FACULDADE DE CIÊNCIAS DE TIMBAÚBA
CURSO SUPERIOR EM GESTÃO DE TECNOLOGIA DA INFORMAÇÃO

ALAN RODRIGO FERREIRA DE LUCENA
RENAN DIAS LEITE

GESTÃO DE VULNERABILIDADE EM APLICAÇÕES WEB
EXPLORAÇÃO DE SQL INJECTION

Timbaúba/PE
2019

ALAN RODRIGO FERREIRA DE LUCENA
RENAN DIAS LEITE

GESTÃO DE VULNERABILIDADE EM APLICAÇÕES WEB
EXPLORAÇÃO DE SQL INJECTION

Trabalho de Conclusão de Curso apresentado ao curso de Gestão de Tecnologia da Informação, ministrado pela Faculdade de Ciências de Timbaúba, sob Orientação do Professor Sérgio Corrêa dos Santos.

Timbaúba/PE
2019

Ficha catalográfica

Ficha de aprovação

AGRADECIMENTOS

Agradecemos a todos que colaboraram com o desenvolvimento e conclusão dessa pesquisa, toda a nossa família, cônjuges, amigos, professores da FACET, ao nosso orientador Prof. Sérgio Corrêa dos Santos, pela amizade, paciência, métodos diferenciados aplicados em sala de aula, oportunidades, por ter acreditado que éramos capazes, e pelas orientações.

Ao Prof. Messias Rafael Batista, por suas recomendações metodológicas, ao Prof. Victor Sotero pelo suporte na implementação do ambiente para a prova de conceito e em especial, ao nosso amigo Wesley Paulo Sousa Santos, pelo auxílio em meio à parte prática da pesquisa.

Atrás de um grande sucesso, sempre há uma talentosa equipe.

Augusto Liberato

RESUMO

Por intermédio de uma maior expansão relacionada a utilização de serviços em páginas *web*, ponderada, em paralelo, à junção acentuada às estruturas sistematizadas de dados, as degenerações originadas por ataques, até o presente, são caracterizadas como um desafio pertinente para instituições e organizações que, porventura, são suscetíveis a esse tipo de adversidade. Ocorre a necessidade da adoção de boas práticas, quanto às políticas de segurança, para evitar problemas à saúde dos dados da organização. Assim, tal estudo visa evidenciar a importância da adoção das orientações da OWASP de defesa contra o SQL *Injection,* alinhado à gestão de segurança, em face às consequências que a vulnerabilidade pode ocasionar. Objetivando o desenvolvimento de um ambiente didático e prático para a experimentação e aplicação dos conceitos apresentados nesse trabalho, foi implementado um protótipo conceitual de aplicação *web,* contendo, propositalmente, a vulnerabilidade de Injeção de SQL. Foram adotadas, na sua implementação, tecnologias e abordagens que tornaram fácil a execução dos procedimentos de teste/ataque, compreensão da implementação, vulnerabilidade, e a correção das falhas. Conclui-se que, se as medidas de proteção recomendadas pela OWASP, tomando como base as diretrizes do padrão de segurança de dados do PCI DSS, puderem ser adotadas de forma mais eficiente na manutenção e construção de aplicações web, a incidência de ameaças de injeção de SQL será significativamente menor.

Palavras-chave: Ameaça; Desenvolvimento; Injeção de SQL; OWASP; PCI DSS; Segurança.

ABSTRACT

Through further expansion related to the use of services on web pages, weighted, in parallel, to the strong joining of systematized data structures, the degenerations caused by attacks, until now, are characterized as a pertinent challenge for institutions and organizations that may be susceptible to this kind of adversity. There is a need to adopt good practices regarding security policies to avoid problems to the health of the organization's data. Thus, this study aims to highlight the importance of adopting OWASP guidelines about defense against SQL Injection, aligned with security management, in view of the consequences that vulnerability may cause. Aiming the development of a didactic and practical environment for the experimentation and application of the concepts presented in this work, a conceptual web application prototype was implemented, purposely containing the SQL Injection vulnerability. Technologies and approaches have been adopted in their implementation that have made it easy to perform test / attack procedures, understand implementation, vulnerability, and correction of the failures. It is concluded that if OWASP recommended protection measures based on the PCI DSS data security standard guidelines, can be adopted more efficiently in maintaining and building web applications, the incidence of SQL injection threats will be significantly smaller.

Keywords: Threat; Development; SQL Injection; OWASP; PCI DSS; Safety.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE ABREVIATURAS

ANSI – American National Standards Institute

API – Application Programming Interface

CERT – Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil

CIS – Center for Internet Security

DDL – Data Definition Language

DML – Data Manipulation Language

DNS – Domain Name System

HTML – Hypertext Markup Language

HTTP – Hyper Text Transfer Protocol

HTTPS – Hyper Text Transfer Protocol Secure

IP – Internet Protocol

ISO – International Organization for Standardization

ISSAF – Information System Security Assessment Framework

NIST – National Institute of Standards and Technology

OSI – Open Systems Interconnection

OSSTMM – Open Source Security Test Methodology Manual

OWASP – Open Web Application Security Protect

PCI DSS – Payment Card Industry Data Security Standard

PHP – Hypertext Preprocessor

RFC – Request for Comments

SANS – SysAdmin Audit Network Security

SGDB – Sistema de Gerenciamento de Banco de Dados

SQL – Structured Query Language

SQLI –Structured Query Language Injection

SSL – Secure Socket Layer

TCP – Transmission Control Protocol

TLS – Transport Layer Security

URL – Uniform Resource Locator

WAF – Web Application Firewall

SUMÁRIO

1 INTRODUÇÃO

Mediante uma maior disponibilidade dos dados na rede mundial de computadores, fez-se impulsionar novos interesses. E, conforme tal evolução de serviços, afloraram, igualmente, dificuldades relacionadas à segurança dos dados dispostos, proporcionando aplicações com falhas em suas implementações. Acarretando, assim, a exploração de lapsos, dentre tais, fragilidades que são recorrentes atualmente. Segundo a *Open Web Application Security Project* (OWASP, 2017), são as falhas de implementação que permitem a introdução de códigos, realizando, desse modo, consultas ou modificações nos Bancos de Dados de forma arbitrária, em especial o *Structured Query Language Injection* (SQLI), que, têm sido amplamente utilizada na captura e desvio de informações e invasões de redes corporativas. Tipificando o cenário em que ataques são cada vez mais direcionados às aplicações dispostas em tal ambiente.

Dentre os quais, abundantes fatores contribuem para que aplicações *web* tornem-se alvos recorrentes de ataques. Segundo Lins (2017), segurança relacionada a aplicações *web* ainda são aspectos negligenciados. Fazendo com que, em diversos casos, não exista equipe de segurança executando ações de monitoramento refletida nas atividades, impossibilitando uma resposta efetiva aos incidentes. Equipe essa que se faz indispensável no cenário em que se identifica um aumento na complexidade das aplicações, do acesso à Internet, e dos recursos relacionados à conexão com a rede, salienta Stallings (2008).

> Esse aumento nos ataques coincide com o aumento do uso da Internet e com aumentos na complexidade dos protocolos, aplicações e da própria Internet. Infraestruturas críticas contam cada vez mais com a Internet para as suas operações. Os usuários individuais contam cada vez mais com a segurança da Internet, de e-mail, da Web e das aplicações baseadas na Web. Assim é preciso que haja uma grande gama de tecnologias e ferramentas para agir contra a ameaça crescente. Em um nível básico, os algoritmos criptográficos para a confidencialidade e autenticação assumem maior importância. Além disso, os projetistas precisam focalizar os protocolos baseados na Internet e as vulnerabilidades dos sistemas operacionais e das aplicações atacadas. (STALLINGS, 2008, p. 5).

No âmbito estrutural, devido ao avanço da tecnologia, refletida na segurança, implementações em servidores, redes e serviços, detêm um melhor benefício

refletido à proteção atual. Todavia, algumas aplicações não possuem dispositivos relacionados à proteção aos dados consumidos na *web,* como os *WAF's*[1] que buscam dispor de uma camada intensiva de proteção na aplicação. Como resultado, ataques ordenados sucedem em aplicações que realizam requisições diretamente ao banco de dados, por exemplo.

A presente pesquisa apresenta a vulnerabilidade SQL *Injection.* Estimando suas consequências, de forma que, ao se fazer o tratamento da vulnerabilidade baseado em metodologias de desenvolvimento, constituirá uma possível manifestação da importância, ponderado das orientações e defesas relativas à segurança de aplicações, em face da vulnerabilidade.

1.1 APRESENTAÇÃO DO TEMA

Por intermédio de uma maior expansão relacionada à utilização de serviços em páginas *web*, ponderada em paralelo, a junção acentuada às estruturas sistematizadas de dados. As degenerações originadas por ataques, até o presente, são caracterizadas como um desafio pertinente para instituições e organizações que, porventura, são suscetíveis a esse tipo de adversidade. O Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT) aponta um aumento no número de incidentes de segurança em ambientes *web,* em relação aos anos de 2016 e 2017, no qual 53,16% dos casos reportados são originários de *scans*[2] baseados em ferramentas de detecção de vulnerabilidades (CERT, 2018).

Uma das técnicas difundidas com maior quantitativo de exploração de tais vulnerabilidades é o SQL *Injection,* que consiste na introdução de código para preenchimento da estrutura de requisição SQL em formulários *web,* contendo campos de pesquisa, *login* e senha, com exemplo. Técnica que permite, caso se apresente a vulnerabilidade, certa modificação da *string*[3] de comunicação da aplicação com o interpretador do banco de dados, permitindo a definição de

[1] Um firewall de aplicativo da *web* filtra, monitora e bloqueia o tráfego HTTP para e de um aplicativo da *web*.
[2] Técnica normalmente implementada por um tipo de programa, projetado para efetuar varreduras em redes de computadores.
[3] É uma sequência de caracteres, geralmente utilizados para representar palavras, frases ou textos de um programa, expressas por variáveis.

comandos de manipulação de dados - *Data Manipulation Language* (DML) ou comandos de definição de dados – *Data Definition Language* (DDL).

A linguagem[4] responsável pela manipulação e gerenciamento de dados é a SQL, proposta juntamente com o modelo relacional de banco de dados, no ano de 1970, por Edgar Frank Codd e padronizada pela *American National Standards Institute* (ANSI) para a operação de banco de dados relacionais. Padronizada, a SQL veio a se tornar a linguagem declarativa de dados relacionais em oposição a outras.

Segundo a *International Organization for Standardization* (2005):

> A informação é um ativo que, como qualquer outro ativo importante, é essencial para os negócios de uma organização e consequentemente necessita ser adequadamente protegida. Isto é especialmente importante no ambiente dos negócios, cada vez mais interconectado. Como um resultado deste incrível aumento da interconectividade, a informação está agora exposta a um crescente número e a uma grande variedade de ameaças e vulnerabilidades. (ISO 27002, 2005, p. 5.)

Concebendo que deve manter o cuidado com a manutenção e construção de aplicações *web,* em nível de boas práticas de desenvolvimento, para que estas sejam usufruídas, através da rede, de forma mais segura, as orientações e defesas contra o SQL *Injection* da OWASP são a base para a proteção contra tais vulnerabilidades.

1.2 PROBLEMÁTICA

Evidenciar a importância da adoção de orientações e defesa contra a Injeção de SQL, em meio ao processo associado ao desenvolvimento seguro de aplicações *web,* proposto pela OWASP. Assim, expondo de forma contínua, recomendações e implementações relativas à sua importância. Aspirando minimizar eventuais danos, provenientes de técnicas amplamente difundidas, relacionada, como exemplo, ao tema proposto neste projeto.

---

[4] No final de 1973, um grupo de pessoas trabalhava nos laboratórios de pesquisa da *IBM Corporation*, com o objetivo de desenvolver uma linguagem específica que lhes permitiriam acessar essas bases de dados de forma padrão, que nascidas talvez sob outro conceito, começaram a se adaptar ao chamado modelo relacional, que estava emergindo como um padrão de fato na geração de modelos complexos de dados. (Racciatti, 2002, p. 3, tradução livre).

A Injeção de SQL lidera o ranking das 10 (dez) vulnerabilidades mais recorrentes do ano de 2017, segundo o OWASP (2017). Logo, o problema permite o seguinte questionamento:

Qual a importância da implementação de defesas que proporcionam maior proteção às aplicações *web* contra o *SQL Injection*? Ao passo que através do procedimento metodológico adotado nesse trabalho vamos buscar demonstrar que depois da adoção e implementação das recomendações pode-se verificar a proteção contra a vulnerabilidade no ambiente de teste proposto nesse estudo. Assim, ao se evidenciar a importância da proteção contra a vulnerabilidade SQL *Injection*, mediante o processo associado ao desenvolvimento seguro de aplicações *web* em que são usados códigos com técnicas de tratamento de entrada de dados fica evidente que, não se trata apenas de um problema de ordem estrutural, pois se tais medidas fossem adotadas de forma mais eficiente, a incidência de ameaças do tipo injeção SQL seria significativamente menor.

1.3 JUSTIFICATIVA

Organizações e seus sistemas de informação enfrentam, cada vez mais, ameaças de segurança. Isso se deve, em grande parte, à sua dependência de sistemas e serviços de informação. Independente de qual forma se caracterize, as informações são ativos, como qualquer outro ativo importante para os negócios, que deve ser protegido adequadamente. Conforme a ISO 17799 (2000), a segurança da informação baseia-se na preservação da confidencialidade, integridade e disponibilidade, ao passo que são fundamentais para a competitividade nos negócios da empresa.

> Gemalto (Euronext NL0000400653 GTO), líder mundial em segurança digital, lançou hoje as mais recentes descobertas do Breach Level Index, revelando que 2,6 bilhões de dados foram roubados, perdidos ou expostos mundialmente em 2017, um aumento de 88% em relação a 2016. Enquanto os incidentes de violação de dados diminuíram 11%, 2017 foi o primeiro ano de divulgação pública em que as violações superaram mais de 2 bilhões de registros de dados comprometidos desde que o Breach Level Index começou a rastrear as violações de dados em 2013. (INDEX, 2018).

A integridade, confidencialidade e disponibilidade dos dados em aplicações *web* são de suma importância na esfera organizacional. Entretanto, com a ampliação

de serviços na *web*, há um aumento no índice de vulnerabilidades, devido à constância do aperfeiçoamento de novas práticas e do surgimento de novos métodos de SQL *Injection*, resultantes da negligência em relação às políticas de segurança, originando diversos problemas à segurança dos dados. Baseado nos dados da *Breach Level Index,* é explícita, a necessidade de execução de ações preventivas, que atendam às necessidades das organizações, visto que a negligência, perante as políticas de segurança, implica em grandes perdas para as organizações, tais como: Financeiras e exposição de dados sensíveis.

Nesse contexto, as seguintes hipóteses são consideradas: A necessidade da adoção de boas práticas, quanto às políticas de segurança, que são importantes para evitar problemas à saúde dos dados da organização. Os problemas resultantes da negligência ou desconhecimento, com relação à execução de orientações de segurança, tornam aplicações suscetíveis a vulnerabilidades. Assim, tal estudo visa evidenciar a importância da adoção das orientações da OWASP de defesa contra o SQL *Injection,* alinhado à gestão de segurança, em face às consequências que a vulnerabilidade pode trazer. De modo geral, o presente estudo segue a linha de pesquisa da Segurança da Informação, ao passo que estabelece interlocuções com a área da Gestão da Tecnologia da Informação.

1.4 OBJETIVOS

1.4.1 Objetivo Geral

Evidenciar a importância da adoção das orientações da OWASP de defesa contra o SQL *Injection,* alinhado à gestão do plano de segurança e desenvolvimento, em face às consequências da vulnerabilidade.

1.4.2 Objetivos Específicos

- Caracterizar a vulnerabilidade SQL *Injection* através de um ambiente configurado;
- Aplicar as orientações e defesas contra SQL *Injection* recomendadas pela OWASP;

- Demonstrar a importância da gestão do plano de segurança no desenvolvimento de aplicações *web* baseado nas orientações e defesas contra o SQL *Injection* propostas pela OWASP;

1.5 Estrutura do Trabalho

O presente trabalho está estruturado em 5 (cinco) capítulos, como descrito a seguir:

Capítulo I: Introdução – Introdução e contextualização acerca da temática.

Capítulo II: Fundamentação Teórica – Síntese analítica baseada na bibliografia relacionada à temática com o objetivo de dar embasamento para o desenvolvimento do estudo.

Capítulo III: Metodologia – Trata das metodologias utilizadas para o desenvolvimento do estudo.

Capítulo IV: Desenvolvimento – Desenvolvimento do estudo da temática, baseado na metodologia e referencial teórico-metodológico apresentado.

Capítulo V: Considerações Finais – Trata das considerações finais, sintetizando as contribuições da pesquisa, e indicando caminhos para futuros estudos.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 AMBIENTE WEB

2.1.1 Arquitetura Cliente-Servidor

Tradicionalmente, a comunicação entre cliente-servidor acontece de forma que o servidor, quando ativo, vai “escutar” a porta 80, por padrão, em um processo também conhecido como “*listering*”, em que será recebida uma requisição, via protocolo HTTP à página solicitada. Tanenbaum (1994), traz um detalhamento desse processo, nas seguintes etapas:

1. O navegador determina o URL (*Uniform Resouce Locator*), verificando qual foi selecionado.

2. O navegador pergunta ao DNS[5] (*Domain Name System*) qual o endereço de IP[6] (*Internet Protocol*) do site em questão.
3. O DNS responde ao navegador com o IP.
4. O navegador estabelece uma conexão TCP[7] (*Transmission Control Protocol*) com a porta 80 (porta padrão do serviço HTTP) do servidor.
5. Em seguida, o navegador envia um comando GET, o qual requisita uma determinada página.
6. O servidor envia a página solicitada.
7. A conexão TCP é encerrada.
8. O navegador apresenta a página.
9. O navegador busca e apresenta as imagens da página *web*.

Não obstante, os servidores *web* não só permanecem apenas com a função de servir a requisições de páginas estáticas.

2.1.2 Servidor Web

Dispõe sua responsabilidade em receber requisições HTTP e HTTPS, fazendo a entrega de conteúdo para exibição pelos navegadores de internet pertencentes a camada de aplicação. Existem vários exemplos de servidores, sendo alguns deles proprietários. Para o presente estudo, escolhemos o *Apache*, pois além de ser um servidor de código fonte aberto, por intermédio da sua API (*Application Programming Interface*), pode-se implementar novas funcionalidades para o aumento referente à segurança, apresentando-se também, como um servidor de alta performance.

Ainda que configurados e atualizados com diretrizes de segurança, tais serviços podem ser comprometidos devido a falhas nas aplicações *web* em decorrência do mau desenvolvimento.

---

[5] (Sistema de Nomes e Domínios, em português), responsável por decodificar os nomes dos domínios dos sites digitados nos navegadores *web* em números IP.

[6] Endereço de (Protocolo da Internet, em português), que se caracteriza por um rótulo numérico atribuído a cada dispositivo conectado a uma rede de computadores que utiliza protocolo de Internet para comunicação.

[7] (Protocolo de Controle de Transmissão, em português), também chamado de TCP/IP pelo fato de ser complementado pelo Protocolo da Internet (IP), onde o TCP é um protocolo de nível da camada de transporte onde se assentam grande parte de outras aplicações.

2.1.3 Linguagem de desenvolvimento *Web* PHP

PHP apresenta-se como uma linguagem, de modo que pode ser executado tanto em plataforma Windows quanto em várias distribuições Unix. No PHP, não precisamos declarar variáveis antes de usá-las, assim como *arrays*[8] podem ser definidos de uma melhor forma. Sem contar que algumas características de orientação a objetos se fazem presentes, permitindo a organização e o encapsulamento do código (CÔRTE, 2002). Além disso, tem suporte a um grande número de banco de dados, e também a outros variados serviços, baseados em diversos protocolos, ainda sendo possível abrir sockets e interagir com mais deles, inclusive sendo possível criar arquivos de diferentes formatos. (ROCHA, 2007). O uso de *tags* permite que o Servidor *Web* reconheça que se trata de um código PHP que está dentro das mesmas, e possa chamar o interpretador PHP para executar o código.

O PHP pode trabalhar em conjunto com a linguagem de marcação, o HTML, permitindo que em um mesmo arquivo contenha blocos de instruções com essas linguagens, de modo a ficar responsável pela parte dinâmica, enquanto o HTML pela parte estática, que fica fora das *tags*.

2.1.4 Banco de Dados

Banco de Dados é uma coleção de dados relacionados. Devido ao fato de ser um termo genérico, a definição de um banco de dados envolve especificar os tipos, estruturas e restrições dos dados a serem armazenados. Sua construção se baseia no processo de armazenamento dos dados em meio controlado, o que geralmente é feito pelo SGBD (Sistema de Gerenciamento de Banco de Dados), (ELMASRI, 2005).

O ambiente de testes que foi configurado possui o MySQL como gerenciador da base de dados. Sua escolha foi fundamentada em algumas características que permitem um melhor desempenho e gestão dos dados que ele dará suporte.

---

[8] Matriz que armazena vários valores em uma única variável.

2.2 SEGURANÇA DA INFORMAÇÃO

A ABNT NBR ISO/IEC 17799:2005 (2005, p.ix), define segurança da informação como sendo “a proteção da informação de vários tipos de ameaças para garantir a continuidade do negócio, minimizar o risco ao negócio, maximizar o retorno sobre os investimentos e as oportunidades de negócio”. De forma que;

> A segurança da informação é obtida a partir da implementação de um conjunto de controles adequados, incluindo políticas, processos, procedimentos, estruturas organizacionais e funções de software e hardware. Estes controles precisam ser estabelecidos, implementados, monitorados, analisados criticamente e melhorados, onde necessário, para garantir que os objetivos do negócio e de segurança da organização sejam atendidos. Convém que isto seja feito em conjunto com outros processos de gestão do negócio. (ABNT NBR ISO/IEC, 2005, p. 9).

2.2.1 Confidencialidade, Integridade e Disponibilidade

Segundo Stallings (2008), a Recomendação X.800 da ITU-T, *Security architecture for OSI* ainda define um serviço de segurança como um serviço que é fornecido por uma camada de protocolo de comunicação de sistemas abertos, que garante segurança nos sistemas ou das transferências de dados. Corroborando com essa definição, a RFC (*Request for Comments*) da Internet, sob a referência RFC 2828, *Internet Security Glossary,* oferece a seguinte definição:

> Um serviço de processamento ou comunicação que é fornecido por um sistema para prover um tipo específico de proteção aos recursos do sistema; os serviços de segurança implementam políticas (ou diretrizes) de segurança e são implementados por mecanismos de segurança. (STALLINGS, 2008, p. 8.)

Tais serviços ainda encontram, na Recomendação X.800, uma distinção em cinco categorias, e quatorze serviços específicos, sendo que somente apenas a Confidencialidade, Integridade e Disponibilidade, com suas respectivas categorias serão referenciadas no seguinte estudo. A vantagem competitiva das empresas se baseia na capacidade de controlar a divulgação de informações importantes.

A Recomendação também estabelece que é necessário se ter a proteção do fluxo de tráfego contra análise, de modo que o atacante não tenha possibilidade de identificar a origem e o destino do mesmo. Os tipos de ataques que normalmente

ocorrem nesse contexto são do tipo passivo, no qual a proteção se baseia no serviço de segurança pautado em medidas preventivas. Baseando-se na Recomendação X.800, Integridade é a garantia de que os dados recebidos estão exatamente como foram enviados por· uma entidade autorizada (ou seja, não contém modificação, duplicação, inserção, reordenação ou exclusão), sendo assim, é um dos aspectos de extrema importância para garantir os dados armazenados, e transmitidos por Sistemas de Informação.

A Integridade se apresenta como um serviço também orientado à conexão, de modo que está relacionado a um determinado fluxo de dados ou determinado campo dentro de uma mensagem. Assim, a melhor forma seria a proteção total do fluxo. E, ao contrário da confidencialidade, o serviço de integridade está voltado à proteção contra os ataques ativos, passando a atuar no tratamento da detecção do ataque.

Segundo Stallings (2008), tanto a Recomendação X.800 quanto a RFC 2828 vão definir disponibilidade como uma propriedade de um sistema, ou recurso desse mesmo em estar acessível, disponível, ou ser utilizável por alguma entidade autorizada. Logo, o serviço de disponibilidade vai preconizar medidas para a proteção do sistema, com o objetivo de garantir sua disponibilidade sempre que os usuários precisarem.

2.2.3 Vulnerabilidade e Ameaças

A ABNT NBR ISO/IEC 17799:2005 (2005, p.3), define vulnerabilidade como sendo a “fragilidade de um ativo ou grupo de ativos que pode ser explorada por uma ou mais ameaças”. A respeito da Gestão de Segurança da Informação, é necessário prover a devida importância a ameaças que podem comprometer a segurança da informação. É a partir delas que se pode estimar os riscos que envolvem a atividade organizacional.

Segundo a ABNT NBR ISO/IEC 17799:2005 (2005, p.3), ameaça é a “causa potencial de um incidente indesejado, que pode resultar em dano para um sistema ou organização”. As ameaças podem ser de dois tipos: físicas ou lógicas. De modo que também podem ser internas ou externas. Segundo Barrett e King, (2010), uma ameaça é qualquer coisa que coloque em perigo a segurança da rede. Complementando o conceito, a (RFC 2828) define ameaça como potencial para a violação da segurança quando há uma circunstância, capacidade, ação ou evento

que pode quebrar a segurança e causar danos. Ou seja, uma ameaça é um possível perigo que pode explorar uma vulnerabilidade. Barrett e King, (2010) informam que, a avaliação das ameaças seja iniciada logo quando se configura um servidor. Além do que, a maioria dos servidores vem com configurações padrões, que os deixam passíveis de acesso não autorizado, ou ameaças.

2.2.2 Aspectos da Política de Segurança da Informação (PSI)

> Política de segurança de informações é um conjunto de princípios que norteiam a gestão de segurança de informações e que deve ser observado pelo corpo técnico e gerencial e pelos usuários internos e externos. As diretrizes estabelecidas nesta política determinam as linhas mestras que devem ser seguidas pela instituição para que sejam assegurados seus recursos computacionais e suas informações. (TRIBUNAL DE CONTAS DA UNIÃO, 2012).

Políticas de segurança claras e definidas são importantes para manter a saúde de uma organização. Cabe à gestão incentivar que elas sejam praticadas pelos seus usuários e colaboradores. A implementação de políticas de segurança requer cautela, sendo que seu retorno pode ser difícil de estimar, pois é uma despesa que não gera receita, de modo que negligenciá-las pode acarretar adversidade. As políticas de segurança, nesse âmbito, têm como objetivo auxiliar na compreensão da dinâmica contida em um ataque, objetivando direcionar o desenvolvimento seguro nas áreas que necessitam de maior atenção.

Ao gerente de segurança, deve-se atribuir o papel de poder fazer uso de uma metodologia sistemática para definir os requisitos, de segurança e definir as técnicas para satisfazer esses requisitos partindo da premissa de avaliar efetivamente as necessidades de segurança de uma organização, a fim de tomar as melhores decisões e obter os melhores resultados. Note-se que dificuldades já são recorrentes em ambiente de processamento de dados centralizado de modo que; com o uso de redes de longa distância, os problemas são agravados. (STALLINGS , 2008).

Alinhado ao uso de algumas ferramentas, a gestão também pode fazer uso de outros meios para a prevenção e mitigação dos ataques em serviços dispostos; Normas, políticas de segurança, ISOs, dentre outras estratégias, como testes e detecção de vulnerabilidades, os quais são fundamentais para verificar se o sistema

está passível de acesso indevido, tendo ele mecanismos de proteção já incorporados ou não. Uma arquitetura de grande utilidade para a gerência em segurança é a OSI (*Open Systems Interconnetion*) contida na Recomendação X.800 da ITU-T,[9] *Security architecture for OSI.* Ela é quem lança base para a adoção da sistemática para boas práticas de segurança que podem ser adotadas.

2.2.4 Ataques

Segundo a RFC 2828, um ataque pode ser considerado como atentado à segurança de um sistema derivado de uma ameaça inteligente, ou seja, um ato inteligente que é uma tentativa deliberada (especialmente usando-se de técnicas) de burlar os serviços de segurança e violar política de segurança de um sistema. Quanto à origem dos ataques, eles podem ser internos ou externos. Exemplos de ataques internos são aqueles que podem surgir de um erro causado por um funcionário despreparado, que não seguiu as políticas de segurança da organização, ou, ainda, por insatisfação, em que um funcionário pretenda penalizar ou repassar informações para a concorrência em meio à espionagem industrial.

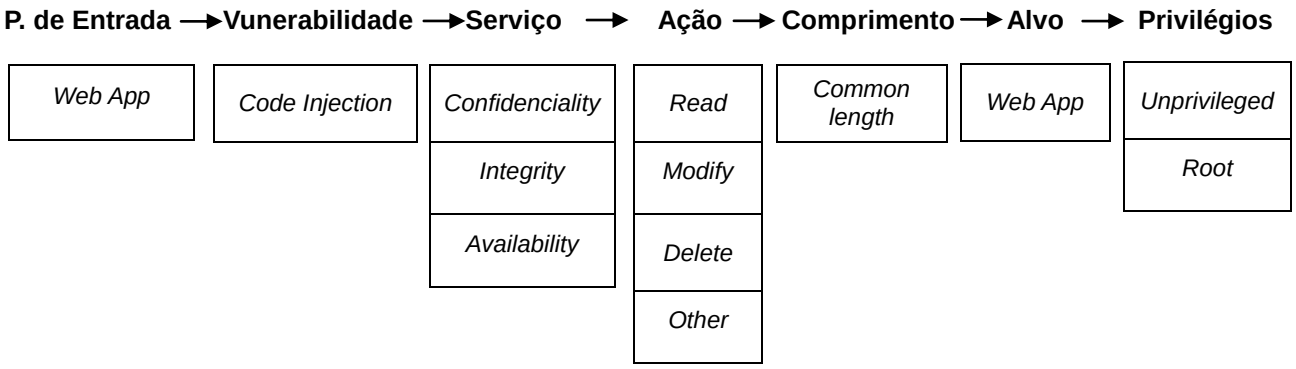
Já os ataques externos podem ser oriundos da espionagem industrial, que podem ocorrer devido à corrida pelo domínio do mercado. Em tal grau, a recomendação X.800, referente à RFC 2828 também classifica os ataques como sendo de dois tipos; *ataques passivos* e *ataques ativos*. Os ataques passivos caracterizam-se pelo fato de o agente ter o intuito de monitorar a transmissão, a fim de obter informações acerca do tráfego de rede.

Ataques ativos caracterizam-se pelo fato de haver, de certa, forma alguma mudança do fluxo de dados ou criação de um fluxo falso. Os ataques *web* se baseiam na exploração dos protocolos HTTP e HTTPS. Cruz, (2017), adapta uma taxonomia para ataques *web* baseada no *FIRST: Common vulnerability scoring system (cvss),* com o intuito de poder auxiliar na compreensão dos ataques e na construção de aplicações mais seguras. Neste estudo, por sua vez, serão adaptados apenas partes da taxonomia que concerne ao SQL *Injection,* para o alcance dos mesmos objetivos.

---

[9] International Telecommunications Union (ITU) Telecommunication Standardization Sector (ITU-T): Agência patrocinada pelas Nações Unidas que desenvolve padrões “Recomendações” voltadas à área de telecomunicações e à Open Systems Interconnection (OSI).

Taxonomia de Ataques Web – SQL *Injection:*



1. Ponto de Partida:

- *Web Aplication Attacks* (Ataque em Aplicações *web*): Ataque resultante da exploração da vulnerabilidade nas Aplicações *Web*.

2. Vulnerabilidade:

- *Code Injection* (Injeção de Código): Ataque SQL *Injection* resultante da incorporação de código nas aplicações *web,* mediante rotinas de validação de entrada mal configuradas.

3. Serviço:

- *Confidentiality* (Confidencialidade): Proteção contra divulgação de informação privada a entidades não autorizadas.
- *Integrity* (Integridade): Objetiva proteger os dados contra manipulação e modificação indesejada.
- *Avaliability* (Disponibilidade): Objetiva a garantia de disponibilidade de um determinado ativo de TI.

4. Ação:

- *Ready* (Leitura): Permite a leitura dos dados contidos no banco de dados do servidor *web*.
- *Modify* (Modificar): Permite a alteração dos dados contidos em um registro ou mais registros do banco de dados.
- *Delete* (Exclusão): Permite apagar determinado registro do banco de dados.

- *Other* (Outros): Qualquer outra ação via SQL *Injection* que manipule o banco de dados.

5. Comprimento dos Argumentos para a Solicitação HTTP:

- *Common length* (Comprimento Comum): Ataques baseados em argumentos curtos.

6. Alvo:

- *Web Application* (Aplicação *Web*): Aplicação, ou sistema projetado para acesso, via navegador, com objetivo de promover acesso a determinado serviço.

7. Privilégios:

- *Unprivileged User* (Usuário sem Privilégios): Quando se obtêm as credenciais de acesso de um usuário com poucos privilégios administrativos.
- *Root* (Administrador): Quando se obtêm credenciais de acesso de administrador que possui privilégios máximos em um sistema.

2.3 PADRÃO DE SEGURANÇA DE DADOS - PCI DSS

Para lançar as bases do Plano de Segurança voltado para o desenvolvimento seguro de aplicações *web*, pode-se tomar como fundamento o Padrão PCI DSS[10] (*Payment Card Industry Data Security Standart*), originalmente elaborado para lançar diretrizes, compor um conjunto de requerimentos e procedimentos de aprimoramento de segurança, com a finalidade de proteger os dados dos titulares de cartões, conforme também possibilita adoção de medidas de segurança de dados em caráter mais genérico.

Os requisitos especificados podem ser aplicados a todas as entidades envolvidas nos processos da organização e elementos de sistemas que participam do processamento de dados. Fazem parte: componentes de rede, servidores, aplicativos e sistemas de gerenciamento de base de dados. A organização dos requisitos os classifica em seis grupos logicamente relacionados, dos quais quatro

[10] (Padrão de Segurança de Dados da Indústria de Pagamento com Cartão, em português), responsável por definir diretrizes e compor um conjunto de requerimentos e procedimentos de segurança com a finalidade de proteger os dados dos portadores de cartões.

serão a base do Plano de Segurança, voltado para o desenvolvimento seguro de aplicações *web.*

1. **Construir e manter a segurança de rede e sistemas**
2. **Proteção de Dados**
3. **Manter um programa de gerenciamento de vulnerabilidades**
4. **Monitorar e testar as redes com regularidade**

O Escopo do Plano de Segurança voltado para aplicações web para a proteção contra SQL *Injection,* tomará como base os requisitos do Padrão PCI DSS mencionados.

1. Construção e manutenção da segurança de rede e sistemas

Deve existir a proteção dos sistemas frente aos acessos não autorizados de redes não confiáveis ou por meio da internet.

REQUISITOS DO PCI DSS

1.1 Implementação de uma DMZ[11] (*Demilitarized Zone*) para limitar o tráfego para componentes do sistema que oferece serviços, protocolos e portas acessíveis publicamente.

1.2 Limitações do tráfego de entrada da internet a endereços IP na DMZ.

PROCEDIMENTOS DE TESTE

1.1 e 1.2 Análise das configurações do firewall e do roteador para verificar e confirmar a implementação dos requisitos.

ORIENTAÇÃO

1.1 e 1.2 O DMZ é a parte da rede responsável pelo gerenciamento das conexões entre a internet (ou redes não confiáveis) e os serviços que uma empresa disponibiliza para o público (como um servidor *web*). Esse recurso evita que indivíduos mal-intencionados acessem a rede interna da empresa pela internet ou por meio de serviços, protocolos ou portas de forma não autorizada.

[11] (Zona Desmilitarizada, em português), subrede que se situa entre uma rede confiável e uma rede não confiável, promovendo um isolamento físico entre as ruas redes.

2. Proteção dos Dados

Determinação de não se poder utilizar padrões disponibilizados pelo fornecedor para senhas do sistema e outros parâmetros de segurança, haja vista que indivíduos as utilizam com frequência, com o intuito de comprometer os sistemas.

REQUISITOS DO PCI DSS

2.1 Desenvolvimento de padrões de configuração para todos os componentes do sistema, certificando-se que esses padrões abrangem todas as vulnerabilidades de segurança conhecidas, e que estão de acordo com os padrões de fortalecimento do sistema aceitos pelo setor. As fontes dos padrões de proteção do sistema aceitos pelo setor podem incluir, entre outros:

- *Center for Internet Security* (CIS);
- *International Organization for Standardization* (ISO);
- *Instituto SysAdmin Audit Network Security* (SANS);
- *National Institute of Standards and Technology* (NIST).

2.1.1 Implementação de apenas uma função no servidor principal para evitar funções que exigem diferentes níveis de segurança no mesmo servidor. Servidores devem ser implementados separadamente. Porém, em caso de virtualização, uma função principal por componente do sistema virtual deve ser utilizada.

2.1.2 Remoção de todas as funcionalidades desnecessárias; scripts, drivers, subsistemas, sistemas de arquivos e servidores desnecessários.

2.2 Criptografia de todo acesso administrativo que não utiliza console durante a criptografia forte.

PROCEDIMENTOS DE TESTE

2.1 Análise dos padrões de configurações do sistema da organização, para todos os tipos de componentes do sistema e verificação da conformidade dos padrões de configuração do sistema com os padrões de proteção aceitos pelo setor.

1 - Análise das políticas e questionamento frente os funcionários, objetivando a confirmação, atualização dos padrões de configuração do sistema em conformidade com os novos problemas de vulnerabilidade que são identificados.

2 - Análise das políticas e questionamento frente aos funcionários, objetivando a confirmação da aplicação dos padrões de configuração adequadas de sistema, quando novos sistemas forem instalados na rede.

3 - Verificação se os padrões de configuração do sistema incluem todos os procedimentos estabelecidos para todos os componentes do sistema.

2.1.1

1 - Seleção de amostra dos componentes do sistema, inspeção das configurações para verificar se somente uma função principal está implementada por servidor.

2 - No caso do uso de tecnologias de virtualização, efetuar a inspeção das configurações do sistema para verificar se somente uma função principal está implementada por componente ou dispositivo do sistema virtual.

2.1.2

1 - Seleção de amostra dos componentes do sistema, inspeção das configurações, para verificar se todas as funcionalidades desnecessárias foram removidas.

2 - Consulta da documentação e parâmetros de segurança, para verificar se as funções ativadas estão documentadas e suportam a configuração segura.

3 - Consulta da documentação e parâmetros de segurança para verificar se somente as funcionalidades registradas estão presentes nos componentes do sistema da amostra.

2.2

1 - Seleção de amostra dos componentes do sistema, inspeção das configurações, para verificar se o acesso administrativo que não utiliza console se encontra criptografado e realiza o que segue:

2 - Observação do *logon* de um administrador em cada sistema e análise das configurações, para verificar se o método de criptografia forte é invocado antes da senha do administrador ser solicitada.

3 - Análise dos serviços e arquivos de parâmetro dos sistemas para determinar se o Telnet e outros comandos de *logon* remoto não seguros não estão disponíveis para o acesso que não utiliza console.

4 - Observação do *logon* de um administrador em cada sistema para verificar se o acesso às interfaces de gerenciamento baseadas na *Web* é criptografado com criptografia forte. Análise da documentação do fornecedor e questionamento frente aos funcionários, para verificar se a criptografia forte para a tecnologia utilizada está implementada, de acordo com as melhores práticas do setor e/ou recomendações do fornecedor.

ORIENTAÇÃO

2.1 Objetivando dar suporte e auxiliar na configuração de sistemas que possuem vulnerabilidades, as organizações de segurança criaram recomendações e orientações.

Exemplos:

www.nist.gov

www.sans.org

www.cisecurity.org

www.iso.org

Os padrões de configuração do sistema deverão ser mantidos atualizados para garantir que as deficiências recentemente identificadas sejam corrigidas antes de o sistema ser instalado na rede.

2.1.1 Considerando as necessidades de segurança de diferentes funções do servidor, como parte dos padrões de configuração do sistema e processos relacionados, as organizações devem garantir que as funções que exigem diferentes níveis de segurança não coexistam no mesmo servidor.

2.2.2 Resolução das implicações de segurança específicas, associadas a funções desnecessárias, através da inclusão nos padrões de proteção do servidor e processos acerca da remoção das funcionalidades sem importância, para que funções desconhecidas não corram risco de ser exploradas.

2.2 Para serem considerados com "criptografia forte", os protocolos reconhecidos pelo setor, com resistências de chave adequadas e gerenciamento de chave, devem estar corretos, conforme aplicável para o tipo de tecnologia utilizada.

3. Manter um programa de gerenciamento de vulnerabilidades

Vulnerabilidades de segurança são exploradas para obtenção de acesso privilegiado aos sistemas. Porém, muitas podem ser solucionadas pelos *patches* de segurança disponibilizados pelos fornecedores, e devem ser instalados pelas entidades que gerenciam os sistemas.

REQUISITOS DO PCI DSS

3.1 O desenvolvimento seguro de aplicativos internos e externos (incluindo acesso administrativo via *Web*) deve seguir as seguintes conformidades:

- De acordo com o PCI DSS;
- Baseados nos padrões e/ou melhores práticas do setor;
- Incorporação de segurança da informação no ciclo de desenvolvimento do software.

3.1.1 Revisão do código, antes da disponibilização para a produção ou clientes objetivando identificar possíveis vulnerabilidades no código, usando processos manuais ou automatizados. Devendo seguir as seguintes conformidades:

- Outras pessoas, além do próprio, autor e pessoas cientes das técnicas de análise de código, bem como das práticas de codificação seguras, possam analisar todas as alterações feitas no código;
- Garantir que, através da revisão de código, seja possível o desenvolvimento em acordo com as diretrizes de codificação segura;
- Implementação das correções, antes da liberação;
- Revisão dos resultados das análises dos códigos e aprovação pelo gerenciamento, antes da liberação.

3.2 Tratamento contínuo das novas ameaças e vulnerabilidades dos aplicativos *webs* voltadas ao público, para garantir a proteção contra invasões por meio dos seguintes métodos:

- Analisar, pelo menos anualmente, os aplicativos, por meio de ferramentas ou métodos, manuais ou automáticos, de avaliação de segurança das vulnerabilidades;
- Instalar um firewall, como exemplo de uma solução técnica automatizada que detecte e previna intrusões baseadas na *Web*.

PROCEDIMENTOS DE TESTE

3.1 Análise dos processos de desenvolvimento do software por escrito, para verificar:

1 - Se os processos foram baseados nos padrões e/ou nas melhores práticas do setor.

2 - Se a segurança da informação foi incluída durante o ciclo de vida.

3 - Se os aplicativos de software foram desenvolvidos de acordo com o PCI DSS.

4 - Se processos por escrito de desenvolvimento do software foram implementados, mediante o questionamento dos desenvolvedores do software.

3.1.1

1 - Análise dos procedimentos escritos do desenvolvimento do software, questionando os funcionários responsáveis, para confirmar se todas as alterações nos códigos dos aplicativos personalizados foram revisados usando processos manuais ou automáticos, conforme o definido anteriormente no ponto **3.1.1** dos requisitos de PCI DSS.

2 - Selecionar uma amostra de alterações recentes dos aplicativos personalizados, para verificação se o código do aplicativo está em conformidade com o item acima.

3.2 Aplicativos *web,* deve-se certificar se os métodos estejam implementados de acordo com o que se segue:

- Análise dos procedimentos documentados, questionando os funcionários e analisando os registros de avaliação de segurança da aplicação, para confirmar se todos os aplicativos web são analisados usando ferramentas ou métodos manuais ou automatizados de avaliação de segurança de vulnerabilidades.

- Análise da configuração do sistema, mediante questionamento dos funcionários, para verificação da existência de uma solução técnica automatizada para detecção e prevenção de intrusões baseadas na *Web* (por exemplo, um *firewall*).

ORIENTAÇÃO

3.1 Sem a implementação de uma proteção durante o processo de definição de requisitos, análise e teste de desenvolvimento de software, as vulnerabilidades podem se apresentar no ambiente de produção. Entender como os dados confidenciais são controlados pelos aplicativos, para que se possa identificar onde eles mais precisam ser protegidos.

3.1.1 Necessidade de haver um indivíduo com conhecimento e experiência nas técnicas de análise de código envolvido no processo de inspeção, de preferência que não seja o desenvolvedor do código, para que a revisão seja independente e objetiva. A importância da análise manual em complemento ao uso de ferramentas automatizadas, para inspeção de código, devido às limitações das ferramentas automáticas em identificar problemas na codificação. Logo, diante da dificuldade de se corrigir problemas em código depois de ele ter sido distribuído ou liberado para ambientes de produção, salienta-se a importância de corrigi-los antes da sua liberação.

3.2 Quando mal codificadas, as aplicações *web* podem ser alvo de invasores ocasionando exploração das vulnerabilidades. Os requisitos de análise de aplicativos ou instalação de firewalls destinam-se a reduzir esse número de comprometimentos de segurança.

4. Testar regularmente os sistemas e processos de segurança

Necessidade da regularidade de testes nos componentes do sistema, processos e softwares personalizados para assegurar que os controles de segurança constituirão um ambiente em transformação.

REQUISITOS DO PCI DSS

4.1 Realização de testes de penetração externos pelo menos anualmente após qualquer melhoria ou modificação significativa da estrutura ou nos aplicativos. Exemplos:

- Sistema operacional;
- Sub-rede adicionada ao ambiente;
- Servidor *Web*.

4.2 As vulnerabilidades exploráveis que forem encontradas durante o teste de penetração devem ser corrigidas, e o teste ser repetido para verificar as correções.

PROCEDIMENTOS DE TESTE

4.1

1. Deve ser feita a análise do escopo do trabalho e dos resultados do teste de penetração mais recente, para averiguar se os testes foram realizados conforme a metodologia definida, com regularidade, e após quaisquer alterações significativas.

2. Verificação se o teste foi realizado por um recurso interno qualificado ou um terceiro externo, e, caso seja possível, se há alguma dependência organizacional responsável pelo teste.

4.2 Análise dos resultados do teste de penetração, para verificar a correção das vulnerabilidades exploráveis detectadas, e se o segundo teste confirmou sua correção.

ORIENTAÇÃO

4.1 Testes regulares de penetração são medidas de segurança que ajudam a diminuir as chances de acesso indevido a sistemas por indivíduos não autorizados. Dependendo da configuração de um determinado ambiente, existirá uma diferenciação referente ao que se irá tratar em melhoria ou modificação significativa.
A realização dos testes de penetração, após melhorias e modificações na rede vai garantir que os controles supostamente implementados ainda estejam funcionando corretamente após o processo de alteração.

2.4 OPEN WEB APPLICATION SECURITY (OWASP)

A OWASP (*Open Web Application Security Project*) é uma organização sem fins lucrativos, a nível mundial, focada em melhoria de software e produção de conhecimento acerca de segurança. Ela opera como uma comunidade de profissionais que colabora para o desenvolvimento de ferramentas de software e documentação baseada em segurança de aplicações. Seu caráter aberto confere imparcialidade sobre as informações e práticas a indivíduos, empresas, universidades, agências governamentais e outras organizações. O presente estudo utiliza como base a metodologia de desenvolvimento seguro de um dos projetos da OWASP, o "*OWASP Development Guide Project*".

- Revisão e Inspeção de Manuais

Os manuais são todos e quaisquer documentos que dizem respeito à arquitetura de um software ou aplicação. Podem estar contidas informações acerca da tecnologia, definições de controle ou acesso ao sistema, requisitos funcionais ou não funcionais e padrões de desenvolvimento seguros caso tenham sido adotados. A OWASP (2014) recomenda que tais documentos relativos a segurança de pessoal, políticas e todos os processos organizacionais que envolvem o ciclo de desenvolvimento de software das empresas sejam passíveis de inspeções e revisões.

- Modelagem de ameaças

Segundo Sándor e Sebestyén-Pál (2017), modelagem de ameaça é um método que se baseia na identificação, categorização e classificação das ameaças de segurança de um objeto de análise, o qual irá lançar as bases para a definição de técnicas, métodos e algoritmos de mitigação de ameaças de segurança. Também permite que desenvolvedores possam identificar as ameaças antes mesmo do software estar pronto, de forma que se possa entender quais as possíveis ações de um atacante diante dos sistemas. Com base na OWASP (2017), a modelagem de ameaças também pode ser a avaliação de riscos acerca do desenvolvimento que irá permitir ao desenvolvedor mitigar as vulnerabilidades, com foco no desenvolvimento de soluções que recuperem o sistema após incidentes, ao passo que, também, é

uma ferramenta de projeto que pode demonstrar como a arquitetura do sistema irá funcionar diante de ataques.

- Revisão de código fonte

Revisão de código sugere que o desenvolvedor ou profissional de segurança faça uma análise do código fonte da aplicação, para identificação de problemas que possam comprometer um sistema ou aplicação *web*. De acordo com a OWASP, vulnerabilidades críticas podem não ser encontradas com qualquer outra forma de análise ou teste, por isso é importante promover uma verificação criteriosa do código fonte que compõe o sistema ou aplicação. Exemplos de problemas que podem ser detectados através da revisão de código seguindo as métricas sugeridas pela OWASP são:

- Erros na modelagem da lógica do negócio;
- Problemas de controle de acesso;
- Utilização de algoritmos criptográficos de forma.

- Teste de Penetração

Existem vantagens com relação à realização do processo de *pentesting.* Segundo a OWASP (2014), o tempo de execução do mesmo se torna menor pelo uso de algumas ferramentas automatizadas de teste, porém com a desvantagem de só ser possível a sua realização quando a aplicação ou o software já está em um nível de desenvolvimento avançado. Sendo assim, faz-se necessário que o mesmo já se apresente em execução, para que o teste possa ser feito. Segundo Weidman (2014), o teste de invasão ou *pentesting* é considerado uma simulação de ataques reais, visando avaliar os riscos associados a brechas de segurança em potencial. Corroboram com essa definição YADAV e NAVDETI (2014) quando afirmam que o processo também define tentativas de intrusão a aplicações, ou até mesmo nas redes de computadores, para avaliar a segurança de determinado alvo.

Engebretson (2013) define o teste de penetração como uma tentativa legal e autorizada de localizar e explorar com sucesso sistemas de computadores com o objetivo de torná-los mais seguros. Quando concluído adequadamente, o teste de penetração deve gerar recomendações específicas para a correção dos problemas

descobertos durante o procedimento de teste. Em Bishop (2007), o teste de penetração também se caracteriza com base nos próprios aspectos das falhas de segurança, onde os mesmos, a serem analisados podem ser acordados entre o demandante e o analista que realizará o procedimento, com o objetivo de verificar até que ponto houve dificuldade no processo de quebra de segurança e quais são os impactos e consequências de uma invasão real.

Além disso, o teste de penetração também é realizado em algumas circunstâncias para a obtenção de certificações junto a organismos reguladores onde, por fim, visam melhoria da infraestrutura organizacional e pessoal da instituição ou organização.

O processo também pode ser conhecido como:

- Pen Testing
- PT
- Hacking
- Ethical Hacking
- White Hat Hacking

Existem diferenças entre Testes de Penetração (*Penetration Testing*) e Avaliação de Vulnerabilidade (*Vulnerability Assessment*). Nesse âmbito, os autores Yadav e Navdeti (2014), Allen (2002), Engebretson (2013), e Press (2011) convergem acerca dessa diferenciação.

Com base nesses autores, pode-se definir a Avaliação de Vulnerabilidade como sendo o processo que faz a “revisão” dos serviços e sistemas, em busca de problemas de segurança, fazendo a avaliação e classificação das vulnerabilidades, também usando algumas ferramentas de automação, diferindo, assim, do teste de penetração, o qual realmente executa a exploração e ataques, para provar que tais problemas existem. Engebretson (2013), inclusive aborda o processo de avaliação de vulnerabilidades como uma das etapas utilizadas para concluir um teste de penetração. Cabe salientar o caráter limitado de alguns testes, o que não promove a garantia total e segurança de um sistema, ou prevenção de prováveis ataques. Porém, esses testes são importantes para identificar as vulnerabilidades a serem corrigidas reduzindo as chances de ataques bem-sucedidos, ao passo que devido à

descoberta de novas vulnerabilidades, eles se tornam obsoletos, sendo que devem ser realizados periodicamente.

Existem algumas padronizações acerca da forma de como um Teste de Penetração pode ser conduzido. Tanto a OWASP como outras comunidades de segurança, associações, órgãos e institutos, corroboram para o desenvolvimento de metodologias com a finalidade de servirem de guias para a realização desses testes.

Dentre elas podemos citar:

- ISSAF – (*Information System Security Assessment Framework*)
- *Pentest Frameworks*
- NIST SP800-115 e SP800-42
- OSSTMM – (*Open Source Security Test Methodology Manual*)
- OWASP *Testing Guide*

Segundo Assunção (2014), existem duas formas de se realizar testes de segurança, no que se refere à localização do analista testador. Ou seja, os testes podem ser feitos internamente ou externamente à rede da Instituição.

Existem três tipos de *Penetration Test*.

O critério de classificação se dá de acordo com o nível de conhecimento que o atacante possui acerca do “alvo” em questão.

- *Black Box* (Caixa Preta) – O analista não possui o conhecimento da infraestrutura a ser testada, de modo que, antes mesmo de dar início ao teste de invasão, ele terá que descobrir a dimensão e a localização dos sistemas de informação.
- *White Box* (Caixa Branca) – O analista conhece a infraestrutura e a aplicação a ser testada. Sendo também um tipo de teste que permite avaliar até que ponto a segurança pode estar comprometida internamente.

  Há casos em que o analista pode estar servido de diagramas de rede, códigos fonte de aplicações e tabelas de endereçamento IP.
- *Gray Box* (Caixa Cinza) – O analista, de certo modo terá conhecimento parcial acerca da infraestrutura ou sistema de informação a ser testado.

Figura 1 - Topologia de *Penetration Test.*

Fonte: Autoria própria, 2019.

- Relatório de Teste de Penetração

O relatório de Teste de Penetração é o documento produzido pelo analista, o qual irá conter todos os resultados, parciais ou totais, do seu teste, esteja em andamento ou já finalizado. Ele é o produto do processo de auditoria de segurança, e será entregue aos gestores. Nele constarão todas as vulnerabilidades, problemas ou as etapas realizadas, e recomendações para apoiar a mitigação das vulnerabilidades encontradas.

2.5 ORIENTAÇÕES E DEFESAS CONTRA SQL INJECTION

Segundo a OWASP Top 10 *Most Critical Web Application Risks* (OWASP, 2017), o SQL *Injection* lidera o *rank* das vulnerabilidades mais recorrentes do ano de 2017.

A Injeção de SQL consiste no envio de códigos não validados via campos de pesquisa (*login* e senha) das aplicações *web*, de modo que essa vulnerabilidade permite o preenchimento da estrutura de requisição SQL contida na aplicação, realizando as consultas ou modificações nos bancos de dados nos servidores *web* de modo não autorizado. Se realizada com sucesso, essa injeção permitirá a exploração da sintaxe do interpretador base, e realizará a execução de comandos de manipulação ou definição de dados baseado na concatenação de código, pondo em risco a integridade do banco de dados. Devido à negligência ou desconhecimento em relação à execução de um plano de segurança, sobretudo das metodologias de desenvolvimento seguro, a não validação dos dados recebidos nos campos de inserção – onde eles são passados diretamente para as consultas ao

banco, – é o motivo mais comum que faz com que essa vulnerabilidade ocorra em aplicações *web*.

- Tipos de Ataques SQL Injection:
    - *Poorly Filtered Strings*

  Envio de parâmetros diretamente para a consulta SQL.
    - *Incorrect type handling*

  Quando não ocorre a validação de um parâmetro que é enviado diretamente para a consulta SQL.
    - *Signature Evasion*

  Quando ocorre a evasão da assinatura de sistema para detecção de SQL *Injection*.
    - *Filter Bypassing*

  Quando filtros que impedem a inserção de caracteres que permitem o SQL *Injection* são burlados.
    - *Blind SQL Injection*

  Baseia-se no retorno de condições verdadeiras ou falsas diante da resposta do ataque ao sistema.

2.5.1 Defesas Primárias

Consultas dinâmicas ao banco de dados permitem a exploração da vulnerabilidade de SQL *Injection*. Quando uma aplicação *web* é desenvolvida de forma simples sem seguir ou desconhecendo orientações e/ou formas de proteção contra vulnerabilidades baseada na gestão de desenvolvimento seguro, essas consultas permitem a injeção de SQL através das entradas fornecidas pelo usuário. Segundo as recomendações da OWASP (2016), a proteção contra a Injeção de SQL se baseia em duas restrições que os desenvolvedores precisam conhecer e adotar:

- Rescindir rotinas que permitem consultas dinâmicas através do interpretador do banco de dados;
- Impedir que a entrada fornecida pelo usuário contenha trechos de SQL malicioso que afete a lógica da consulta a ser executada.

Para evitar ameaças oriundas da exploração de SQL *Injection,* e atendendo essas duas recomendações, a OWASP (2016) elenca quatro técnicas que podem ser utilizadas em praticamente qualquer tipo de linguagem de programação, e com qualquer tipo de base de dados. Existem outros bancos de dados que possuem características diferentes, a exemplo das bases de dados XML, que podem ser alvo de Injeção de *XPatch* e *XQuery*, onde, mesmo assim, essas técnicas podem ser usadas para a sua proteção. Porém, não fará parte do escopo do presente estudo abordar esse tipo de vulnerabilidade nessa variante de base de dados.

Algumas linguagens de programação possuem algumas especificidades acerca de como as seguintes técnicas poderão ser adotadas. Nesse estudo, as medidas adotadas baseiam-se na linguagem PHP, objetivando a proteção do código da aplicação *web*.

As técnicas de defesa primárias são:

1. **Uso de Instruções preparadas (com consultas parametrizadas);**
2. **Uso de procedimentos armazenados;**
3. **Validação de entrada na lista de permissões;**
4. ***Escaping* das entradas fornecidas pelo usuário.**

Tendo como defesas adicionais:

1. **Aplicação do mínimo de privilégios;**
2. **Execução da validação de entrada da lista de permissões como defesa secundária.**

Defesas Primárias:

1. **Uso de Instruções preparadas (com consultas parametrizadas)**

Consultas parametrizadas têm como base o uso de instruções preparadas com ligação variável, e, segundo a recomendação da OWASP, são como todos os desenvolvedores deveriam ser orientados a fazer as consultas nos bancos de dados. Essa técnica faz com que seja definido antecipadamente todo o código SQL, para que, só depois, cada um dos parâmetros para a consulta seja passado. Assim,

permitindo que o banco de dados possa fazer a distinção entre o código e dados, independente de cada entrada fornecida pelo usuário. Ao implementar essa técnica de proteção, o desenvolvedor garante que não se possa alterar a consulta de forma maliciosa, mesmo se os comandos sejam inseridos como entrada na aplicação *web*. Mesmo se for inserido a ID de um usuário; **admin' or '1' = '1** a consulta não estaria vulnerável, e apenas faria a busca do nome de usuário que correspondesse literalmente à sequência inteira: **admin' or '1' = '1.**

Na linguagem PHP, a recomendação específica seria:

- **PHP – Uso de PDO com consultas parametrizadas fortemente tipadas (usando bindParam()).**

Em raros casos, pode haver notáveis perdas de desempenho devido ao uso de declarações preparadas. Nesse caso, as recomendações da OWASP são;

a) Validar fortemente todos os dados;
b) Escapar de todas as entradas fornecidas pelo usuário fazendo uso de uma rotina específica do fornecedor do banco de dados.

2. Uso de procedimentos armazenados

A técnica de programação de procedimentos armazenados tem o mesmo efeito das consultas parametrizadas, porém nem sempre é segura contra a injeção de SQL.

Nela, o código SQL para um procedimento armazenado é definido e armazenado no próprio banco de dados, e, depois, chamado pela aplicação.

O procedimento armazenado é composto por uma instrução SQL que será pré-processada pela API do Sistema de Gerenciamento de Banco de Dados, que vai receber os parâmetros de forma separada da *string* de consulta principal.

Nessa *string* principal, os parâmetros serão marcadores que depois serão substituídos pelo SGBD por seus correspondentes valores, durante a execução da consulta.

Exemplo:

SELECT nome FROM cidade WHERE estado = ?

Parâmetro 1: nomeEstado

Neste caso, o valor da variável atribuída a “nomeEstado” será apenas usado como critério comparativo com a coluna de “Estado”.

Não obstante, se o valor recebido pela variável “nomeEstado” for a entrada; “ ' or 1 = 1 ”, o resultado da busca será literalmente o conteúdo da *string*; “ ' or 1 = 1 ”, fazendo com que o sentido original da consulta seja garantido.

Os procedimentos armazenados não incluem geração dinâmica de SQL de forma insegura, sendo que os desenvolvedores normalmente não geram SQL dinâmico dentro dos procedimentos armazenados. Mas, quando isso não puder ser evitado, deve-se usar validação de entrada ou *Escaping* apropriado, para ter a garantia de que todas as entradas que forem fornecidas pelo usuário para o procedimento armazenado não possibilitem a injeção de SQL. O uso da técnica de procedimentos armazenados também possui riscos, pois exige direitos de execução para que funcionem. Ou seja, caso um servidor seja acessado, o invasor terá todos os direitos sobre a base de dados, onde anteriormente ele só poderia ter acesso de leitura.

3. Validação de entrada na Lista de Permissões

O uso de variáveis de ligação, geralmente como nomes de tabelas, colunas ou indicadores de classificação, não é recomendado em partes das consultas SQL. Logo, a validação de entrada ou redesenho de consultas se faz como defesa mais recomendada. No caso das palavras que podem ser nomes de tabelas ou colunas devem vir relacionadas aos seus respectivos códigos, e não dos parâmetros do usuário. Porém, existe a possibilidade de os valores dos parâmetros do usuário serem utilizados para direcionar nomes de tabelas e colunas diferentes. Sendo assim, para garantir que a entrada não validada do usuário não termine na consulta, os valores dos parâmetros deverão ser mapeados para os nomes legais ou esperados da tabela ou coluna.

Quadro 1 - Exemplo de validação de tabela

```
String NomeTabela;
switch(PARAM):
  case "Valor1": NomeTabela = "fooTable";
                 break;
  case "Valor2": NomeTabela = "barTable";
                 break;
  ...
  default      : throw new InputValidationException("Valor inesperado fornecido"
                                                   + " para o nome da tabela");
```

Fonte: Autoria própria, 2019.

Dessa forma, o “NomeTabela” pode ser anexado à consulta SQL, porém com a ressalva que funções genéricas de validação de tabela podem acarretar perdas de dados. Nos casos de ordem de classificação, por exemplo, recomenda-se que haja um tratamento na entrada fornecida pelo usuário, de forma que ela seja convertida em um tipo booleano, e que ele seja usado para a seleção de um valor seguro a ser anexado à consulta, sendo mais uma técnica de segurança a ser implementada. Exemplo:

Quadro 2 - Exemplo de conversão booleana da entrada do usuário.

```
public String someMethod(boolean sortOrder) {
 String SQLquery = "some SQL ... order by Salary " + (sortOrder ? "ASC" :
"DESC");`
 ...
```

Fonte: Autoria própria, 2019.

4. *Escaping* das entradas fornecidas pelo usuário

Recomenda-se que essa quarta técnica apenas seja usada como último recurso, quando não existir a possibilidade da utilização das técnicas mencionadas anteriormente.

- Das anteriores, a validação de entrada é a melhor escolha.

- Dentre todas as técnicas mencionadas, o *Escaping* das entradas fornecidas pelo usuário é uma metodologia frágil, e não impede a Injeção de SQL em todas as situações. Essa técnica permite limitar ou barrar a entrada do usuário antes de ser usada em uma consulta. O MySQL pode dar suporte a dois tipos de escape:

1. **ANSI_QUOTES**: Modo SQL e um modo que podemos ativar ou desativar.
2. Modo **MySQL.**

No modo ANSI SQL é necessário codificar todos os caracteres “ ' ” (aspas simples) com “ '' ” (aspas duplas).

No modo MySQL da seguinte forma:

Quadro 3 - Listas de caracteres em ANSI.

NUL (0x00) --> \0	ASCII NUL (X'00')
BS (0x08) --> \b	backspace
TAB (0x09) --> \t	tabulação
LF (0x0a) --> \n	nova linha (avanço de linha)
CR (0x0d) --> \r	retorno de transporte
SUB (0x1a) --> \Z	ASCII 26 (Control + Z)
" (0x22) --> \"	aspas duplas
% (0x25) --> \%	porcentagem
' (0x27) --> \'	aspas simples
\ (0x5c) --> \\	barra invertida
_ (0x5f) --> _	underline
Todos os outros caracteres não alfanuméricos com valores ASCII menores que 256 -> \ c onde 'c' é o caractere original não alfanumérico.	

Fonte: INJECTION, OWASP. SQL Injection Prevention Cheat Sheet. 2016.

- Codificação hexadecimal de entradas

Essa técnica de *Escaping* permite que toda a entrada de usuário seja recebida e passe por um processo de codificação hexadecimal. À aplicação *web* vai ser atribuído o papel de:

- Codificar hexadecimalmente as entradas dos usuários antes de incluí-la na instrução SQL,
- Fazer com que a instrução SQL compare os dados levando tudo em consideração. Exemplo, se houver a necessidade da procura de um registro correspondente a um ID da sessão e o usuário fornecer a sequencia abc123 como o ID da sessão, a instrução será:

```
SELECT ... FROM session WHERE hex_encode(sessionID) = '616263313233'
```

Devido a variações no que diz respeito aos bancos de dados *hex_encode*, deverá ser substituído pelo recurso que se refere ao respectivo banco utilizado. Baseado nos códigos ASCII / UTF-8, a sequência “616263313233” é a versão codificada em hexadecimal da sequência recebida do usuário. E mesmo que o invasor tente transmitir uma sequência contendo caracteres especiais, a SQL resultante vai apenas conter dígitos numéricos ou letras de (a) a (f), fazendo com que os caracteres que possibilitem realmente a injeção SQL não possam ser concatenados na consulta SQL.

2.5.2 Proteção do Código PHP contra SQLi

A melhor maneira de impossibilitar a injeção de SQL é fazendo o uso das instruções preparadas e consultas parametrizadas. Baseado na linguagem PHP podemos ter duas opções:

1. Fazendo o uso do PDO:

```
$stmt = $pdo->prepare('SELECT * FROM funcionarios WHERE nome = :nome');
$stmt->execute(array('nome' => $nome));
foreach ($stmt as $row) {
}
```

2. Fazendo o uso do MySQLi (para MySQL):

```
$stmt = $dbConnection->prepare('SELECT * FROM funcionarios WHERE nome = ?');
$stmt->bind_param('s', $nome);
$stmt->execute();
$result = $stmt->get_result();
while ($row = $result->fetch_assoc()) {
}
```

2.5.3 Defesas adicionais

Além das defesas primárias recomendadas, as orientações da OWASP também mencionam algumas defesas que podem ser adotadas em caráter adicional para minimizar os danos de um ataque de SQL *Injection*. A recomendação baseada no último privilégio orienta que o desenvolvedor deve minimizar os privilégios atribuídos a todas as contas de dados, de forma a não atribuir direitos de acesso do tipo de administrador para as contas da aplicação.

Tais contas apenas devem ter acesso de leitura concedido às tabelas as quais precisam ter acesso de forma a não apresentarem qualquer direito às tabelas relacionadas à base de dados. Ou seja, apenas executando os procedimentos armazenados de que precisam. O acesso não autorizado é outra ameaça que pode afetar as bases de dados. Logo, reduzir os privilégios concedidos à aplicação *web* irá reduzir as chances descritas anteriormente de acesso não autorizadas. Mesmo que o invasor não se encontre fazendo o emprego de infiltrações destacadas ao SQL na investida. Outra recomendação é que o SGBD esteja em execução em uma

conta do sistema operacional que tenha privilégios restritos. Acerca das contas de usuário, não se deve fazer o uso da mesma conta de proprietário para se conectar à base de dados. Além disso, cada usuário deve ter acesso de seleção conforme apenas o necessário. Uma página de *login*, por exemplo, requer apenas o acesso de leitura aos campos de usuário e senha na tabela, e nunca de gravação em qualquer campo. E, por fim, a validação de entrada também pode ser uma defesa secundária usada para detectar entradas não autorizadas previamente à consulta SQL.

2.6 ESTUDOS CORRELATOS

A Segurança da Informação é fundamental para prevenção ponderada a infortúnios de ativos, vazamento ou alterações de informações e comprometimento dos serviços das organizações, ao passo que, também é uma forma de gestão. Os principais artigos utilizados citam o CERT para permitir um panorama inicial sobre os incidentes de segurança registrados ao longo dos anos. O CERT.br é o Grupo de Resposta a Incidentes de Segurança para a Internet no Brasil, sendo responsável por tratar incidentes de segurança. Costa (2016) reporta estatísticas do ano de 2016 em comparação aos dois anos anteriores, demonstrando um sensível aumento no número de incidentes de segurança.

As novas estatísticas, no que se refere aos anos de 2017 e 2018 apresentam um aumento nos incidentes de segurança, em relação a 2016, nos quais diferentes tipos se intercalam, caracterizando, assim, o exposto no tema abordado. Monteverde e Campiolo (2014) segmentam suas análises em duas etapas: A primeira, relacionada à identificação das vulnerabilidades, e, posteriormente, a segunda, referida à exploração. No transcorrer referente à identificação, os autores fazem utilização de ferramentas distintas, com o propósito de identificar possíveis vulnerabilidades a serem exploradas na etapa posterior. Será utilizada tal proposta de segmentação. Não obstante da análise referida às vulnerabilidades e suas particularidades. Os referidos autores identificam similaridade pela sua utilização da *OWASP Top 10.* Classificando, assim, referências relacionadas ao nível crítico das vulnerabilidades detectadas. Por intermédio de análise de etapas, posteriormente, será retratada a fase de exploração, à qual igualmente encontrar-se-á fracionada:

- Exploração de injeção de SQL;
- Quebra de gerenciamento de sessão;
- Configuração incorreta de Segurança;
- Exposição de dados sensíveis e componentes com vulnerabilidades conhecidas.

Segundo os autores, em suma, a alíquota dos resultados fez-se positiva. Podendo, assim, evidenciar tal situação do sistema analisado. Ademais de toda a bibliografia, os referidos anteriormente apresentam-se como únicos a implementar em tal método. Há uma necessidade de caráter instantâneo concernente à conscientização do mercado e principalmente, às empresas, refletindo de tal forma aos riscos envolvidos, ao passo que, caso haja explorações bem sucedidas, os prejuízos serão maximizados.

Relacionando-se de tal maneira, uma possível maximização em treinamento a desenvolvedores e a utilização das recomendações referentes à documentação abordada por meio da OWASP durante o processo de desenvolvimento, faz-se necessária. Farias (2009) expõe práticas que, podem ser utilizadas para limitar ou impedir a injeção SQL em aplicações *web*. Inclusive, salienta que se trata de pequenas ações que se configuram por baixo custo, e que podem ser de grande importância em nível de implementação. Na etapa prática, ele ainda apresenta um sistema *web* a título de exemplo o qual, se apresenta propositalmente vulnerável à Injeção de SQL, objetivando, através da sua utilização, demonstrar os efeitos práticos da implementação das boas práticas expostas no trabalho.

Já Callori (2007), realiza um levantamento sobre determinadas características de ataques por SQL *Injection*, porém, com enfoque nos presentes em três dos principais gerenciadores de banco de dados do mercado (SQL Server, MySQL e Oracle), onde propõe os melhores procedimentos a serem realizados para minimizar ataques dessa natureza. Lopes (2017) baseia sua abordagem em cinco objetivos específicos: apresentando, assim, em suma, as principais vulnerabilidades em aplicações *web*; os principais fundamentos relacionados a testes de invasão; adotando algumas das ferramentas mais utilizadas relacionadas à automatização do teste de penetração em aplicações *web* com SQL *Injection*; a metodologia para se

realizar esses testes e apresentar implementações de contra medidas com enfoque na proteção cibernética contra ataques SQL *Injection*.

Atingindo, assim, tais objetivos, o referido fez utilização do método observacional, uma vez que o trabalho teve base na identificação de aspectos essenciais de fenômenos ou eventos empíricos relativos ao teste de penetração com injeção SQL. Direcionando-se ao empregado de métodos comparativos, ao se identificarem as semelhanças e diferenças nas métricas e comandos necessários à realização da utilização da vulnerabilidade em códigos diversos, com auxílio de classes retratadas em sistemas de bancos de dados. Obtendo, como conclusão, recomendações durante o desenvolvimento de um código de aplicações com técnicas de tratamento de entrada de dados e outras contramedidas à vulnerabilidade SQL.

3 METODOLOGIA

A pesquisa bibliográfica baseou-se mediante consultas a livros, monografias, artigos e *paper's* sobre o tema em estudo. A presente pesquisa pode também ser classificada como aplicada, já que metodologias e ferramentas serão utilizadas para que através do protótipo conceitual, seja aplicada as orientações da OWASP de defesa contra o SQL *Injection* face às consequências da vulnerabilidade e as recomendações relativas à gestão de segurança, relacionando assim, ao desenvolvimento seguro de aplicações *web*. Objetivando desta forma, compreender melhor a importância da adoção de melhores práticas de desenvolvimento de aplicações web e da proteção contra ataques por injeção de SQL.

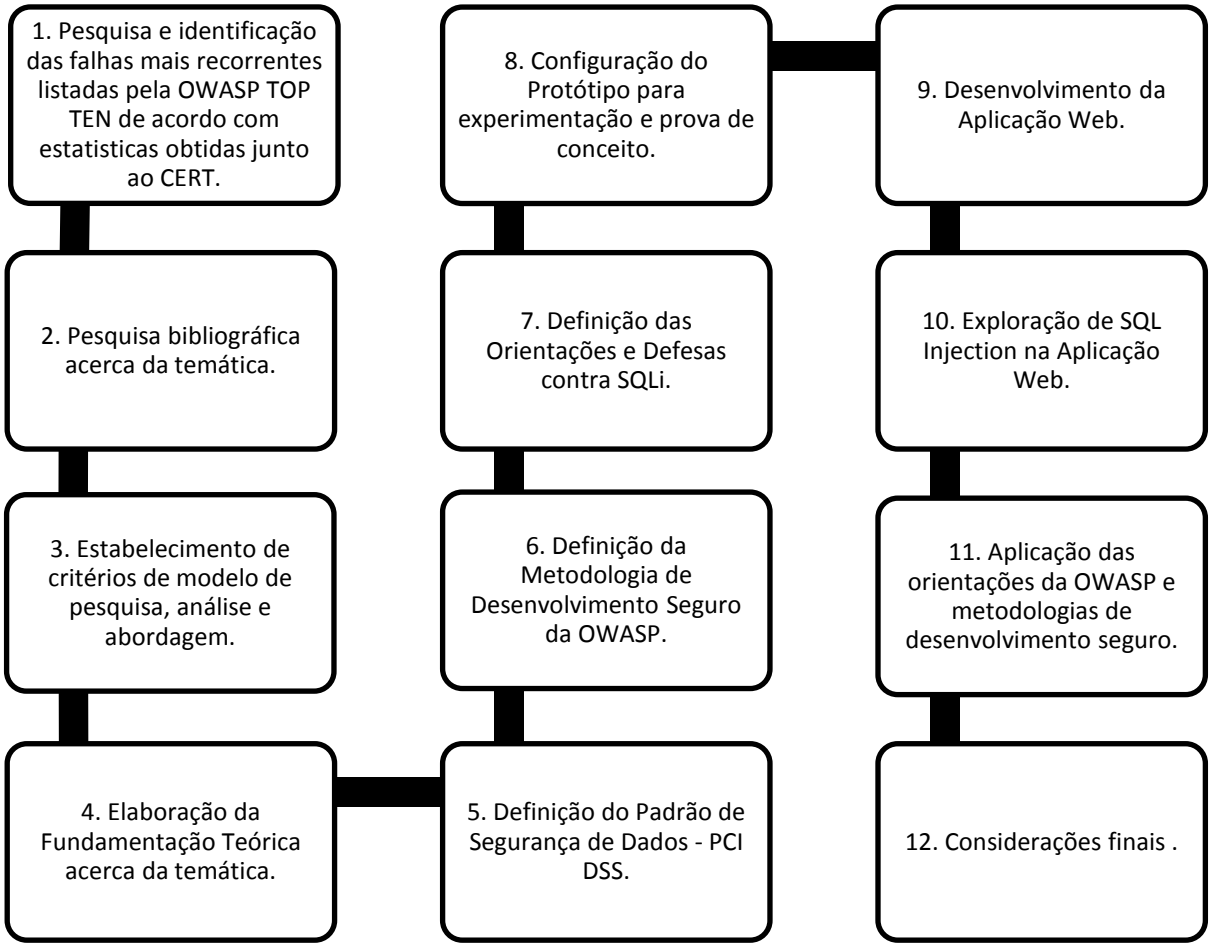
Objetivando o desenvolvimento de um ambiente didático e prático para a experimentação e aplicação dos conceitos apresentados nesse trabalho, foi desenvolvido um protótipo conceitual de aplicação *web*, contendo propositalmente a vulnerabilidade de Injeção de SQL. Foram adotadas, na sua implementação, e abordagens que tornaram fácil a execução dos procedimentos de teste/ataque, compreensão da implementação, vulnerabilidade e a correção das falhas.

O método observacional foi predominantemente utilizado, no qual, dá suporte à identificação de aspectos essenciais e acidentais de fenômenos ou eventos empíricos. Assim o trabalho terá como base na identificação de aspectos essenciais e acidentais de fenômenos ou eventos práticos relacionados a testes de invasão em

aplicações *web*. Como procedimentos abordados para a realização desta pesquisa, podem ser relacionados abaixo: Levantamento das referências pertinentes ao tema; seleção das fontes levantadas, mediante análise crítica; leitura e fichamento das informações obtidas; consolidação dos resultados obtidos, juntamente as estatísticas fornecidas pelo CERT.

O fluxograma a seguir ilustra o fluxo do estudo:

Figura 2 - Fluxograma do Estudo.



Fonte: Autoria própria, 2019.

4 DESENVOLVIMENTO

4.1 PROTÓTIPO PARA EXPERIMENTAÇÃO E PROVA DE CONCEITO

Com o objetivo de desenvolver de um ambiente para a experimentação e aplicação dos conceitos apresentados neste trabalho respaldados por todo referencial teórico metodológico, foi implementado um protótipo conceitual de aplicação *web,* contendo propositalmente, a vulnerabilidade de Injeção de SQL. Foram adotadas, no seu desenvolvimento, tecnologias e abordagens que tornem fácil a execução dos procedimentos de teste/ataque, a compreensão da implementação, vulnerabilidade e a correção das falhas.

1 Sistema Operacional Kali Linux

Kali Linux, disponível em <https://www.kali.org/>, é um sistema operacional baseado em um projeto de código aberto, mantido e financiado pela *Offensive Security*, uma provedora de serviços de treinamento em segurança e teste de penetração.

2 Sistema de Gerenciamento de Banco de Dados MySQL

MySQL, disponível em <https://www.mysql.com/>, é um sistema de gerenciamento de base de dados de código aberto, tendo se tornado opção popular de banco de dados para aplicativos baseados na *Web*.

3 Linguagem de desenvolvimento Web PHP

PHP, linguagem de programação de código fonte aberto, voltada para o desenvolvimento *Web*, com o intento de poder ser integrada a páginas HTML.

4 Servidor *Apache*

O Servidor HTTP *Apache,* disponível em <https://www.apache.org/>, é um servidor *web* de processamento de dados e execução de arquivos distribuídos, tendo se tornado opção popular de banco de dados para aplicativos baseados na *Web*. A aplicação *web* confeccionada, baseia-se na linguagem PHP, a qual irá trabalhar em conjunto com o servidor *Apache*. Baseando-se em Assunção (2015) chegamos à conclusão que tal combinação de soluções em Servidor *Web* e

linguagem *web* são a mais popular, onde corroboram para essa afirmação Assunção (2009) e a pesquisa realizada pela W3Techs[12], que confronta o Nginx e o *Apache*, que fica em primeiro lugar em termos de popularidade, além de ter independência de plataformas específicas, o que o torna possível de ser implementado em qualquer sistema operacional. O diagrama a seguir permite verificar a tendência histórica na porcentagem de sites que usam as duas soluções selecionadas.

Gráfico 1 - Comparativo de popularidade entre Apache e Nginx.

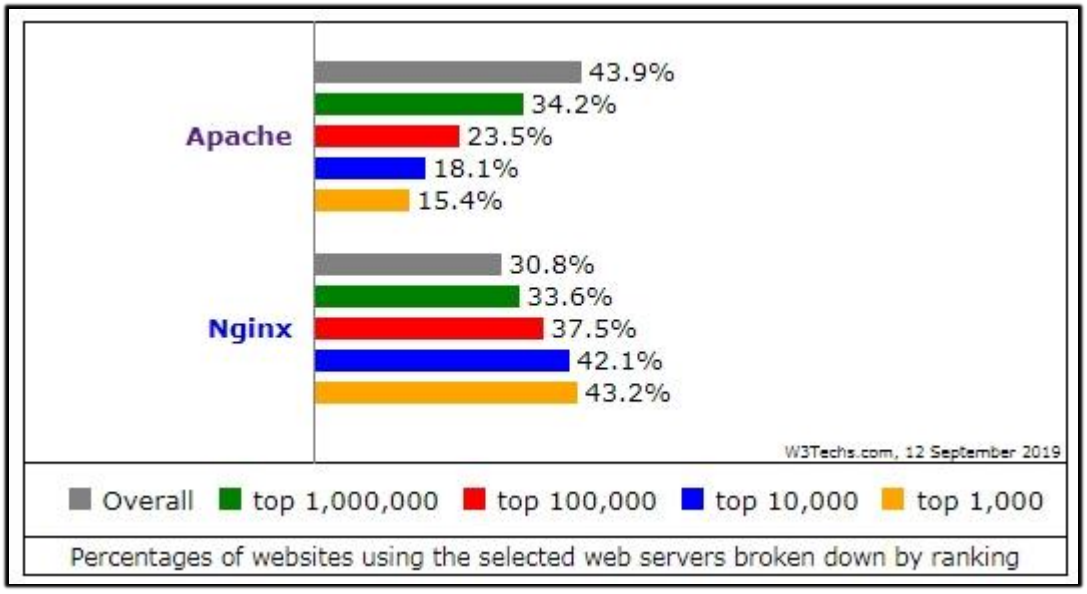
Fonte: https://w3techs.com/technologies/comparison/ws-apache,ws-nginx.

Acesso em 12 de Setembro de 2019.

O diagrama abaixo mostra as porcentagens de sites que usam as tecnologias selecionadas, discriminadas por classificação.

[12] Disponível em: <https://w3techs.com/technologies/comparison/ws-apache,ws-nginx>. Acesso em 12 de Setembro de 2019.
A W3Techs é uma divisão dos Serviços baseados na *Web* Q-Success. O objetivo é coletar informações sobre o uso de vários tipos de tecnologias usadas na criação e execução de sites e produzir e publicar pesquisas que fornecem informações sobre esse assunto.

Gráfico 2 - Classificação por ranking.



Fonte: https://w3techs.com/technologies/comparison/ws-apache,ws-nginx . Acesso em 12 de Setembro de 2019.

O *Apache* é usado por 43,9% de todos os sites cujos servidores conhecemos.
O *Apache* é usado por 34,2% de todos os sites cujos servidores conhecemos e que figuram no top 1.000.000.

- Conexão com o Banco de Dados

Todos os comandos que se referem ao acesso ao banco de dados MySQL são disponibilizados no seu site oficial <www.mysql.com >.

- Conexão com o Servidor

No MySQL utilizamos o comando de conexão; *mysqli_connect,* tendo a seguinte sintaxe;

```
$conexao = mysqli_connect (“localhost”, “user” , “senha”);
```

Existirá um valor, do tipo inteiro, retornado por esse comando, sendo armazenado na variável **$conexao**, para uso posterior.

O primeiro parâmetro a ser informado é o endereço onde se localiza o servidor(host), que vem acompanhado da porta de conexão. Na maioria dos casos,

é a porta: 80. O segundo e o terceiro parâmetros são o usuário (*login*) e a senha para conexão, respectivamente.

- Seleção do Banco de Dados

Com o comando; *mysqli_select_db*, uma vez conectado ao servidor, poderemos escolher o banco de dados desejado.

```
mysqli_select_db ($conexao , “NomeDoBanco”);
```

Em caso de falha, esse comando retorna o valor “0”, e em caso de sucesso retorna o valor “1”.

Como primeiro parâmetro, temos o nome da base de dados e em seguida, o identificador da conexão.

Todas as consultas realizadas depois da execução desse comando irão utilizar a base de dados selecionada.

- Consultas SQL

Consultas são o modo de interação com o Servidor MySQL após o estabelecimento da conexão e a seleção da base de dados a ser utilizada.

Baseado em SQL (*Structured Query Language*), o comando *mysql_query* vai usar a seguinte sintaxe:

```
mysqli_query ($conexao , “$consulta”);
```

O primeiro parâmetro a ser informado é a *string* de consulta, e o segundo é referente à conexão com o servidor, respectivamente.

Em caso de falha, esse comando retorna o valor “0”, e em caso de sucesso retorna o valor “1”.

Porém, se o comando a ser executado na *string* for um *SELECT*, o valor de retorno pode ser armazenado em uma variável $resultado.

Exemplo;

```
$consulta = “SELECT * FROM NomeDoBanco WHERE usuario= '$usuario' AND senha= '$senha'”;
```

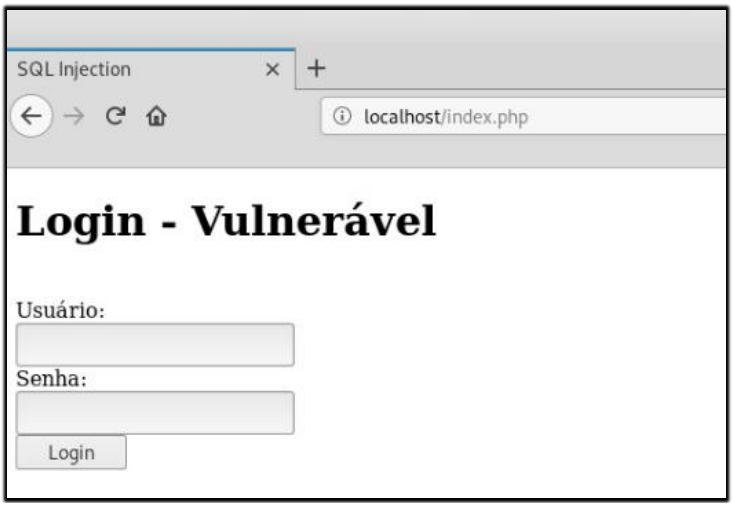
O valor pode ser tido como;

```
$resultado = mysql_query($conexao, $consulta);
```

4.1.1 Desenvolvimento da Aplicação Web

Subsequentemente, consta exposto o formulário para a realização de *login.* Conforme descrito anteriormente, contendo a necessidade da realização de informes (nome de usuário e senha), para que, a autenticação no sistema, de acordo com as credenciais gravadas no banco de dados, seja realizada. Encontrando-se, assim, dois campos de entrada que foram propositalmente configurados para serem vulneráveis à inserção de código SQL, o qual consente a exploração, desvios de autenticação e definição de comandos de manipulação e definição de dados.

Figura 3 - Tela de login vulnerável



Fonte: Autoria própria, 2019.

Trecho de código referente ao formulário de *login*:

Quadro 4 - Código HTML do formulário de login vulnerável.

```
<form action="index.php" method="POST">
  <h1>Login - Vulnerável</h1><br>
        Usuário:<br>
       <input type="text"
              name="usuario"<br><br>
        Senha:<br>
       <input type="text"
              name="senha"<br><br>
       <input type="submit" value="Login">
</form>
```

Fonte: Autoria própria, 2019.

O formulário de *login* fará a interação com o código a seguir, de forma que o arquivo *index.php* receberá, do formulário, as variáveis **$usuário** e **$senha**, fazendo, respectivamente, a conexão com a base de dados, gerando a instrução SQL e, por fim, verificando o resultado da consulta.

Quadro 5 - Código PHP do formulário de login.

```
1. <?php
2. ini_set("display_errors",1);
3.
4. if($_SERVER['REQUEST_METHOD'] == 'POST') {
5. $nc = mysqli_connect('localhost','root','1111');
6. mysqli_select_db($nc,'injection');
7.
8. $usuario = $_POST['usuario'];
9. $senha = $_POST['senha'];
10. $sql = "SELECT * FROM login WHERE usuario = '$usuario' AND
    senha='$senha'";
11. $query = mysqli_query($nc, $sql);
12.
13. if(mysqli_num_rows($query)==1) {
14.     header("location:admin.php");
15. }
16.
17. else {
18.     echo "Usuário ou senha inválidos";
19. }
```

```
20.}
21.?>
```

Fonte: Autoria própria, 2019.

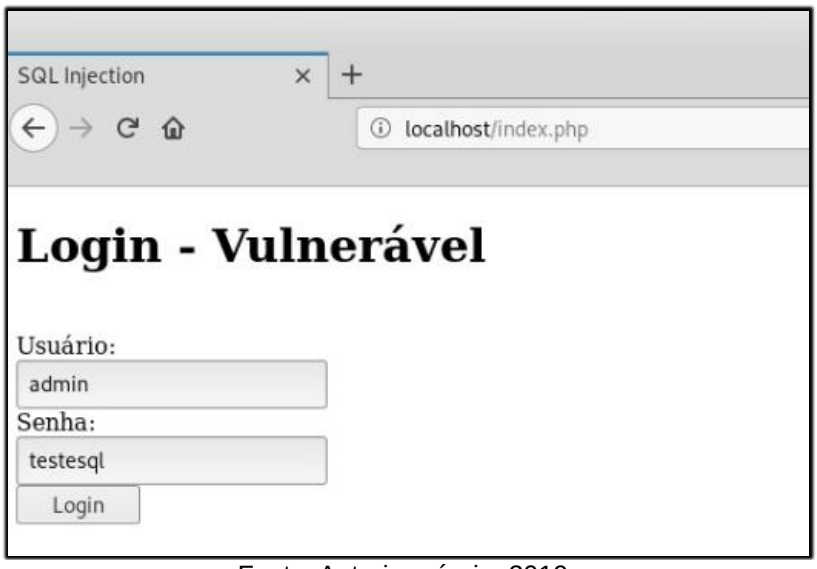
O código fonte completo da aplicação, desenvolvida com ausência de métodos de defesa contra o SQL *injection,* consta no **Apêndice "A"** deste trabalho.

4.1.2 Exploração de SQL Injection em aplicação vulnerável

Na **figura 4**, temos a tela de *login* da página *web,* na qual a consulta ao banco de dados é realizada da seguinte forma:

```
$sql = "SELECT * FROM login WHERE usuario = '$usuario' AND senha='$senha'";
```

Figura 4 - Tela de login vulnerável com credenciais válidas.



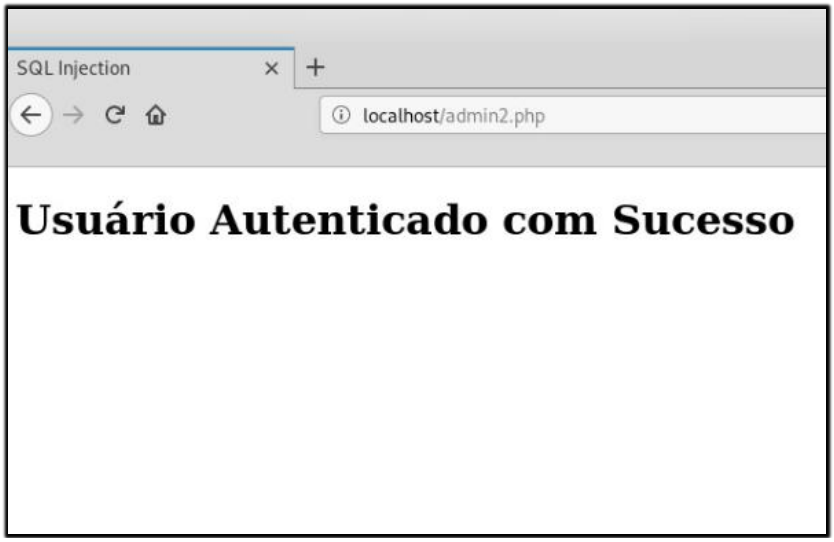
Fonte: Autoria própria, 2019.

Perante as entradas informadas, a requisição é enviada ao servidor da aplicação *web,* informando conteúdo inserido das caixas de *login* e senha da seguinte forma:

```
$sql = "SELECT * FROM login WHERE usuario = 'admin' AND senha='testesql";
```

Nesse caso, a rotina de código faz a utilização direta desses dados de entrada, sem a realização de nenhum tratamento no comando SQL acima.

Figura 5 - Acesso à aplicação web por meio de credenciais válidas.

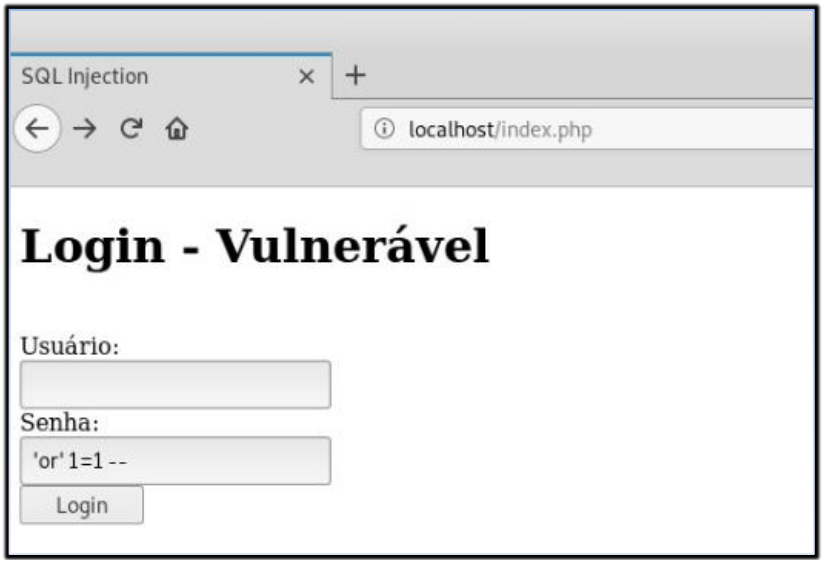


Fonte: Autoria própria, 2019.

Conforme a **figura 6**, a seguir, pode ser constatado que a inserção da expressão “ ' or 1=1-- ” faz a expressão se tornar verdadeira, de forma que:

```
$sql = “SELECT * FROM login WHERE usuario= “ ” AND senha= “' or 1=1--”;
```

Figura 6 - Tela de login vulnerável com código de injeção SQL.



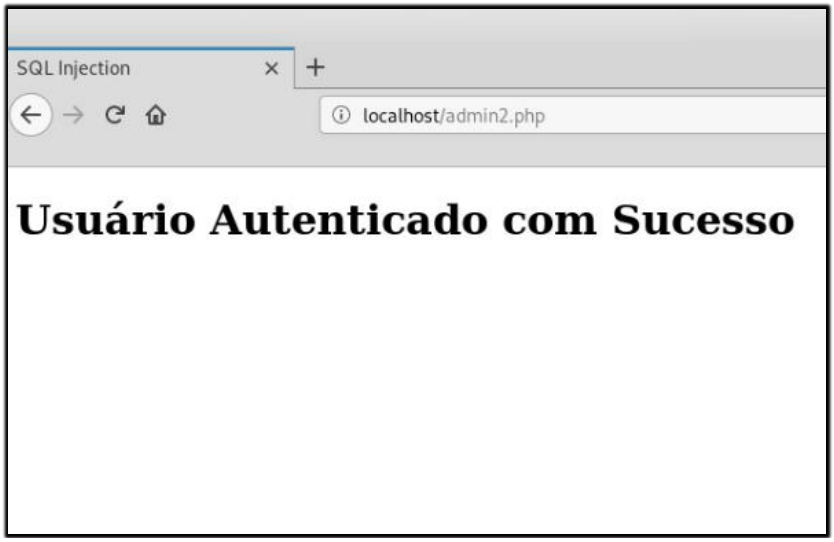
Fonte: Autoria própria, 2019.

Em concordância ao exposto, um dos campos de requisição é concluído com esse código, que permitirá o sucesso da implantação SQL, pois, após a expressão 1=1 (um é igual a um), será confirmada, retornando, assim, todos os usuários, e o

comando “ -- ” fará com que todo o código existente depois da inserção dele seja ignorado, devido à sintaxe da linguagem.

Ao retornar todos os usuários, a aplicação realizará a autenticação de um dos usuários, fazendo com que o atacante tenha o acesso à aplicação, como mostra a **figura 7**. Acesso a aplicação web por meio de injeção de código SQL.

Figura 7 - Acesso à aplicação web por meio de injeção de código SQL.



Fonte: Autoria própria, 2019.

4.1.3 Aplicação das orientações da OWASP e metodologias de desenvolvimento seguro

O código apresentado a seguir, consta da tela de acesso da aplicação, onde se implementaram as orientações referentes ao uso de instruções preparadas em PDO. De forma que a consulta não seja criada de forma dinâmica, para evitar que ela seja modificada.

Quadro 6 - Código PHP do formulário de login seguro.

```
1. <?php
2.
3. if($_SERVER['REQUEST_METHOD'] == 'POST') {
4. $usuario = $_POST['usuario'];
5. $senha = $_POST['senha'];
6.
7. $root = 'root';
```

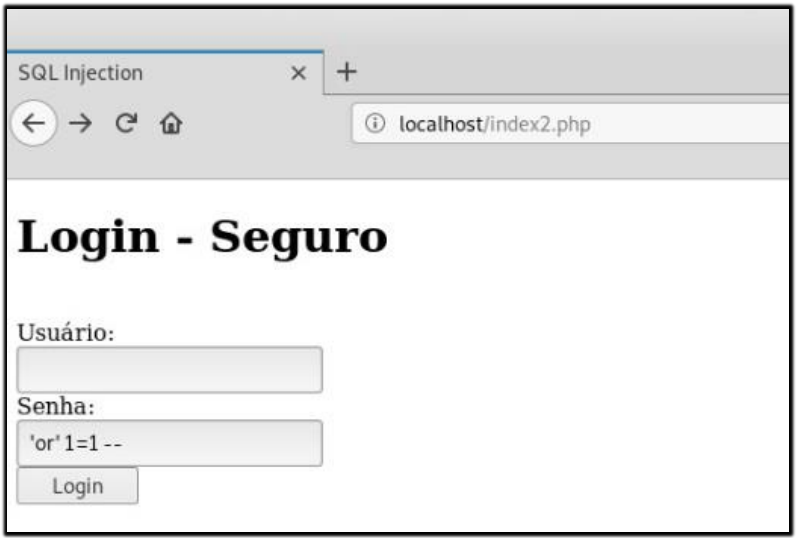
```
8. $password= '1111';
9. $conn = new PDO('mysql:host=localhost;dbname=fail', $root, $password);
10.
11. $sql = "SELECT * FROM login WHERE usuario=:usuario AND senha=:senha;";
12.      $stmt = $conn->prepare( $sql );
13.      $stmt-> bindParam( ':usuario', $usuario );
14.      $stmt-> bindParam( ':senha', $senha );
15.      $stmt->execute();
16.      $result = $stmt->fetchAll();
17.
18. if (count($result)) {
19.    header("location: admin2.php");
20. }
21. else {
22.    throw new Error($stmt->errorInfo());
23.
24. }
25. }
26.
27. ?>
```

A conexão com o banco de dados é realizada entre as linhas 7 e 9, tendo uma instanciação da classe PDO na linha 9. Nas linhas 4 e 5, são recebidas as variáveis **$usuario** e **$senha**. O método ***prepare()***, usado para preparar a consulta SQL, vai ser definido na linha 12

Neste caso, as variáveis não vão diretamente para a consulta. No lugar delas, teremos espaços reservados para os parâmetros a serem escapados de forma automática pelo driver do PDO, mediante a função ***bindParam()*** na linhas 13 e 14. Após o tratamento nas linhas anteriores, na linha 15, o comando SQL será executado ***$stmt->execute();***, verificando o resultado da consulta e dando acesso à aplicação através da confirmação das credenciais cadastradas, e fazendo o redirecionamento para o arquivo ***admin2.php,*** que contém a página da aplicação.

Caso seja feita a injeção de código SQL no formulário de *login,* como na **figura 8**, a aplicação fará o devido tratamento, não permitindo a exploração da vulnerabilidade.

Figura 8 - Tela de login segura com código de injeção SQL.



Fonte: Autoria própria, 2019.

O código fonte completo da aplicação, desenvolvida com a presença de métodos de defesa contra o SQL *injection,* consta no **Apêndice "B"** deste trabalho.

Não fará parte do escopo do presente estudo abordar esse tipo de vulnerabilidade na variante via URL. Sendo que o único cenário de ataque diz respeito apenas ao formulário de *login.* Mas, fazer o tratamento adequado, objetivando a solução do problema, não é difícil. No caso, recomenda-se que se faça o tratamento da variável recebida, de modo que o seu valor seja, obrigatoriamente um número inteiro, através do processo de *casting*, impossibilitando, assim, a Injeção pela URL da aplicação. Exemplo: ***$id = (int) $_GET['id'];***

5 CONSIDERAÇÕES FINAIS

O presente estudo encontra-se em um cenário onde instituições demonstram negligência ou desconhecimento acerca de práticas de segurança nos processos da organização, elementos de sistemas que participam do processamento de dados ou desenvolvimento seguro de aplicações *web*. Logo, a necessidade da elaboração de uma pesquisa nesse contexto é nítida, haja vista que, entre as contribuições desse trabalho, destacam-se a identificação da vulnerabilidade de injeção de código como a técnica mais difundida, com o maior quantitativo e incidência de danos relativos a sua exploração, e salientar a importância da gestão do plano de segurança no desenvolvimento seguro de aplicações *web,* baseado nas orientações e defesas contra o SQL *Injection,* propostas pela OWASP, sabendo que a maioria da

bibliografia que aborda essa temática tem como foco métodos de detecção de SQL *Injection,* através de uso de ferramentas automatizadas.

As áreas de Gestão e Segurança da Informação lançam as bases para o desenvolvimento desse estudo, pois foram abordados conceitos, métodos, e procedimentos pertinentes às respectivas áreas de conhecimento, haja vista que o referencial teórico metodológico presente no capítulo dois permite a compreensão acerca da temática. Mediante as recomendações da OWASP, mostrou-se importante a utilização da metodologia de codificação segura na elaboração de aplicações voltas a *Web*.

Desenvolvendo um protótipo para experimentação e prova de conceito, pode-se, de forma prática, explorar a vulnerabilidade, aplicar os conceitos apresentados, e demonstrar que, com a adoção dessas metodologias e orientações, a incidência de ameaças de segurança dessa natureza torna-se significativamente menor, principalmente quando alinhadas ao padrão de segurança de dados recomendado, mediante exposto.

5.1 LIMITAÇÕES DO TRABALHO E PESQUISAS FUTURAS

A referida pesquisa abordou a vulnerabilidade SQL *Injection,* implementando-a como prova de conceito, em um cenário de ataque ao qual sua exploração se deu por uma aplicação composta por uma tela de *login* com a falha propositalmente configurada, para acesso a uma aplicação *web*. Consta que esse cenário não é o único possível. A injeção de código SQL possui algumas variantes, inclusive a que diz respeito ao *Blind* SQL, que, inclusive, possui outras metodologias de mitigação que não constam no escopo da nossa abordagem.

Não houve o uso de nenhuma ferramenta automatizada para o escaneamento das falhas, pois as metodologias e recomendações para resolvê-las foram utilizadas no cenário configurado onde possibilitaram a aplicação da prova de conceito, justificando, assim, a importância de sua adoção frente à resolução das falhas de codificação.

Para trabalho futuros, poderemos fazer uma avaliação de *frameworks* e WAFs, de forma que eles possam ser usados de forma complementar às recomendações e orientações de desenvolvimento seguro, para promover maior

proteção, não só acerca das vulnerabilidades de SQL *Injection*, mas de outras falhas que eles possam detectar e impedir. E como projeto futuro, tratando-se de proteção contra injeção de código, verificar a possibilidade da criação de uma ferramenta automatizada de escaneamento de vulnerabilidades.

REFERÊNCIAS

ALLEN, Lee. **Advanced Penetration Testing for Highly Secured Environments**: The Ultimate Security Guide. Birmingham, UK: Packt Publishing Ltd, 2012.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR ISO/IEC 27002:2005 tecnologia da informação - técnicas de segurança - código de prática para gestão da informação. Rio de janeiro: 2005, Disponível em: <http://www.cienciianasnuvens.com.br/site/wp-content/uploads/2014/09/215545813-ABNT-NBR-177991.pdf > Acesso em 15 nov. 2019.

ASSUNÇÃO, Marcos Flávio Araújo: **Honeypots e Honeynets**. Florianópolis: Visual Books, 2009.

_____ **Análise de eficiência na detecção de vulnerabilidades em ambientes web com o uso de ferramentas de código aberto**. 2015. V. I. (Mestrado em Gestão de Sistemas de Informação e Gestão do Conhecimento). FUMEC: Belo Horizonte – MG, 2015. Disponível em: <http://www.fumec.br/revistas/sigc/article/view/3396/1871>Acesso em: 27 Nov. 2019.

_____ **Segredos do hacker ético**. 5ª ed. Florianópolis: Visual Books, fev. 2014. Disponível em: <https://docero.com.br/doc/ve8evv> Acesso em: 27 Nov. 2019.

BARRET, Diane; KING, Todd. **Redes de Computadores**. Editora LTC, 2010.

BASTA, Alfred; BASTA, Nadine; BROWN, Mary. **Segurança de computadores e teste de invasão**. Trad: Lizandra Magon de Almeida. 2ª ed. São Paulo: Cengage Learning, 2014. Disponível em: <https://docero.com.br/doc/55sv15> Acesso em: 27 Nov. 2019.

BISHOP, Matt. About penetration testing. **IEEE Security & Privacy**, v. 5, n. 6, p. 84-87, 2007. Disponível em: <https://dl.acm.org/citation.cfm?id=1340109> Acesso em: 27 Nov. 2019.

CALLORI, Felipe. **Estudo de Técnicas de Proteção Contra Invasões por SQL** Injection, 2007. V. I. (Bacharelado em Ciência da Computação). UFMGR, 2007. Disponível em: <https://docplayer.com.br/903123-Estudo-de-tecnicas-de-protecao-contra-invasoes-por-sql-injection-felipe-callori.html.> Acesso em: 26 mar. 2019.

CORREA, Sérgio. OSSEC HIDS – Instalação e configuração no CentOS 6.5. 2014. Disponível em: <https://www.vivaolinux.com.br/artigo/OSSEC-HIDS-Instalacao-e-configuracao-no-CentOS-65> . Acesso em: 14 nov. 2019.

CÔRTE, Leandro. **Método para a avaliação de servidores WWW no ambiente corporativo. 2002**. (Dissertação de Mestrado em Informática) – UFRGS, Porto Alegre, 2002. Disponível em: <https://lume.ufrgs.br/handle/10183/3375> Acesso em: 13 abr. 2019.

COSTA, José Victor Pereira. **Análise de Vulnerabilidades de Segurança em Portais de Governos Eletrônicos**, 2017. V.I (Bacharelado em Ciência da Computação), UFUB – Universidade Federal de Uberlândia, Uberlândia, MG, 2017. Disponível em: <https://repositorio.ufu.br/bitstream/123456789/20400/6/AnaliseVulnerabilidadesSeguranc%CC%A7a.pdf.> Acesso em: 19 mar. 2019.

CRUZ, Carlos Magno Bispo Rosal da. **Auditoria de segurança da informação em sistemas e aplicações**. 2017. Disponível em: <http://repositorio.unb.br/bitstream/10482/25258/1/2017_CarlosMagnoBispoRosaldaCruz.pdf.> Acesso em 12 de out. 2019.

DANTAS, Marcus. **Segurança da Informação**: Uma Abordagem Focada em Gestão de Riscos. Recife: Livro Rápido-Elógica, 2011. Disponível em: <http://www.marcusdantas.com.br/files/seguranca_informacao.pdf> p.11-13. Acesso em: 26 mar. 2019.

DSS, PCI: Requisitos e Procedimentos da Avaliação de Segurança. v. 3, Nov. 2013. Disponível em: <https://pt.pcisecuritystandards.org/_onelink_/pcisecurity/en2pt/minisite/en/docs/PCI_DSS_v3.pdf.> Acesso em: 23 nov. 2019.

ENGEBRETSON, Patrick. **The basics of hacking and penetration testing**: ethical hacking and penetration testing made easy. Waltham: Elsevier, 2011. Disponível em: <http://index-of.co.uk/Hacking-Coleccion/167%20-%20The%20Basics%20Of%20Hacking%20And%20Penetration%20Testing%20%20Ethical%20Hacking%20And%20Penetration%20Testing%20Made%20Easy%20%5B-PUNISHER-%5D.pdf> Acesso em: 27 Nov. 2019.

ELMASRI, Ramez et al. **Sistemas de banco de dados**. 2005.

FARIAS, Marcelo Bukowski de. Injeção de SQL em aplicações Web: Causas e prevenção, 2009. V. I. (Graduação em Ciência da Computação). UFRGS – Porto Alegre – RS, 2009. Disponível em: <https://www.lume.ufrgs.br/bitstream/handle/10183/18573/000730977.pdf?sequence=1>. Acesso em: 17 mar. 2019.

INDEX, Breach Level. 2017: The Year of Internal Threats and Accidental Data Breaches. Gemalto 2017 Annual BLI Report, 2018.

INJECTION, OWASP. SQL **Injection Prevention Cheat Sheet**, 2016. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html> Acesso em: 20 nov. 2019.

ISO/IEC 17799. Tecnologia da Informação – Código de Prática para Gestão da Segurança de Informações, 2000. Disponível em: <https://pt.scribd.com/document/818921/Norma-ISO-IEC-17799-2000-GESTAO-DE-SEGURANCA>Acesso em: 20 mai. 2019.

LINS, Lucas de Almeida. **Avaliação de Scanners de Vulnerabilidade**: dez riscos mais críticos em avaliações web. 2017. v. I (Bacharelado em Ciências de Informação). UFRJ, RJ, Jul. 2017. Disponível em: <http://bsi.uniriotec.br/tcc/textos/201707LucasLins.pdf.> Acesso em: 25 jul. 2019.

LOPES, Aristides. Testes de Invasão em Aplicações Web Utilizando SQL Injection: Um estudo epistemológico. Disponível em: https://repositorio.uniceub.br/jspui/bitstream/235/12401/1/51500400.pdf Acesso em: 26 mar. 2019.

MONTEVERDE, W.A.; CAMPIOLO, R. Estudo e Análise de Vulnerabilidades Web. p. 415-423, 2014.

OLIVEIRA, Maria Engel de. **Orkut:** O Impacto da Realidade da Infidelidade Virtual. In: PONTIFÍCIA UNIVERSIDADE CATÓLICA (Brasil). O Surgimento da Internet. [Rio de Janeiro, RJ]: Pontifícia Universidade Católica, 2007. Disponível em: <https://www.maxwell.vrac.puc-rio.br/9888/9888_4.PDF> p. 1-2. Acesso em: 29 mar. 2019.

OWASP, Top 10-2017: The Ten Most Critical Web Application Security Risks. **OWASP,** mar. 2018. Disponível em: <https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf> Acesso em: 17 mar. 2019.

PRESS, EC-Council. **Penetration Testing Procedures & Methodologies**. Course Technology Cengage Learning, p. 2. 2011.

RACCIATTI, Hernán. **Técnicas de SQL Injection**: Un Repaso , 2002. 2013. versão online. Disponível em: <https://www.redeszone.net/app/uploads/Tecnicas-de-SQL-Injection.pdf>.Acesso em: 19 mar. 2019.

RECOMMENDATION, X.800, **Security Architecture for Open Systems Interconnection for CCITT Applications**. International Telecommunication Union (ITU), 1991. Disponível em: <https://www.itu.int/rec/T-REC-X.800-199103-I>

ROCHA, Cerli Antônio da. **Desenvolvendo web sites dinâmicos**: PHP, ASP. JSP. Rio de Janeiro: Campus, v. 210, n. 2, 2003. Disponível em: <https://vdocuments.mx/desenvolvendo-websites-dinamicos-php-asp-jsp.html.>

SÁNDOR, H.; SEBESTYÉN-PÁL, G. **Optimal security designin the Internet of Things.** 2017 5th International Symposium on Digital Forensic and Security (ISDFS). P. 1-6, 2017.

SANTOS, Marcos Vinicius Soares; FERREIRA, Diorgenes; RODRIGUES, Marcos Paulo Maia; MOREIRA, Renato César Oliveira. **Segurança em Banco de Dados: Uma visão geral sobre segurança e suas principais deficiências**. In: 8º FÓRUM FEMEG UNIVERSIDADE: Saberes e Práticas Inovadoras. n. 8. 24-27 Set. 2014. Campus Universitário Prof. Darcy Ribeiro. Montes Claros, MG, 2014. Disponível em: <http://www.fepeg2014.unimontes.br/sites/default/files/resumos/arquivo_pdf_anais/s

eguranca_em_banco_de_dados_uma_visao_geral_sobre_seguranca_e_as_principais_deficiencias_na_seguranca.pdf.> Acesso em 23 mai. 2019.

SEBBEN, VICTOR HUGO FOLCHINI. **Segurança em Sistemas WEB:** Uma análise da extensão PDO como forma de proteção de sistemas PHP contra injeções de SQL. Passo Fundo, v. 57, 2014. (Tecnologia em Sistemas para Internet). IFSUL, Passo Fundo, RS, 2014. Disponível em: <https://painel.passofundo.ifsul.edu.br/uploads/arq/20160331163722187465530 1.pdf .> Acesso em: 07 mai. 2019.

SHIREY, Robert. RFC 2828: Internet security glossary. **The Internet Society**, v. 13, 2000. Disponível em: <https://www.rfc-editor.org/info/rfc2828> Acesso em 07 set. 2019.

STALLINGS, William; BRESSAN, Graça; BARBOSA, **Akio.Criptografia e segurança de redes**. Pearson Educaión, 2008.

TANENBAUM, Andrew Stuart. **Redes de Computadores**. Rio de Janeiro: Campos, 1994.

TATROE, Kelvin et al Programming PHP: Creating Dynamyc Web Pages. “O’Reilly Media, Inc.”, 2013.

WEIDMAN, Georgia. **Testes de invasão**: Uma introdução prática ao hacking. 1ª ed. Trad: Lúcia Kinoshita. São Paulo: Novatec Editora, 2014.

YADAV, Sushilkumar; NAVDETI, Chandrakant. Survey: Secured techniques for vulnerability assessment and penetration testing. IJCSIT) **International Journal of Computer Science and Information Technologies**, v. 5, n. 4, p. 5132-5135, 2014.

APÊNDICE A – Código fonte da aplicação web desprovido de métodos de defesa contra o SQL *Injection.*

```
1.  <?php
2.  ini_set("display_errors",1);
3.
4.  if($_SERVER['REQUEST_METHOD'] == 'POST') {
5.  $nc = mysqli_connect('localhost','root','1111');
6.  mysqli_select_db($nc,'injection');
7.
8.  $usuario = $_POST['usuario'];
9.  $senha = $_POST['senha'];
10. $sql = "SELECT * FROM login WHERE usuario = '$usuario' AND
    senha='$senha'";
11. $query = mysqli_query($nc, $sql);
12.
13. if(mysqli_num_rows($query)==1) {
14.     header("location:admin.php");
15. }
16.
17. else {
18.     echo "Usuário ou senha inválidos";
19. }
20. }
21. ?>
22.
23. <!DOCTYPE html>
24. <html>
25. <head>
26. <title>SQL Injection</title>
27. </head>
28. <body>
29. <form action="index.php" method="POST">
30. <h1>Login - Vulnerável</h1><br>
31. Usuário:<br>
32. <input type="text"
33. name="usuario"<br><br>
34. Senha:<br>
35. <input type="text"
36. name="senha"<br><br>
37. <input type="submit" value="Login">
38. </Form>
39. </body>
40. </html>
```

APÊNDICE B – Código fonte da aplicação web com de métodos de defesa contra o SQL *Injection.*

```
1.  <?php
2.  if($_SERVER['REQUEST_METHOD'] == 'POST') {
3.  $usuario = $_POST['usuario'];
4.  $senha = $_POST['senha'];
5.
6.  $root = 'root';
7.  $password= '1111';
8.  $conn = new PDO('mysql:host=localhost;dbname=fail', $root, $password);
9.
10. $sql = "SELECT * FROM login WHERE usuario=:usuario AND senha=:senha;";
11.       $stmt = $conn->prepare( $sql );
12.       $stmt-> bindParam( ':usuario', $usuario );
13.       $stmt-> bindParam( ':senha', $senha );
14.       $stmt->execute();
15.       $result = $stmt->fetchAll();
16.
17. if (count($result)) {
18.     header("location: admin2.php");
19. }
20. else {
21.     throw new Error($stmt->errorInfo());
22. }
23. }
24.
25. ?>
26.
27. <!DOCTYPE html>
28. <html>
29. <head>
30. <title>SQL Injection</title>
31. </head>
32. <body>
33. <form action="index2.php" method="POST">
34. <h1>Login - Seguro</h1><br>
35. Usuário:<br>
36. <input type="text"
37. name="usuario"<br><br>
38. Senha:<br>
39. <input type="text"
40. name="senha"<br><br>
41. <input type="submit" value="Login">
42. </Form>
43. </body>
44. </html>
```