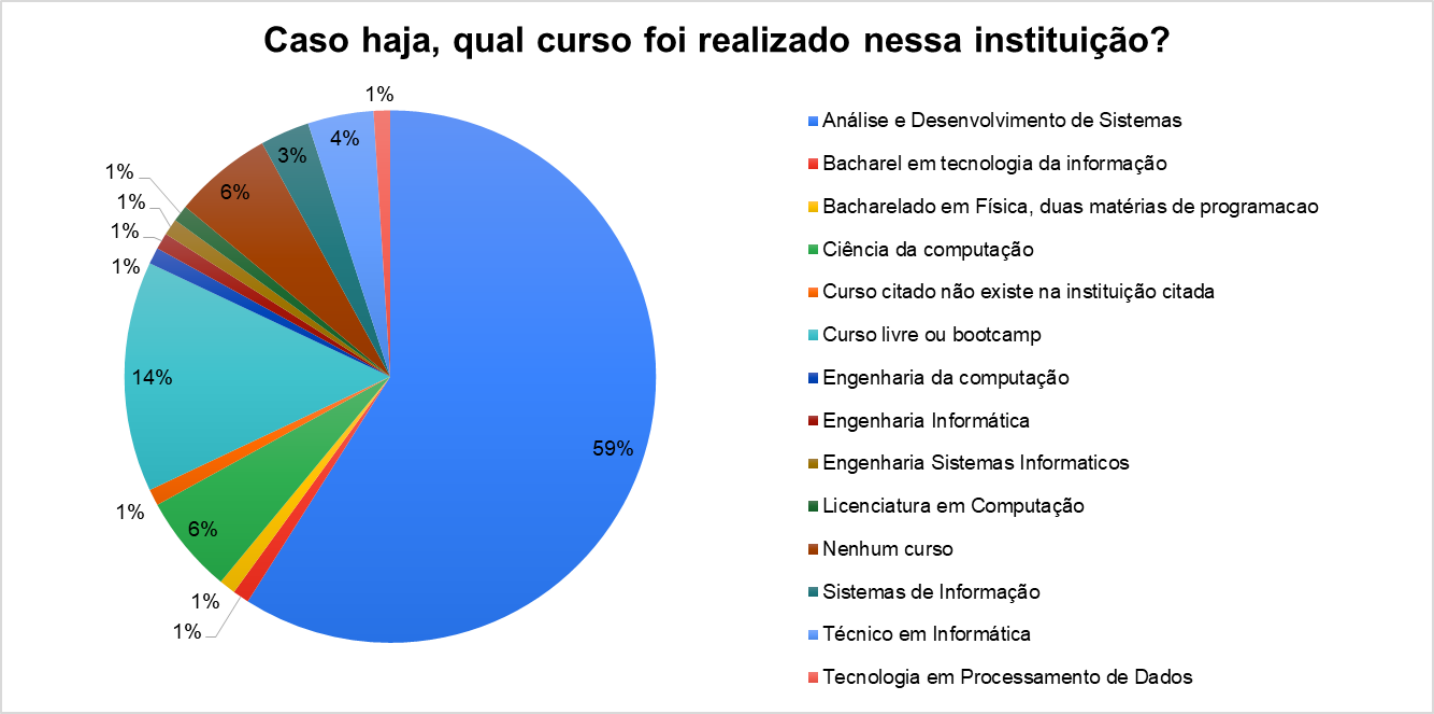

Figura 3 - Resultado da segunda questão do formulário.

Em relação à pergunta: “Atualmente, você tem algum trabalho ligado à área de tecnologia? Se sim, qual o seu cargo atual?” Vemos que a resposta mais comum, fora não ter um trabalho relacionado a tecnologia é ser desenvolvedor, com 32% das respostas. Se ainda juntarmos as 7 pessoas que responderam que em um cargo anterior já foram desenvolvedores com as 32 pessoas que responderam que são desenvolvedores, atualmente temos 39 pessoas e, portanto, 39% dos respondentes são ou já foram desenvolvedores. No entanto, nesses que responderam que trabalharam ou trabalham como desenvolvedor temos diferentes cargos e níveis de experiência, desde respostas como “Estagiário de Engenharia de Software” até respostas como “Analista desenvolvedor sênior”.

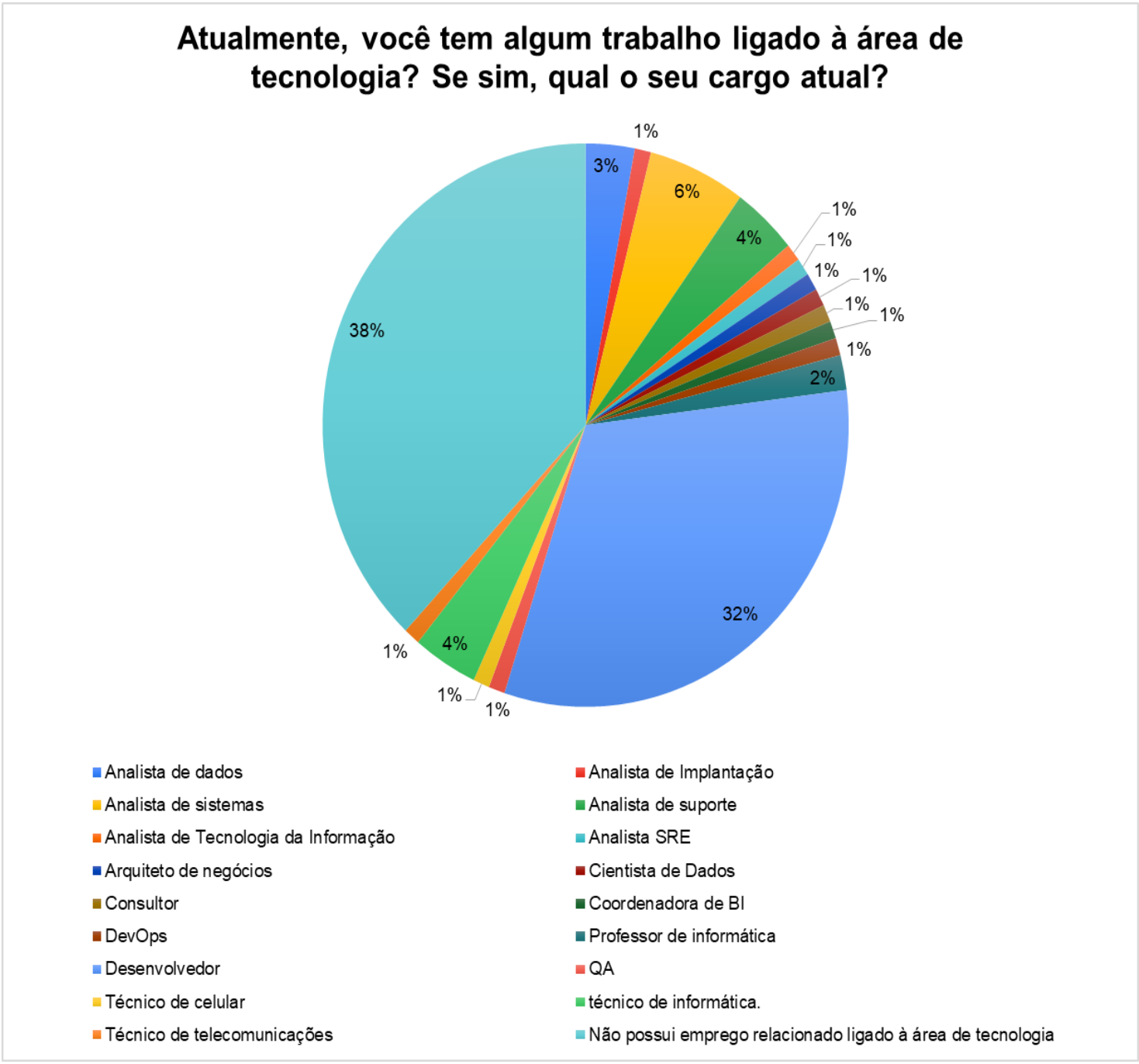

Figura 4 - Resultado da quarta questão do formulário.

Ainda em termos de perfil, vemos uma maioria que trabalha ou trabalhou em uma empresa de grande porte (mais de 100 colaboradores). Conforme pode ser visto na figura a seguir, 54% dos que já tiveram algum trabalho na área de tecnologia esse trabalho foi em uma empresa com mais de 100 colaboradores.

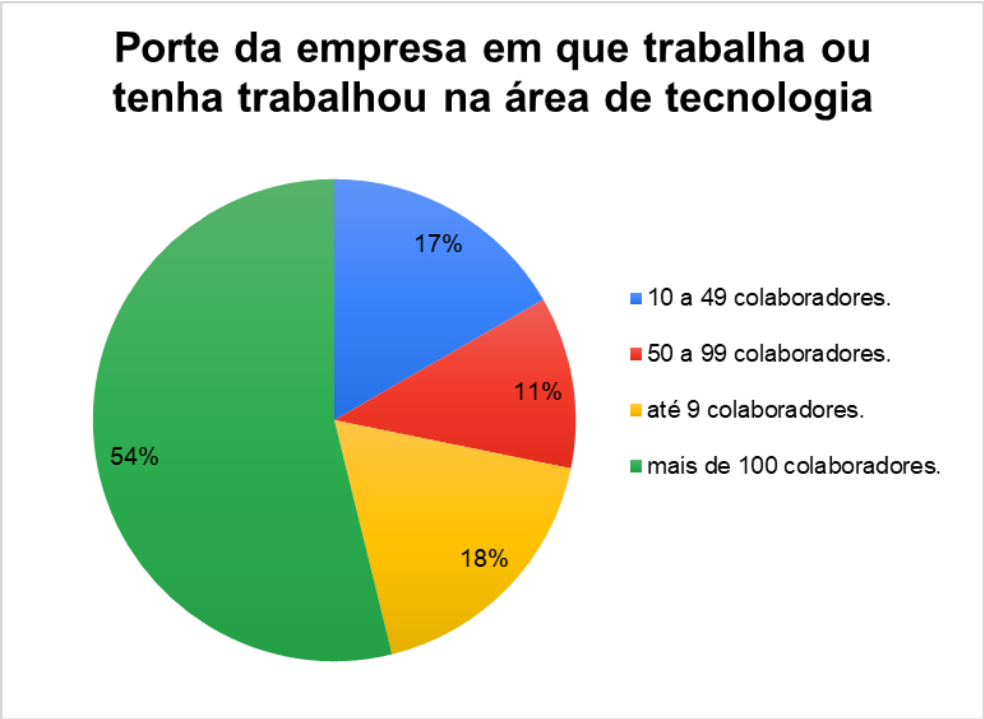

Figura 4 - Resultado da sexta questão do formulário.

Em relação a qual linguagem que os entrevistados programam, vemos que há uma grande variedade, mas que as mais usadas são SQL com 74 respostas, JavaScript com 69, Python com 62 e Java com 56, como pode ser visto na próxima imagem.

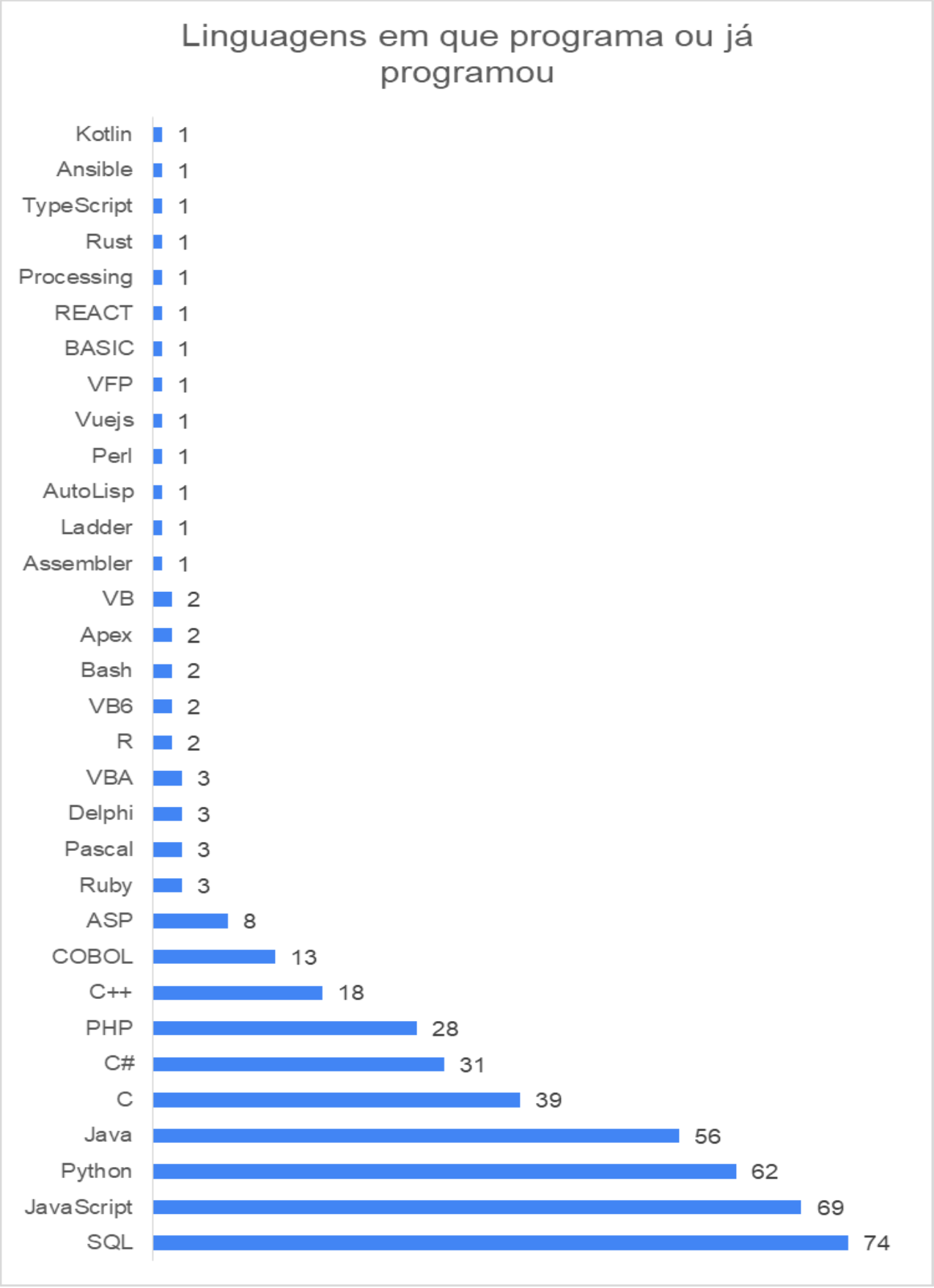

Figura 5 - Resultado da sétima questão do formulário

Nível de conhecimento sobre injeção de SQL

A oitava pergunta pede para os usuários que possuem algum nível de familiaridade com Python tentar identificar o erro de segurança em um pedaço de código. O erro está na parte “query = f"SELECT * FROM users WHERE username = '{username}' AND password = '{password}'"

cursor.execute(query)” em que os valores dos parâmetros da consulta SQL são inseridos diretamente na string de consulta, sem a devida validação ou tratamento, abrindo brecha assim para injeção de SQL. Para mitigar essa vulnerabilidade, é recomendado usar instruções parametrizadas ou consultas preparadas, entre outras ações como descrito na seção de prevenção desse trabalho.

Das pessoas que marcaram que programavam em Python, apenas 40% identificou corretamente o erro. Mostrando assim que a maior parte não tem familiaridade com as técnicas de prevenção a ataques de SQLi.

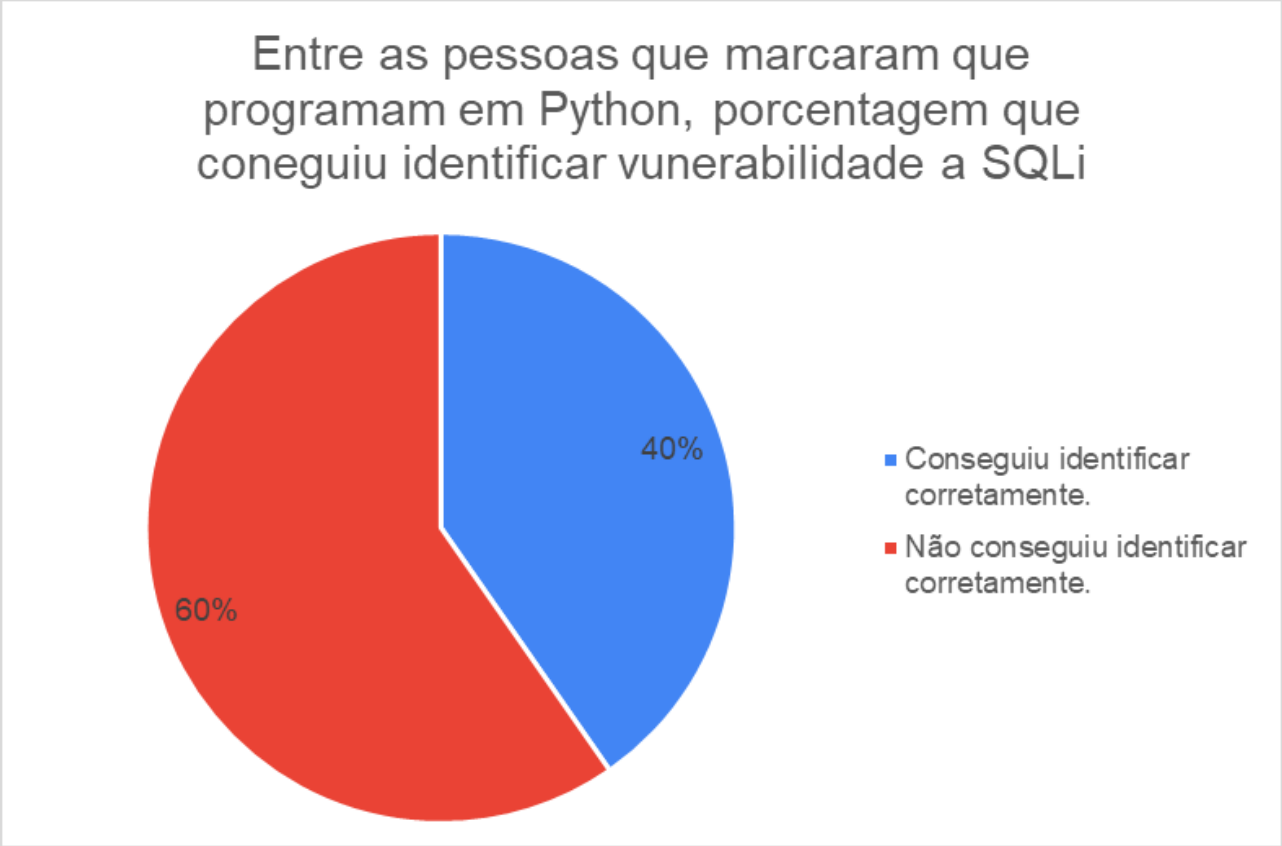

Figura 6 - Resultado da oitava questão do formulário.

A nona questão é parecida com a oitava, mas ao invés de um código em Python foi passado um código em Java. Nesse trecho que código em Java a consulta ao banco de dados não está parametrizada e aparece como “String query = "SELECT * FROM users

WHERE username = '" + username + "' AND password = '" + password + "'";" abrindo brecha para ataques de injeção de SQL.

Das pessoas que marcaram que programavam em Java apenas 39% conseguiu identificar esse erro corretamente. Mostrando que a maioria dessas pessoas não possuem domínio sobre técnicas de prevenção a ataques de SQLi.

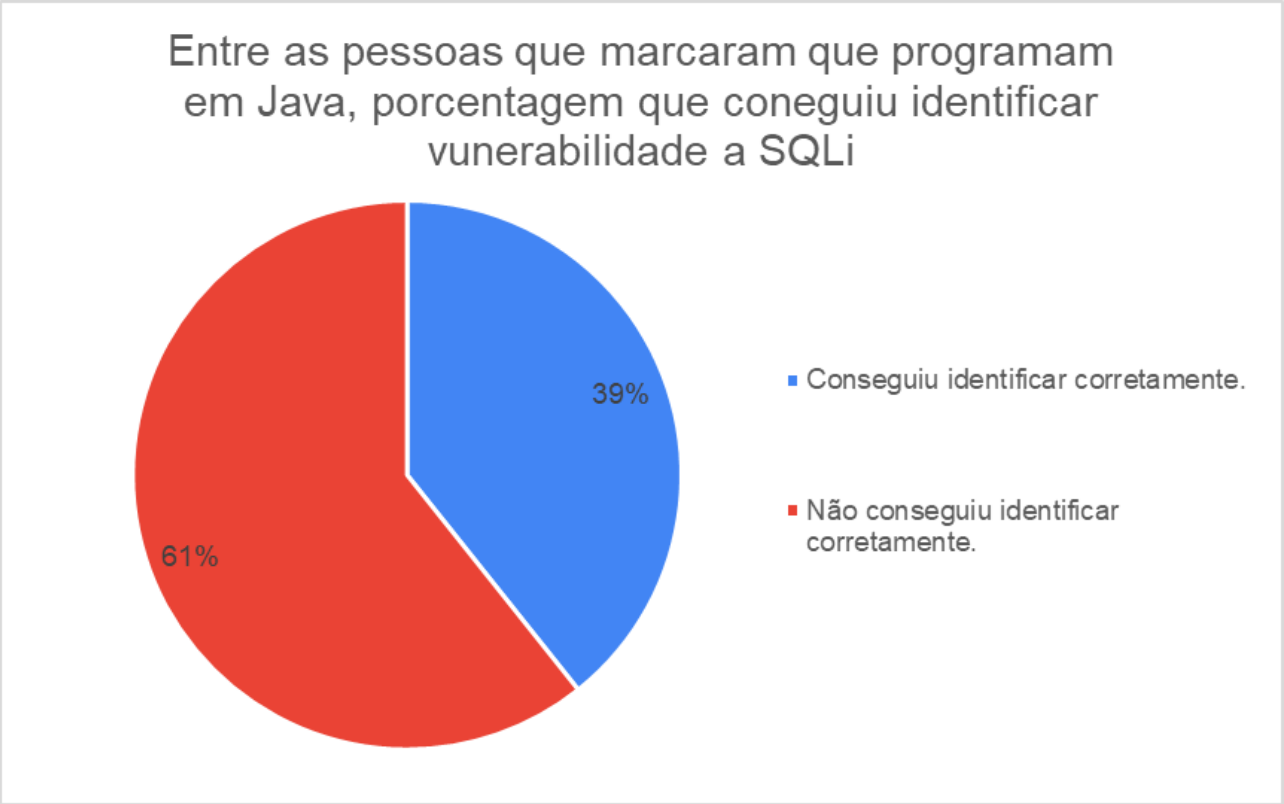

Figura 7 - Resultado da nona questão do formulário.

A décima questão pedia para que dessem uma breve definição de injeção de SQL. Todas as definições que citavam a possibilidade de injetar códigos de SQL em consultas não validadas/parametrizadas foram consideradas corretas, mesmo que com pequenos erros, como reduzir o local que esse ataque pode acontecer a apenas campos de formulários não tratados. Mesmo assim, apenas 41% conseguiu fornecer uma definição correta de SQLi.

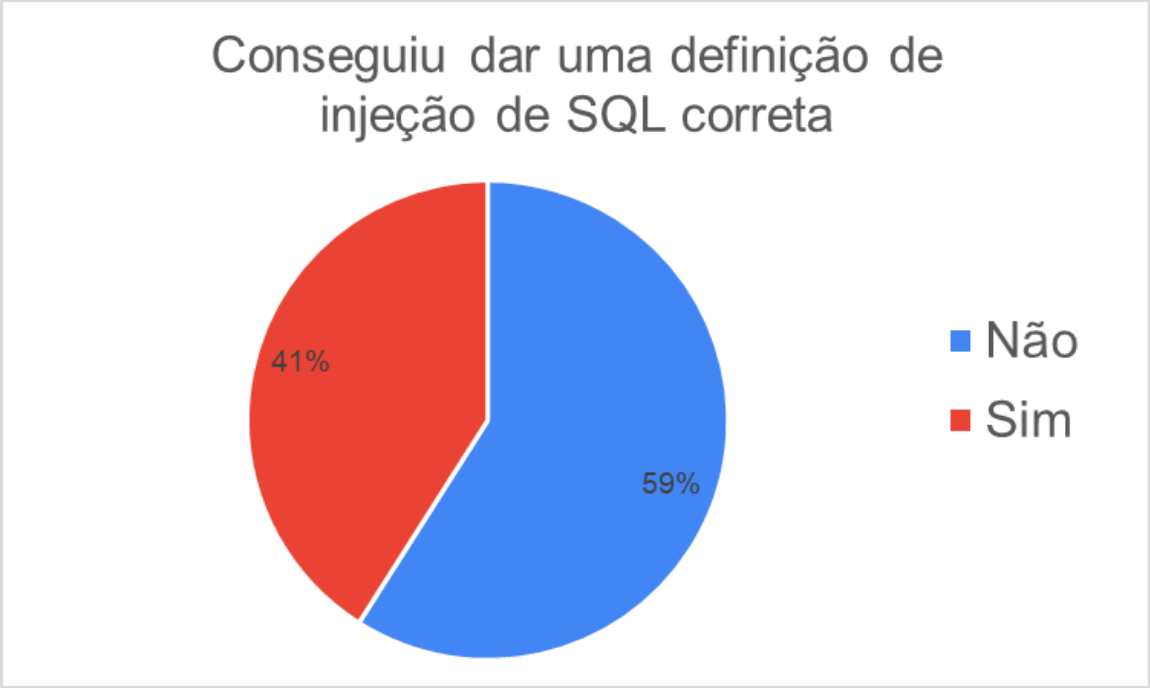

Figura 8 - Resultado da décima questão do formulário.

A décima primeira questão questionava onde os participantes ouviram falar sobre injeção de SQL. O local em que isso mais aconteceu foi em estudos por conta própria (44 respostas) e o local em que menos aconteceu foi no trabalho (27 respostas). Essa ordem de locais mais e menos citados continuou a mesma, considerando apenas as respostas daqueles que acertaram a definição de SQLi na questão anterior.

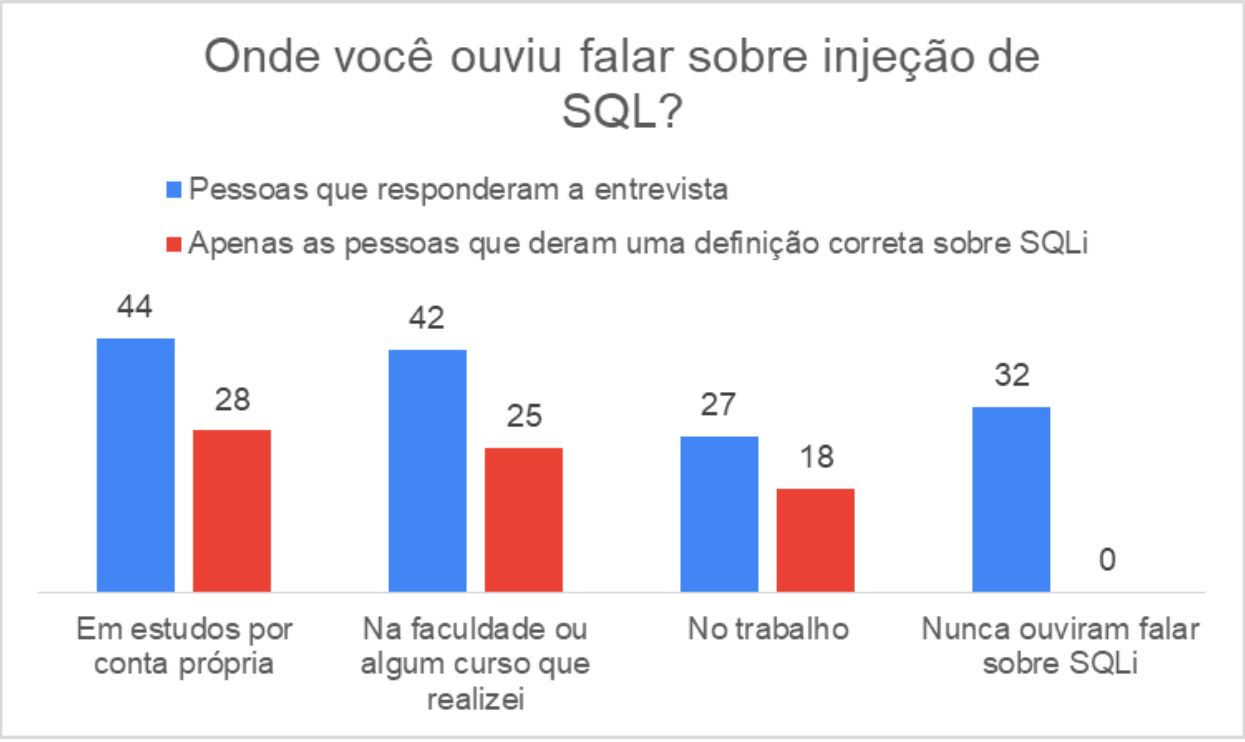

Figura 9 - Resultado da décima primeira questão do formulário.

A décima segunda pergunta diz respeito à identificação das possíveis estratégias de prevenção a ataques de SQL. Foi fornecido 9 estratégias e a pessoa precisava marcar quais se aplicavam na prevenção de ataques de SQLi. 7 dessas alternativas estavam corretas, são elas: validação e sanitização de entrada de dados, utilização de instruções preparadas (prepared statements), utilização de procedimentos armazenados (stored procedures), política de acesso mínimo necessário ao banco de dados, utilização de listas brancas (whitelisting), utilização de firewalls de aplicativos web (WAF), monitoramento de logs. Já duas alternativas não eram estratégias de prevenção de ataques de SQLi, são elas: utilização de “with (nolock)” em seleções no banco de dados e utilização de gerenciador de senhas.

Das alternativas corretas as mais selecionadas foram “Validação e sanitização de dados” (com 58 marcações), “Política de acesso mínimo necessário ao banco de dados” (41) e “Utilização de instruções preparadas (prepared statements)” (40). Já das alternativas incorretas a mais selecionada foi “Utilização de gerenciador de senhas” (com 10 marcações).

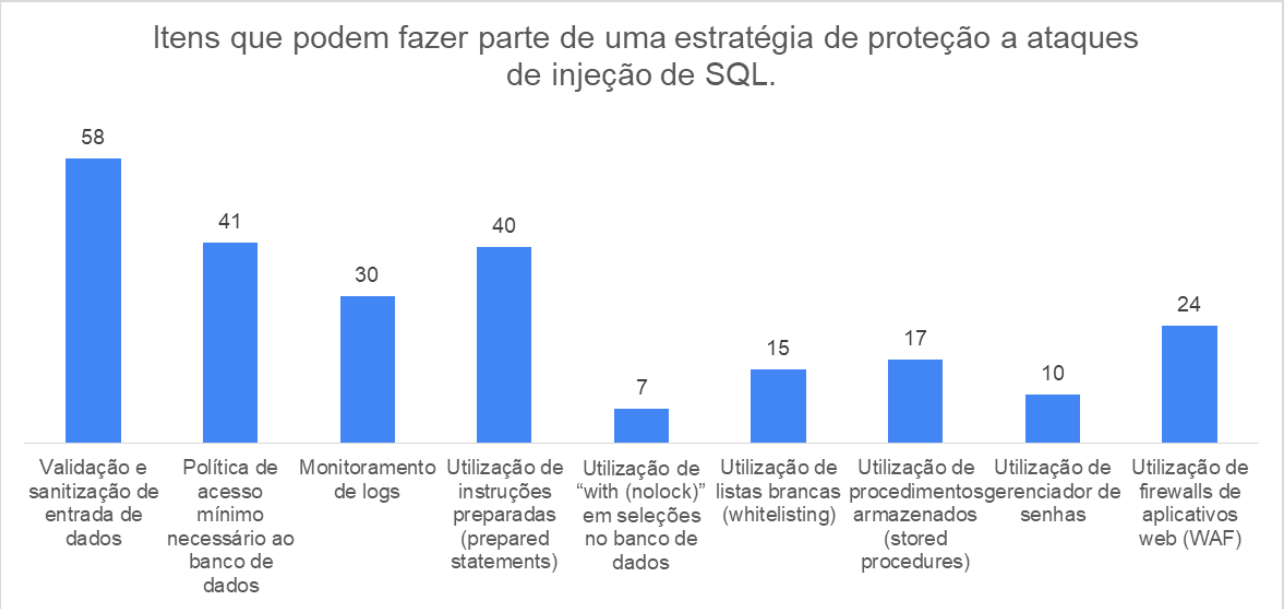

Figura 10 - Resultado da décima segunda questão do formulário.

Ainda nessa questão foi dado uma pontuação para cada pessoa. Foi adicionado 1 ponto para cada alternativa correta selecionada e foi retirado um ponto para cada alternativa incorreta selecionada. De modo que no mínimo uma pessoa poderia pontuar -2 e no máximo 7. Nesse esquema de pontuação vemos que a maioria pontuou de 0 a 3, mostrando baixo conhecimento sobre estratégias de prevenção a ataques de SQLi.

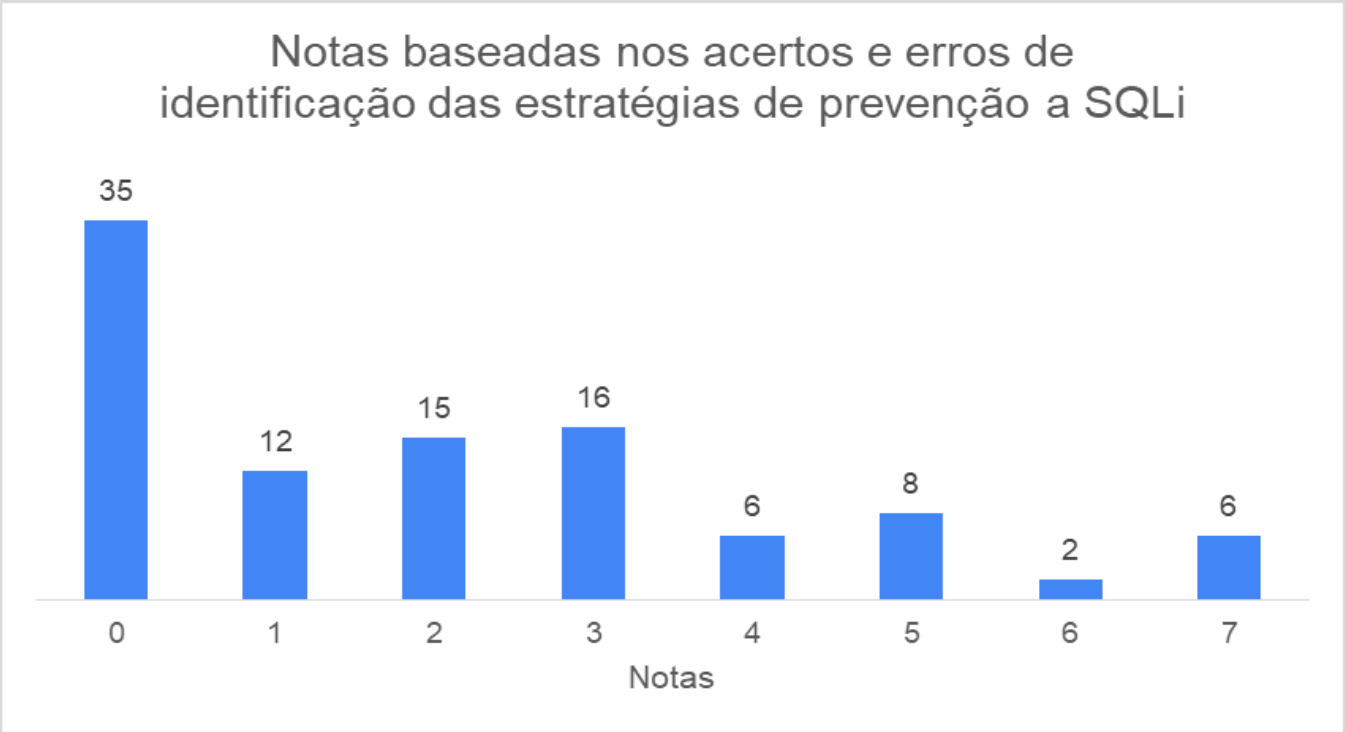

Figura 11 - Notas produzidas a partir dos resultados da décima segunda questão do formulário.

A décima terceira questiona se a pessoa que responde sabe como sanitizar campos e como usar prepared statements na linguagem que mais programa. Vendo os resultados a maioria ou não sabe (46%) ou acha que sim, mas não tem certeza (20%).

Figura 12 - Resultado da décima terceira questão do formulário.

Mesmo filtrando as respostas dessa terceira pergunta para apenas aqueles que antes responderam que trabalham ou já trabalharam como desenvolvedor, ainda assim a maior parte respondeu que não sabe (31%) ou acha que sim, mas não tem certeza (26%).

Figura 13 - Resultado da décima terceira questão do formulário, filtrado por apenas os participantes que já trabalharam como desenvolvedor.

5. Conclusão

Ao examinar a literatura existente sobre ataques de injeção de SQL, notadamente ao considerar o levantamento das 25 vulnerabilidades mais perigosas da CWE e a lista das 10 vulnerabilidades mais comuns da OWASP, fica evidente que os ataques de injeção de SQL permanecem uma ameaça significativa e amplamente prevalente. Essa constatação é crucial, uma vez que, quando bem-sucedidos, esses ataques podem acarretar sérias consequências para a segurança das aplicações.

Essa periculosidade deriva do fato de que os ataques de injeção de SQL envolvem a exploração de falhas que permitem o acesso e, potencialmente, a manipulação não autorizada de bancos de dados. Tal ação, por sua vez, resulta em vazamento de informações confidenciais e, em casos mais extremos, comprometimento integral dos sistemas. Portanto, é inegável que a persistência dessas vulnerabilidades representa uma grave lacuna na segurança que requer constante atenção e medidas de mitigação robustas.

Embora a literatura destaque a persistência e ameaça contínua dos ataques de injeção de SQL, é igualmente importante reconhecer que existem, na mesma literatura, descrições de técnicas eficazes de prevenção para mitigar essas ameaças. O conhecimento sobre como evitar ataques de injeção de SQL já está disponível, e as vulnerabilidades que potencialmente poderiam ser exploradas para tais ataques, conforme relatado no levantamento da OWASP, têm soluções conhecidas.

Portanto, o desafio enfrentado pela comunidade de tecnologia não reside tanto na criação de novos métodos para combater ataques de injeção de SQL, mas sim na ampla implementação das técnicas e boas práticas de prevenção já estabelecidas. Em outras palavras, o foco agora está em disseminar e aplicar efetivamente o conhecimento existente para fortalecer a segurança das aplicações, em vez de buscar constantemente novas abordagens para enfrentar esse tipo de ameaça.

No entanto, é fundamental destacar que o reconhecimento da eficácia das técnicas existentes não elimina a importância de possíveis avanços contínuos na área da segurança cibernética. A evolução constante de novas tecnologias pode, e provavelmente continuará, aperfeiçoar a prevenção de ataques de injeção de SQL. Por exemplo, o desenvolvimento de inteligências artificiais capazes de analisar e identificar vulnerabilidades de segurança em códigos é um progresso significativo que merece atenção.

Portanto, enquanto é essencial consolidar as medidas de segurança existentes, é igualmente valioso estar atento às inovações que podem contribuir para um ambiente digital mais seguro. A combinação de conhecimento consolidado com tecnologias emergentes pode representar um avanço significativo na defesa contra ataques de injeção de SQL e outras ameaças cibernéticas.

Além da grande necessidade de disseminação de métodos preventivos contra ataques de injeção de SQL, as organizações devem adotar estratégias que contemplem múltiplos pontos de defesa, sem depender exclusivamente do conhecimento individual dos programadores. Essa exigência torna-se evidente com os resultados da pesquisa realizada com profissionais envolvidos no desenvolvimento de software.

Notavelmente, ao abordar a competência dos profissionais, apesar de a pesquisa apresentar uma amostragem reduzida para ser conclusiva, ela destaca a existência de programadores com baixo nível de conhecimento sobre injeção de SQL e seus métodos de prevenção. Em todas as questões de conhecimento sobre SQLi e métodos de prevenção (questões 8, 9, 10, 12, 13) o número de participantes que mostraram não ter total domínio do assunto foi maior do que o número de participantes que mostraram ter domínio.

Sempre existirão programadores com lacunas de conhecimentos de segurança, sobretudo entre aqueles com menos experiência (o que representava uma parte significativa dos participantes), que carecem de conhecimento sobre a implementação de boas práticas de segurança no desenvolvimento de códigos. Por isso é importante que empresas e demais instituições forneçam informações de segurança para seus colaboradores e que existam outras implementações de segurança como processos de revisão de código, firewall e outras implementações para impedir que esse ataque aconteça.

Referências

OWASP. (2021). A3:2021 - Injection. Disponível em: https://owasp.org/Top10/A03_2021-Injection/. Acesso em: 21 de outubro de 2023.

OWASP. (2021). OWASP Top 10 Application Security Risks - 2021 Labs Data. Disponível em: https://github.com/OWASP/Top10/blob/master/2021/Data/owasp_top10_appsec_labs_2021_submission.csv. Acesso em: 21 de setembro de 2023.

Cox, J. (2015). The History of SQL Injection, the Hack That Will Never Go Away. Vice. Disponível em: https://www.vice.com/en/article/aekzez/the-history-of-sql-injection-the-hack-that-will-never-go-away. Acesso em:11/11/2023

De Souza, R. (2021). Invasor volta a atacar DataSUS e zomba da segurança do site do governo. Canaltech. Disponível em: https://canaltech.com.br/seguranca/invasor-volta-a-atacar-datasus-e-zomba-da-seguranca-do-site-do-governo-179179/. Acesso em: 12 de novembro de 2023.

MITRE. (s.d.). CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'). Disponível em: https://cwe.mitre.org/data/definitions/89.html. Acesso em: 20 de outubro de 2023.

MITRE. (s.d.). CWE List. Disponível em: https://cwe.mitre.org/data/index.html. Acesso em: 20 de outubro de 2023.

MITRE. (2023). CWE Top 25 Most Dangerous Software Weaknesses - 2023. Disponível em: https://cwe.mitre.org/top25/archive/2023/2023_methodology.html. Acesso em: 20 de outubro de 2023.

Boyd, S. W., & Keromytis, A. D. (2004). SQLrand: Preventing SQL Injection Attacks. Em S. W. Boyd & A. D. Keromytis (Eds.), Lecture Notes in Computer Science (pp. 292–302). DOI: 10.1007/978-3-540-24852-1_21

Sadeghian, A., Zamani, M., & Manaf, A. A. (2013). A Taxonomy of SQL Injection Detection and Prevention Techniques. In Proceedings of 2013 International Conference on Informatics and Creative Multimedia (ICICM) (pp. 53-56). IEEE. DOI: 10.1109/ICICM.2013.6750343

Sharma, K., & Bhatt, S. (2019). SQL injection attacks - a systematic review. International Journal of Information and Computer Security, 11(4/5), 493-511. DOI: 10.1504/IJICS.2019.101560

Diallo, A. K., & Pathan, A. K. (2011). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In Proceedings of 2011 IEEE 15th International Symposium on Consumer Electronics (ISCE) (pp. 468-471). IEEE. DOI: 10.1109/ISCE.2011.5973861

SecurityCast. [SecurityCast] WebCast #23 - SQL Injection. 2015. Realizado por Alcyon Junior, Alberto J. Azevedo, Gilberto Sudre e Gustavo Martinelli. Disponível em: <https://www.youtube.com/watch?v=WFpklonMxuQ\>. Acesso em: 12 de novembro de 2023.

Elshazly, K. et al. A Survey of SQL Injection Attack Detection and Prevention. *Journal of Computer and Communications*, [S.l.], v. 2, p. 1-9, jun. 2014. SciRes. Disponível em: http://www.scirp.org/journal/jcc. DOI: http://dx.doi.org/10.4236/jcc.2014.28001. Acesso em: 20 de outubro de 2023.

ALWAN, Zainab S.; YOUNIS, Manal F. Detection and prevention of SQL injection attack: a survey. International Journal of Computer Science and Mobile Computing, v. 6, n. 8, p. 5-17, 2017.

SINGH, Jai Puneet. Analysis of SQL injection detection techniques. arXiv preprint arXiv:1605.02796, 2016.

Anexo 1

Questionário sobre injeção de SQL

As respostas deste formulário serão utilizadas para o trabalho de conclusão de curso de Gabriela Whitaker Visani. São apenas 13 perguntas. Não serão comercializados, de nenhuma forma, os dados.
Peço que responda o questionário apenas uma vez.

1) Você estuda ou já estudou em algum curso sobre programação?
- Estudei ou estudo por conta própria, nunca participei de nenhum curso.
- Sim, estou cursando ou cursei e não terminei um curso livre ou bootcamp de programação.
- Sim, conclui um curso livre ou bootcamp de programação.
- Sim, estou cursando ou cursei e não terminei um técnico em informática ou outro curso técnico ligado a programação.
- Sim, conclui um técnico em informática ou outro curso técnico ligado a programação.
- Sim, estou cursando ou cursei e não terminei um ensino superior em ciências da computação, análise e desenvolvimento de sistemas ou outro curso superior ligado a programação.
- Sim, conclui ensino superior em ciências da computação, análise e desenvolvimento de sistemas ou outro curso superior ligado a programação.
- Nunca estudei nada relacionado a programação.

2) Caso haja, qual a instituição de ensino que você estuda ou estudou seu curso relacionado a programação?

3) Caso haja, qual curso foi realizado nessa instituição?

4) Atualmente, você tem algum trabalho ligado à área de tecnologia? Se sim, qual o seu cargo atual? Se não, deixe em branco.

5) Apenas caso não tenha respondido a resposta anterior: No passado, você já teve algum trabalho ligado a uma área de tecnologia? Se sim, qual o seu cargo? Se não, deixe em branco.

6) Caso trabalhe ou tenha trabalhado na área de tecnologia, o porte da empresa em que trabalha ou trabalhou era de:

- até 9 colaboradores
- 10 a 49 colaboradores
- 50 a 99 colaboradores
- mais de 100 colaboradores

7) Você programou ou já programou em quais linguagens?
   JavaScript
   SQL
   Javal
   Python
   C#
   PHP
   C++
   C
   ASP
   COBOL
   Outros (quais?)

8) Caso você tenha alguma familiaridade com Python, você identifica algum erro de segurança (não relacionado à falta de criptografia) no seguinte trecho de código de login?

```
import sqlite3

def login_sistema(username, password):
    conn = sqlite3.connect("database.db")
    cursor = conn.cursor()
```

```
    query = f"SELECT * FROM users WHERE username = '{username}' AND password =
'{password}'"
    cursor.execute(query)

    user = cursor.fetchone()
    conn.close()

    return user

username = input("Digite seu nome de usuário: ")
password = input("Digite sua senha: ")

user = login_sistema(username, password)
```

9) Caso tenha algum nível de familiaridade com Java, você consegue identificar mais algum erro de segurança no seguinte trecho código de login, que não esteja relacionado à falta de criptografia nem ao modo como está estabelecida a conexão com o banco de dados?

```
public class LoginPage {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Usuário: ");
        String username = scanner.nextLine();

        System.out.print("Senha: ");
        String password = scanner.nextLine();



        String jdbcURL = "jdbc:mysql://localhost:3306/mydb";
        String dbUser = "seu_usuario";
        String dbPassword = "sua_senha";

        try {
```

```
        Connection connection = DriverManager.getConnection(jdbcURL, dbUser,
dbPassword);

        String query = "SELECT * FROM users WHERE username = '" + username + "'
AND password = '" + password + "'";
        Statement statement = connection.createStatement();
        ResultSet resultSet = statement.executeQuery(query);

        if (resultSet.next()) {
            System.out.println("Login bem-sucedido!");
        } else {
            System.out.println("Falha no login. Tente novamente.");
        }

        resultSet.close();
        statement.close();
        connection.close();
      } catch (SQLException e) {
        e.printStackTrace();
      }
    }
}
```

10) Você sabe o que é injeção de SQL? Se sim, descreva brevemente. Se não, deixe em branco.

11) Se sim, onde você ouviu falar sobre injeção de SQL? (permitido selecionar mais de uma alternativa)

- Na faculdade ou algum curso que realizei
- No trabalho
- Em estudos por conta própria

12) Assinale os itens que podem fazer parte de uma estratégia de proteção a ataques de injeção de SQL. (permitido selecionar mais de uma alternativa)

- Validação e sanitização de entrada de dados.

- Utilização de gerenciador de senhas.
- Utilização de instruções preparadas (prepared statements).
- Utilização de procedimentos armazenados (stored procedures).
- Utilização de “with (nolock)” em seleções no banco de dados.
- Política de acesso mínimo necessário ao banco de dados.
- Utilização de listas brancas (whitelisting).
- Utilização de firewalls de aplicativos web (WAF).
- Monitoramento de logs.

13) Você diria que sabe como sanitizar campos e como usar prepared statements na linguagem que você mais programa?

- Não sei.
- Sei, mas não com muita confiança.
- Sei e estou acostumado a realizar isso.