CENTRO UNIVERSITÁRIO BRASILEIRO – UNIBRA
CURSO DE GRADUAÇÃO TECNOLÓGO EM REDES DE COMPUTADORES

DIEGO RONYE DA SILVA
THAÍS FERREIRA DA SILVA

VULNERABILIDADES COM SQL INJECTION

RECIFE
2021

DIEGO RONYE DA SILVA

THAÍS FERREIRA DA SILVA

VULNERABILIDADES COM SQL INJECTION

Artigo apresentado ao Centro Universitário Brasileiro – UNIBRA, como requisito parcial para obtenção do título de tecnólogo em Redes de Computadores.

Professor Orientador: Msc Ameliara Freire Santos de Miranda

RECIFE

2021

Ficha catalográfica elaborada pela
bibliotecária: Dayane Apolinário, CRB4- 2338/ O.

S586v Silva, Diego Ronye da

Vulnerabilidades com SQL Injection. / Diego Ronye da Silva, Thaís Ferreira da Silva. - Recife: O Autor, 2021.

28 p.

Orientador(a): Diego Leonel Alves de Sá.

Trabalho de Conclusão de Curso (Graduação) - Centro Universitário Brasileiro – UNIBRA. Tecnólogo em Redes de Computadores, 2021.

Inclui Referências.

1. Injeção de SQL. 2. Redes de internet. 3. Aplicações web. 4. Vulnerabilidades. I. Silva, Thaís Ferreira da. II. Centro Universitário Brasileiro - UNIBRA. III. Título.

CDU: 004

Dedicamos esse trabalho a nossa família, mestres e amigos.

AGRADECIMENTOS

Agradeço à DEUS, pela força diária e saúde para finalizar essa jornada de conclusão de curso e superar as minhas dificuldades. E aos colegas que se empenharam na realização desse trabalho.

“Ninguém ignora tudo. Ninguém sabe de tudo.
Todos nós sabemos alguma coisa. Todos
nós ignoramos alguma coisa. Por isso
Aprendemos sempre.”
(Paulo Freire)

SUMÁRIO

Resumo

Diante de um cenário atual, em meio a tantos dispositivos conectados à rede, numerosos acessos a aplicações web, usuários utilizam sistemas que podem ser acessados a partir de qualquer lugar do planeta para realizar tarefas comuns como pagamentos de contas, compras de passagens aéreas, transações bancárias entre outras atividades, a segurança é um fator importante.

Entretanto, na mesma medida que os usuários se sentem seguros nas plataformas digitais, há um crescimento exponencial no acesso a essas plataformas e as mesmas quando surgem vulnerabilidades e métodos de ataques. Dentre eles existe um que é o mais conhecido como injeção de *SQL Injection* ou código de injeção, no qual consiste em fornecer um caminho para obter acesso a informações não autorizada.

Utilizando-se de técnicas de injeção de *SQL*, o invasor poderá obter acesso a informações confidenciais, modificar e apagar registros do banco de dados. Realizando violações de perfis de usuários, obtenções de senhas, detalhes de cartão de crédito ou informações de dados pessoais sigilosos que foram obtidas através da injeção de código *SQL*, acarretando prejuízos a pessoas e empresas cujos dados foram comprometidos.

Veremos de forma sucinta, alguns comandos utilizados para realizar uma consulta *SQL* e algumas brechas exploradas por invasores para se ter acesso a dados sigilosos. Será mostrado boas práticas aqui expostas no combate a esse tipo de ataque.

Palavras-chave: Injeção de SQL. Rede de Internet. Aplicações Web. Vulnerabilidades.

Abstract

Faced with a current scenario, in the midst of so many devices connected to the network, numerous access to web applications, users use systems that can be accessed from anywhere on the planet to perform common tasks such as bill payments, airline ticket purchases, transactions banking among other activities, security is an important factor.

However, as users feel secure on digital platforms, there is an exponential growth in access to these platforms and the same when vulnerabilities and attack methods emerge. Among them there is one that is better known as SQL Injection or code injection, which consists of providing a way to gain access to unauthorized information.

Using SQL injection techniques, an attacker can gain access to sensitive information, modify and delete records from the database. Performing violations of user profiles, obtaining passwords, credit card details or sensitive personal data information that was obtained through the injection of SQL code, resulting in damages to people and companies whose data was compromised.

We will see briefly, some commands used to perform an SQL query and some loopholes exploited by intruders to gain access to sensitive data. It will be shown good practices exposed here in combating this type of attack.

Keywords: SQL Injection. Internet network. Web Applications. Vulnerabilities.

1 INTRODUÇÃO

As vulnerabilidades são falhas que, por si só, não provocam incidentes, pois são elementos passivos que dependem de um agente causador favorável que as explore tornando-as ameaças para a segurança da organização. Vulnerabilidade pode ser descrita como fragilidade de um ativo ou grupo de ativos, com possibilidade de ser explorada por uma ou mais ameaças. Segundo Sêmola (2012, p.47),

> São fragilidades presentes ou associados a ativos que manipulam e/ou processam informações que, ao serem exploradas por ameaças, permitem a ocorrência de um incidente de segurança, afetando negativamente um ou mais princípios de segurança da informação.

A linguagem *SQL*, definida como uma linguagem de pesquisa declarativa padrão para banco de dados relacional, sendo utilizada para realizar busca, manipulação e visualizar dados.

De acordo com Posada (2002):

> Structured Query Language (SQL), é uma linguagem textual usada para interagir com o banco de dados. Existem muitas variedades de SQL; a maioria dos dialetos de uso comum no momento são vagamente baseados em SQL-92, o padrão ANSI mais recente. A típica unidade de execução do SQL é a 'consulta', que é uma coleção de instruções que normalmente retornar um único 'conjunto de resultados'. As instruções SQL podem modificar a estrutura dos bancos de dados (usando declarações de linguagem de definição de dados, ou 'DDL') e manipular o conteúdo dos bancos de dados (usando instruções de linguagem de manipulação de dados ou 'DML').

Percebe a segurança da informação como um componente intrínseco ao uso dos computadores e a considera como uma meta a ser atingida para proteger os sistemas computacionais contra ameaças à confidencialidade, à integridade e à disponibilidade (SUMMERS, p.21).

A maioria das configurações de injeção de *SQL* pode ser evitada usando consultas parametrizadas (também conhecido como instruções preparadas) em vez de concatenação de *string* dentro da consulta (PORTSWIGGER, 2021).

1.1.1 Motivação

O tema que será apresentado no decorrer dos capítulos, ressalta a importância de se prevenir de ataques de injeção *SQL* e a segurança de dados de usuários. O aumento exponencial de invasões em aplicações *web* e a fragilidade de dados na internet ao passar dos anos (CERT.BR, 2020). Vem sendo um problema que se aplica no cenário global. Proporcionando uma amostra de riscos e falhas de ferramentas no uso, que trará algumas consequências expostas à segurança da informação.

1.1.2 Problemáticas

Diante do tema abordado, a seguir será apresentado algumas falhas de segurança em aplicações.

- Falta de controle e autenticidade na consulta do banco de dados.
- Não realizar gerenciamento de acesso em consultas *SQL*.
- Não estabelecer parâmetros para consulta em *SQL*.
- Más instruções de consulta *SQL*.

1.1.3 Objetivos Gerais

O presente trabalho tem o objetivo geral, reconhecer as vulnerabilidades, analisar e impedir os ataques. Comprovar a má codificação da linguagem *SQL*, utilizada por invasores, demonstrando como essas falhas são exploradas pela técnica *SQL injection*.

1.1.4 Objetivos Específicos

Como objetivo específico desse trabalho temos:

- Definir melhorias de segurança de uma consulta de banco de dados.
- Minimizar a exploração de vulnerabilidades.
- Exploração das principais vulnerabilidades em aplicações *web*.
- Utilização de consultas parametrizadas em *SQL*.

1.1.5 Organização do Trabalho

A estrutura da pesquisa apresentada contém 5 capítulos, conforme a lista a seguir:

- O primeiro capítulo apresenta a introdução e descreve a motivação desse trabalho, 1.1.2 problemáticas, 1.1.3 objetivos gerais e 1.1.4 objetivos específicos.
- O segundo capítulo aborda o referencial teórico, 2.1.1 vulnerabilidades, 2.1.2 principais vulnerabilidades *web,* 2.2 Linguagem *SQL,* 2.3 *SQL Injection* e 2.4 métodos de prevenção.
- Capítulo 3 descreve o procedimento metodológico utilizado neste trabalho.
- Capítulo 4 mostra os resultados do trabalho.
- Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 REFERENCIAL TEÓRICO

2.1.1 Vulnerabilidades

Software tem sido amplamente utilizado por invasores, para roubo de informações confidenciais e invasões de redes corporativas.

Vulnerabilidades são más codificações de várias origens, conforme (SÊMOLA, 2003, apud SOUZA, 2007, p.24). Existem centenas de vulnerabilidades, e incontáveis maneiras de explorá-las. Elas podem ser mal implementadas por desenvolvedores de software de empresas, ou até mesmo por falta de atualização do sistema. Identificar as vulnerabilidades e eliminá-las de forma recorrente é a garantia de uma infraestrutura segura e eficiente.

2.1.2 Principais vulnerabilidades web

Na sessão 2.1.3 serão abordadas as principais vulnerabilidades *web* de acordo com *OWASP* (*Open Web Application Security Project*), que correspondem aos invasores que se aproveitam dessas vulnerabilidades, que geram brechas, para invadir os sistemas.

2.1.3 *Cross-site scripting* (XSS)

É uma vulnerabilidade que permite o atacante executar *scripts* maliciosos no navegador do usuário. O ataque ocorre através de *tags* pela entrada de dados de aplicação vulnerável.

2.1.4 Manipulação de dados ocultos

Software vulnerável que permite acesso indevido quando dados ocultos são manipulados indevidamente.

2.1.5 Falha ao restringir acesso a URL ou funcionalidade

Essa vulnerabilidade ocorre quando as verificações de controle de acesso não são executadas e a aplicação não restringe adequadamente suas áreas restritas.

2.1.6 Armazenamento inseguro de criptografia

Dados sensíveis que deveriam ser armazenados de forma criptografada e estão em texto livre ou com uma criptografia inadequada.

2.1.7 Comunicação insegura

Aplicação que trafega dados sensíveis através de canais inseguros.

2.1.8 Falha da especificação de requisitos

Na qual requisitos de software não são identificados de forma precisa, devido a falha de especificação de controles de segurança.

2.1.9 Injeção de comandos

Geralmente é usada como método de *hacking*, tem como um dos pontos mais comuns o formulário, que que em páginas de *web* podem converter uma entrada de exibição para outros usuários, podendo deixar um *script* malicioso em comentários de um site.

2.1.10 Processo inadequado de cadastro de usuários

Cadastro que deve seguir algumas recomendações de segurança, para não haver riscos de expor a aplicação a diversos incidentes.

2.1.11 Quebra de autenticação e gerenciamento de sessão

São mecanismos de autenticação que frequentemente tem falhas de funcionalidades, as aplicações vulneráveis permitem burlar o processo de autenticação através de gestão fraca de sessão ou procedimentos inseguros, podendo permitir que contas privilegiadas sejam alvos frequentes.

2.2 LINGUAGEM SQL

O *SQL* é uma linguagem padrão para trabalhar com banco de dados relacionais. A linguagem *SQL* é usada para realizar funções como inserir dados em um banco de dados, recuperar, excluir e outras ações similares (PRESTON, 2015). É uma linguagem flexível que permite não somente a manipulação, mas também a definição da estrutura e também das regras e das restrições da integridade. Podemos dizer que os comandos *SQL* são as estruturas dessa linguagem, vamos mostrar quais são os comandos mais básicos do *SQL* e para que eles servem, além de dar exemplos práticos da aplicação de cada um deles.

2.2.1 Os principais comandos SQL

Mostraremos quais os comandos mais básicos do *SQL* e para que eles servem, além de dar exemplos práticos da aplicação de cada um deles.

- INSERT: adicionando registros a uma tabela;
- UPDATE: atualizando os registros já inserido;
- DELETE: excluindo registros de uma tabela;
- SELECT: retomando registros de uma tabela;
- CREATE: criando novas tabelas em um banco de dados;
- ALTER: alterando uma tabela já criada;
- DROP: excluindo uma tabela do banco de dados;
- GRANT: permitindo acesso a objetos do banco de dados;
- REVOKE: removendo o acesso a objetos do banco de dados;
- DENY: bloqueando o acesso para objetos e usuários específicos.

2.2.2 Consultando e modificando os registros da tabela

O *DML* (*Data Manipulation Language*) é um subconjunto de linguagem *SQL* que comporta os comandos utilizados para manipular diretamente os dados. Ou seja, por meio deles, é possível realizar ações como inserir informações em uma tabela, realizar consultas, deletar e alterar os registros.

- INSERT: adicionando registros a uma tabela.

```
INSERT into estudantes (id, nome, curso) values (23, 'Rafael', 'Desenvolvimento de
Software');
```

- UPDATE: atualizando os registros já inseridos.

```
UPDATE estudantes SET nome = 'Rafael Rodrigues Maia' WHERE id = 23;
```

- DELETE: excluindo registros de uma tabela.

```
DELETE FROM estudantes;
```

- SELECT: retomando registros de uma tabela.

```
SELECT * FROM estudantes;
```

2.2.3 Criando, alterando e excluindo tabelas de registros

Os comandos do tipo *DDL* (*Data Definition Language*) são utilizados para gerenciar a estrutura do banco de dados. Sendo assim, eles nos permitem alterar a estrutura de um objeto do banco, além de serem usados para criar e excluir tabelas.

- CREATE: criando novas tabelas em um banco de dados.

```
CREATE DATABASE escola;
```

- ALTER: alterando uma tabela já criada.

```
ALTER TABLE estudantes ADD idade INT;
```

- DROP: excluindo uma tabela do banco de dados.

```
DROP TABLE estudantes;
```

2.2.4 Definindo permissões aos objetos do banco de dados

Entre os comandos *SQL*, também existe os que fazem parte do subconjunto *DCL* (*Data Control Language*). Essa é a muito importante, pois é por meio dessas instruções que o profissional desenvolvedor ou programador controla privilégios e o nível de acesso dos objetos do banco.

- GRANT: permitindo acesso a objetos do banco de dados.

```
GRANT SELECT, INSERT, UPDATE ON estudantes TO Maria;
```

- REVOKE: removendo o acesso a objetos do banco de dados.

```
REVOKE SELECT ON estudantes FROM  Maria;
```

- DENY: bloqueando acesso para objetos e usuários específicos.

```
DENY SELECT ON estudantes TO João;
```

2.3 *SQL INJECTION*

SQL injection é um método bastante conhecido para realizar ataques a banco de dados através de formulários que contenham entradas do tipo texto. O ataque via injeção de *SQL* do inglês *SQL injection*, é um ataque realizado quando um código *SQL* é inserido ou concatenado aos parâmetros de entrada fornecidos pelo usuário e posteriormente o código é encaminhado ao banco de dados que o interpreta e executa (CLARKE, 2009). Os desenvolvedores dessas aplicações potencialmente vulneráveis devem ter total atenção a segurança, caso contrário será alvo fácil para futuros ataques de *SQL injection*. Esse ataque consiste basicamente em um comando *SQL* injetado em algum *input* ou um componente de *popscreen* que pode ser feito através de uma manipulação de pacote, variáveis ou simplesmente através de *input* no próprio site ou em sua URL. Ou seja, é um ataque que tem por base manipular código *SQL*.

2.3.1 Um exemplo de como funciona um ataque de injeção SQL em uma aplicação

Uma instrução montada *SQL* para fazer consulta responsável por efetuar o *login* em um sistema.

SELECT * FROM usuarios
WHERE login = ‘admin’ AND senha = ‘minhasenha’

Aparentemente não tem nada de errado com a consulta *SQL*, mas retornará ao usuário caso este campo de *login* e senha sejam atendidos corretamente com os valores que foram enviados, para uma instrução vinda do *back-end* da aplicação.

Em caso de mudança clausula:

SELECT * FROM usuarios
WHERE login = ‘admin’--’ AND senha = ’minhasenha’

Observamos que a *query* é igual, o que mudou é que foi enviado como argumento para campo *login* não mais ‘admin’, mas sim ‘admin’--’.

Em alguns SGBDs (Sistema de Gerenciamento de Banco de dados) que são um sistema diferenciadores de banco de dados “--” representa um comentário, que o nosso caso vai desativar o restante da cláusula, o que vem a ser possível o *login* no sistema, nesse caso apenas usando *login*, ignorando completamente o campo da senha.

Com uma simples mudança é possível quebrar a segurança de uma *query* de consulta, o que vem a ter ligação direta em como nós desenvolvemos nossas aplicações.

2.3.2 Concatenação de comandos de comandos com injeção *SQL*

Query *SQL* comum criada em sistemas. Selecionando todos os campos da tabela “Produtos” para o produto que tem o *id* igual a 10.

SELECT * FROM produtos WHERE id = 10

2.3.3 Concatenação de comandos com injeção de SQL

/* Query com injeção SQL */

SELECT * FROM produtos WHERE id = 10; DROP produtos - - ID = 10; DROP produtos - -

Utilizando a *string* “10; *DROP* produtos - - “, o “;” seria forma de empilhar mais *querys* em uma única chamada. O *SQL Server* é capaz de suportar esse tipo de execução, dependendo da proteção ou da falta dela, é possível criar, alterar, e até remover tabelas do sistema. Em casos extremos é possível até ganhar acesso remoto aos servidores através do banco de dados.

A injeção *SQL* ainda é muito explorada, atualmente ainda existe muitos sites vulneráveis que estão mal configurados a essas vulnerabilidades no banco de dados.

2.3.4 Métodos de prevenção

Portanto, podemos sugerir possíveis soluções de prevenção *SQL injection* incidem em boas práticas de prevenção das aplicações. Segundo Sêmola (2014, p.43): “toda informação ser mantida na mesma condição em que foi disponibilizada pelo seu proprietário, visando protegê-la contra alterações indevidas intencionais ou acidentais”. Uma forma apropriada de se prevenir quanto a isso é tentar ao máximo minimizar as chances de uso de alguns caracteres na consulta ao banco de dados.

Além de visar indispensavelmente conter o ensino de boas práticas de programação a softwares seguros, de acordo com a (OWASP, 2020) as melhores práticas para se prevenir de um ataque *SQLi* (*Structured Query Language Injection)* são descritas no subtópico a seguir:

2.3.5 Parametrização das consultas

Consultas que obrigam desenvolvedores a obter uma definição de código *SQL* para poder prosseguir cada parâmetro. Essa codificação permite que em um banco de dados haja diferença entre códigos e dados, independente da entrada fornecida do usuário.

2.3.6 Usar *stored procedures*

Procedimento no qual o desenvolvedor deverá criar instruções *SQL* com parâmetros automaticamente. Considerando a diferença entre instruções preparadas e procedimentos armazenados no qual o código *SQL* em um procedimento armazenado é definido e armazenado no banco de dados, e logo após, chamado no aplicativo. Técnicas que tem o mesmo resultado na prevenção da injeção de *SQL*.

2.3.7 Escapar toda entrada fornecida pelo usuário

Técnica que pode evitar entrada do usuário antes de uma consulta, o que passa a ser específico para implementação de um banco de dados. Porém, essa técnica deve ser usada quando as opções acima não forem mais viáveis.

2.3.8 Limitar privilégios aos acessos

Em algumas situações, a validação de consulta é a defesa mais apropriada. O ideal é que não seja requisitado valores de tabela ou coluna como parâmetros do usuário, e sim do código.

2.3.9 Validação de entrada da lista de permissões

Considerando que se trata de uma defesa primaria (quando não há validez em uma variável de ligação), pode-se dizer que a validação de entrada pode ser uma defesa secundária obtendo o uso para detectar entradas não autorizadas antes de passar para uma consulta *SQL*.

2.3.10 Último privilégio

Tratando-se de um banco de dados, injeção de *SQL* não é a única ameaça aos dados. Invasores pode alterar valores de parâmetros de valores legais, no qual são apresentados para um valor não autorizado. Porém, pode haver autorização do próprio aplicativo para acessar. Minimizar privilégios concedidos pelo seu aplicativo pode reduzir a possibilidade de tentativas não autorizadas de acesso.

Vale ressaltar que mesmo utilizando esses métodos de prevenção a este tipo de ataque não podemos minimizar um provável dano provocado por algumas vulnerabilidades. Comparando a injeção *SQL* com outras formas de ataques conhecidas como *buffer overflow*, onde a execução de um programa grava dados além do que o *buffer* de memória permite, sobrecarregando o sistema, devido a utilização de rotina insegura.

O *Web Application Firewall* (*WAF*) monitoram, filtram e bloqueiam pacotes de dados conforme eles trafegam em site ou aplicativo *web*, contribuindo para uma segurança das aplicações e servidores.

A barreira de proteção contra invasores que se aplica um conjunto de regras a uma comunicação *HTTP* (*HyperText Transfer Protocol*) e pode ser implantado para proteger um aplicativo especifico ou conjunto de aplicativos *web*.

Tendo em vista que o *waf* não é uma solução de segurança definitiva, e que devem ser usados em um conjunto com outras soluções de segurança de perímetro de rede, utilizando também *firewalls* e sistemas de intrusão para fornecer uma estratégia de defesa.

3 METODOLOGIA

Este trabalho foi desenvolvido a partir do método de pesquisa bibliográfica, onde foi utilizado um conjunto de etapas, tais como: importância do tema no cenário atual, a extração de dados e por fim uma análise exploratória do tema abordado. O objetivo foi evidenciar vulnerabilidades relacionada a *SQL injection* e propor soluções de proteção.

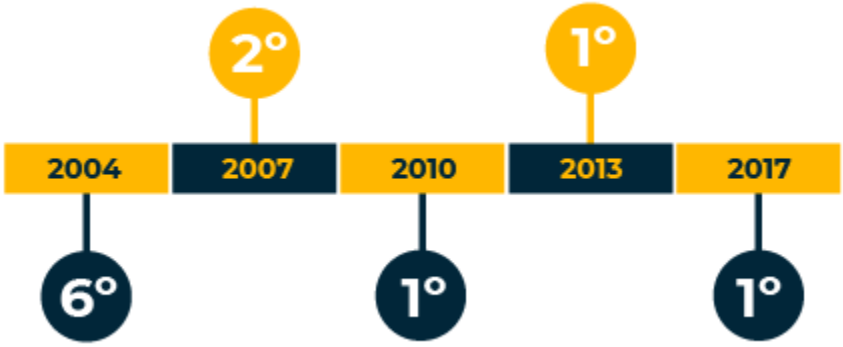
4 RESULTADOS

A Verizon empresa especializada em telecomunicações, anualmente realiza uma avaliação sobre melhores práticas e as vulnerabilidades que mais foram reportadas, semelhante ao que acontece na OWASP. *SQL Injection* e o OWASP Top 10.

Após 17 anos de relatório da OWASP, considerando que o primeiro relatório foi lançado em 2004, atualmente ainda temos esta vulnerabilidade figurando entre o cenário mais importante de *web applications*.

Se traçarmos uma cronologia veremos que a primeira vez que surgiu o relatório foi em 2004, com *injection* já na sexta posição. No relatório seguinte (2007) já estava sem segundo lugar e assim seguiu evoluindo, chegando ao primeiro lugar já no relatório de 2010, onde continua nessa posição até hoje demonstrado pela cronologia de relatório da figura 1.

Figura 1 - Cronologia de relatório

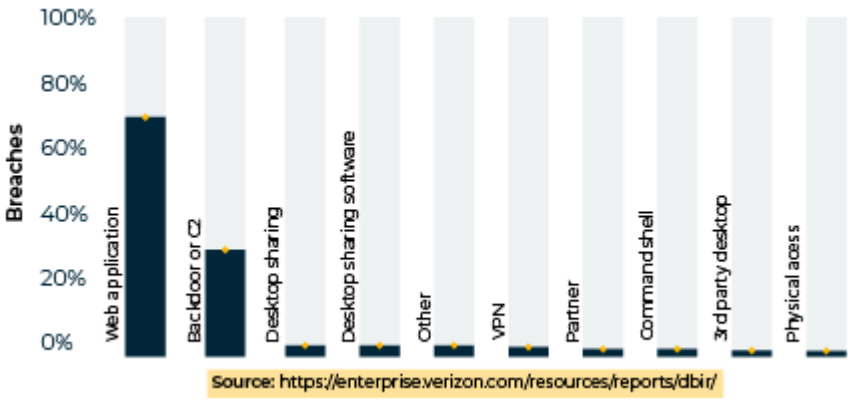


source: https://enterprise.verizon.com/resources/reports/dbir/

São vários os relatórios que ao longo do ano, e principalmente nos meses finais do ano, tentam nos mostrar um cenário do que aconteceu. Neste ano nos foi colocado que mesmo com todas as melhorias em processos de desenvolvimento e outras práticas ainda temos um grande volume dos ataques sendo feitos contra aplicações *web*.

Visto que hoje a grande maioria das aplicações *web* e a grande maioria das empresas tem seu core focado neste tipo de aplicação. O que chama atenção é que mesmo com todos estes dados e informações, ainda temos muitos problemas nas aplicações, sendo o maior deles as vulnerabilidades de *injection*, e dentro desta categoria a *SQL Injection* apresentado no relatório estatístico da figura 2.

Figura 2 - Relatório estatístico



Acunetix considerada um *scanner* de segurança de aplicativo *web* com muitas funcionalidades fornecidas, podendo também encontrar vulnerabilidade na rede. Podendo lidar com muitos dispositivos, oferecendo a capacidade de automatizar sua verificação.

Nessus uma ferramenta que oferece uma ampla variedade no *SO* (Sistema Operacional), aplicativos e vários outros dispositivos entre infraestrutura em nuvem, redes virtuais e físicas. Assim, sendo uma ferramenta que pode impedir as redes de tentativas, feitas por *hackers* e pode verificar as vulnerabilidades que permitem o *hacking* remoto de dados confidenciais.

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho abordou as principais vulnerabilidades *web* por *SQL injection* e com pouco conhecimento técnico é possível explorar as vulnerabilidades com certa facilidade, demonstrando assim o grau de severidade das vulnerabilidades *web*, conhecidas e abordadas pela BATORI onde aponta as 10 principais vulnerabilidades *web* que servirão de base desse estudo.

Essas vulnerabilidades alertam os desenvolvedores de aplicações *web* deve estar atento à possíveis falhas durante o seu desenvolvimento e o seu ciclo de vida.

No entanto, as grandes brechas de segurança que viabilizam a injeção de *SQL injection* estão presentes no código da própria aplicação. É de fato que as boas práticas restringem imensamente o potencial de ocorrência de um ataque. Vale ressaltar que mesmo utilizando de métodos de prevenção a este tipo de ataque não podemos minimizar um provável dano provocado por algumas vulnerabilidades.

Em trabalhos futuros será possível explorar os demais tipos de vulnerabilidades *web* e métodos de prevenção baseado na metodologia BATORI e OWASP complementando este estudo. Recomenda-se o estabelecimento de métricas de segurança que possam auxiliar no desenvolvimento de aplicações *web* a gerir vulnerabilidades e que possam resultar em falhas críticas em aplicações *web*. Também recomenda o estudo de cada falha descrita no top 10 principais vulnerabilidades *web* explorando cada falha minuciosamente em trabalhos distintos.

REFERÊNCIAS

BATORI. Estudo aponta as 10 principais vulnerabilidades em aplicações web. 2018, Brasil. Disponível em: https://administradores.com.br/noticias/estudo-aponta-as-10-principais-vulnerabilidades-em-aplicacoes-web. Acesso em: 22/10/2021.

CLARKE, Justin. SQL Injection Attacks and Defense. 2. ed. Waltam, Massachusetts: Elsevier, 2012. Disponível em: https://asmodeus.pwnsquad.net/collection-6/Hacking/SQL%20Injection%20Attacks%20and%20Defense.pdf.

VERIZON. SQL Injections são nossas baratas digitais. 2020, EUA. Disponível em: https://blog.convisoappsec.com/sql-injection-sao-como-baratas-digitais. Acesso em: 29/10/2021.

OWASP. SQL Injection Prevention Cheat Sheet. [S. l.], 2020. Disponível em: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html . Acesso em: 29 out. 2021.

POSADA, Gilberto Fco Búlfeda. Advanced SQL Injection In SQL Server Applications. San Carlos, Califórnia: Next Generation Security Software Ltd, 2002. Disponível em: https://crypto.stanford.edu/cs155old/cs155-spring09/papers/sql_injection.pdf.

PORTSWIGGER. Injeção SQL, 2021, Brasil. Disponível em: https://portswigger.net/web-security/sql-injection. Acesso em 14/10/2021.