FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

William Toshio Nakagawa

Segurança em aplicações web

Comparativo entre API MySQL e API PDO

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

William Toshio Nakagawa

Segurança em aplicações web

Comparativo entre API MySQL e API PDO

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Ms. Benedito Luciano Antunes de França.

Área de concentração: Tecnologia da informação.

Americana, SP

2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

N152s NAKAGAWA, William Toshio

Segurança em aplicações web:comparativo entre API MySQL e API PDO. / William Toshio Nakagawa. – Americana, 2017.

62f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Benedito Luciano Antunes de França

1. PHP – linguagem de programação I. FRANÇA, Benedito Luciano Antunes de II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.061

William Toshio Nakagawa

SEGURANÇA EM APLICAÇÕES WEB
Comparativo entre API MySQL e API PDO

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo Centro Paula Souza – FATEC Faculdade de Tecnologia de Americana.
Área de concentração: Tecnologia da informação

Americana, 12 de dezembro de 2017.

Banca Examinadora:

Benedito Luciano Antunes de França (Presidente)
Mestre
Fatec Americana

Clerivaldo Jose Roccia (Membro)
Mestre
Fatec Americana

Wladimir da Costa (Membro)
Mestre
Fatec Americana

AGRADECIMENTOS

Primeiramente a Deus, por iluminar sempre o meu caminho colocando nele pessoas de bem.

Aos meus pais e irmãs, pelo amor, incentivo e apoio incondicional.

A minha mulher Daniele, por me aturar durante todo esse tempo em que estive na caminhada acadêmica.

A todos os professores que acompanharam minha jornada enquanto universitário e foram essenciais à minha formação como profissional e, além disso, minha evolução como pessoa.

A todos os amigos da classe ADS 2012s2, companheiros de trabalho (CPD Fatec Americana) e outras pessoas que de alguma forma fizeram parte da minha formação e que vão continuar presentes em minha vida com certeza.

DEDICATÓRIA

Ao Professor Mestre Benedito Luciano Antunes de França, pela paciência, incentivo, apoio, compreensão, amizade e dedicação de suas horas de lazer e descanso para me auxiliar e orientar no desenvolvimento deste trabalho de conclusão de curso. Eu posso dizer que a minha formação, inclusive pessoal, não teria sido a mesma sem a sua pessoa.

RESUMO

Este trabalho pretende demonstrar o quão pode ser vulnerável uma aplicação web, que emprega uma API (Application Programming Interface) desatualizada e depreciada, perante a vários tipos de ataques de injeção SQL (Structured Query Language). Para amenizar estes ataques injetados via barra de endereço do navegador ou via formulários HTML foi implementado uma API mais atualizada e parametrizada, chamada PDO (PHP Data Objects). Discorremos tecnicamente sobre a funcionalidade e a arquitetura exigidas de uma aplicação web, assim como analisamos acerca da linguagem PHP (PHP: Hypertext Preprocessor), banco de dados MySQL, ataques de injeção SQL e Segurança da Informação. Por fim, este trabalho apresenta um estudo de caso fundamentado em um servidor web local, que hospedou duas aplicações web (API MySQL e PDO), por meio das quais foi automatizado um teste de penetração, fazendo o uso de uma ferramenta de escaneamento de vulnerabilidade web Programa ZAP (Zed Attack Proxy), no intuito de verificar se as aplicações sofreram ou não ataques de injeção SQL.

PALAVRAS-CHAVE: API MySQL; API PDO; injeção SQL; MySQL; PHP.

ABSTRACT

This work aims to demonstrate how a web application can be vulnerable, when it employs an outdated and depreciated API (Application Programming Interface) in cases of many types of SQL (Structured Query Language) injection attacks. In order to mitigate these attacks injected via the browser's address bar or via HTML forms, a more updated and parameterized API called PDO (PHP Data Objects) was implemented. We describe technically about the required functionality and architecture of a web application, as well as we analyze PHP (PHP: Hypertext Preprocessor) language, MySQL database, SQL injection attacks and Information Security. Finally, this work presents a case study based on a local web server, which hosted two web applications (API MySQL and PDO), by means of which a penetration test was automated, making use of a vulnerability scanning tool Zap (Zed Attack Proxy) Program, in order to verify if the applications suffered or not SQL injection attacks.

KEYWORDS: API MySQL; API PDO; SQL injection; MySQL; PHP.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS, SÍMBOLOS E SIGLAS

API	Application Programming Interface
CEP	Código de Endereçamento Postal
DB-Engines	Database Engines
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IIS	Internet Information Services
MS-SQL	Microsoft Structured Query Language
MYSQL	My Structured Query Language
MYSQLI	My Structured Query Language Improved
NASA	National Aeronautics and Space Administration
Nginx	Engine X
ORACLE	Oak Ridge Automated Computer And Logical Engine
OS	Operational System
OWASP	Open Web Application Security Project
PDO	PHP Data Objects
PERL	Practical Extraction and Report Language
PHP	PHP: Hypertext Preprocessor
PHP/FI	Personal Home Page/Forms Interpreter
RISC	Reduced Instruction Set Computer
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
VBScritp	Visual Basic Script
W3Techs	Web Technology Surveys
ZAP	Zed Attack Proxy

LISTA DE TABELAS

LISTA DE GRÁFICOS

Sumário

INTRODUÇÃO

A grande rede (Internet) se desenvolve vertiginosamente em vários aspectos, como, por exemplo, em número de usuários, em abrangência a localidades remotas, assim como em tecnologia de transmissões de dados e velocidade de conexão (banda larga). Esse rápido crescimento se dá principalmente em países desenvolvidos, mas também está atingindo outros países em desenvolvimento como o caso do Brasil.

Diante deste cenário de expansão digital, muitas empresas e usuários estão ampliando a gama de serviços prestados no ambiente *on-line*, utilizando sistemas web para gerenciar seus processos e armazenar informações importantes. Esses serviços possuem diversas finalidades, como entretenimento, compras, comunicação e interação em fóruns, redes sociais, blogs e consumo de informações em geral, necessitando, quase sempre, realizar um cadastramento dos dados de seus clientes e usuários.

Estes sistemas web são armazenados em servidores de aplicação ou sistema, e utilizam banco de dados para armazenar os dados produzidos nas páginas web.
Entretanto, muitos desses sistemas web apresentam vulnerabilidades, inconsistências e falhas, devido principalmente a falta de atualização de API (Aplication Programming Interface), de conhecimento e/ou aplicação de técnicas de desenvolvimento de software seguro. Isso pode ser devido ao mau planejamento no desenvolvimento desse sistema, aos vícios de programação, falta de experiência, descuidos, falta de tempo, falta de conhecimentos dos programadores, falta de manutenção e atualizações, além o desconhecimento dos ataques recorrentes em aplicações web, podendo comprometer a segurança dos dados disponibilizados em cadastros pelos usuários de aplicações web.

Deste modo, é imprescindível que os sistemas web tenham um alto nível de segurança, por meio do qual as informações dos seus usuários possam trafegar sem que sejam interceptadas ou violadas até o seu destino, bem como impedir o acesso por pessoas mal-intencionadas, com o intuito de destruir, vender ou modificar as informações disponibilizadas no meio eletrônico.

Um dos problemas recorrentes nas aplicações web codificadas em PHP (acrônimo recursivo para *PHP: Hypertext Preprocessor)*, é a ausência de atualização na API MySQL que já está depreciada, desde a versão do PHP 5.5 e já não se encontra

inserida na versão PHP 7.0, a qual foi lançada em 2015. A atualização da API MySQL, que faz a conexão entre a aplicação web e o banco de dados, se faz necessária devido à falta de segurança e a descontinuidade da API nas versões atuais do servidor PHP, segundo descrito no próprio *site* PHP (2001-2017).

Nesse sentido, o presente Trabalho de Graduação visa mitigar os ataques de injeção SQL nos bancos de dados operacionalizando a API PDO, pois, partimos do pressuposto técnico-funcional que, ao fazer uso de uma API de conexão de banco de dados atualizada, será possível garantir confiança e segurança ao produto disponibilizado para o usuário final. Dessa forma, empregaremos um programa de escaneamento de vulnerabilidades chamado ZAP (Zed Attack Proxy), disponibilizado pela organização sem fins lucrativos OWASP (Open Web Application Security Project), por meio do qual verificaremos se a aplicação web está suscetível às vulnerabilidades de ataques de injeção SQL.

O grau de importância deste estudo se justifica na medida em que busca apresentar uma forma mais segura no desenvolvimento de aplicações em PHP, na intenção de melhorar o desempenho e a qualidade dos sistemas desenvolvidos em PHP utilizando a API PDO. Justificamos a escolha da API PDO em função de suas características, entre elas, de ser uma ferramenta livre, a qual pode ser migrada para vários bancos de dados existentes no mercado.

Nessa direção, atualizaremos um sistema em PHP que está usando a API MySQL para API PDO, a fim de escanear as vulnerabilidades dos ataques de injeção SQL, operacionando a ferramenta OWASP ZAP, no intuito de comparar a quantidade de ataques entre a API MySQL e a API PDO, assim como descrever as vantagens e desvantagens das duas APIs. Para todos os fins, ilustramos que o sistema em PHP, que será objeto desta análise, foi desenvolvido pelo ex-acadêmico André Luís Pizza Basso, egresso do curso de Análise e Desenvolvimento de Sistemas da Fatec Americana, que o desenvolveu com o objetivo de inserir as vagas de empregos e de estágios pelas empresas, registros que poderão ser acessados pelos atuais estudantes desta Fatec dos cursos de Análise e Desenvolvimento de Sistemas, Gestão Empresarial, Jogos Digitais, Logística, Produção Têxtil, Moda e Têxtil e Segurança da Informação. Este sistema ainda está em fase de testes e, em breve, será disponibilizado no Portal da Fatec Americana (http://www.fatec.edu.br), vinculada ao Centro Estadual de Educação Tecnológica "Paula Souza", autarquia estadual

associada à Secretaria de Desenvolvimento Econômico, Ciência, Tecnologia e Inovação, do governo do Estado de São Paulo.

Para fins didáticos, exploraremos teórica e conceitualmente a Aplicação web, o banco de dados MySQL, a linguagem de programação PHP e as noções elementares de Segurança da informação. Ademais, apresentaremos os ataques de injeção SQL utilizados pela ferramenta de escaneamento de vulnerabilidade ZAP, as práticas utilizadas para implementar a API PDO na linguagem PHP e, por último, faremos uma análise e comparação de como as APIs MySQL e PDO se comportam aos ataques de injeção SQL, sob o objetivo de garantir a segurança dessas informações.

Em termos metodológicos, o presente estudo será desenvolvido com base em pesquisas bibliográficas, documentais, artigos eletrônicos, acesso a Portais e na análise de estudo de caso, tomando como parâmetro o sistema desenvolvido pelo ex-acadêmico André Luís Pizza Basso, em função da defesa de seu Trabalho de Graduação intitulado "Desenvolvimento do sistema de gerenciamento de vagas disponíveis no mercado de trabalho para alunos da Fatec Americana", apresentado em 2015, para o curso de Análise e Desenvolvimento de Sistemas, por meio do qual obteve o título de tecnólogo no curso superior especificado.

Na pesquisa bibliográfica e documental serão levantadas informações relativas à linguagem PHP, a fim de abordar os conceitos da linguagem, assim como analisar suas práticas, princípios, valores e suas formas de segurança no desenvolvimento de sistemas web.

Com isso, conforme proposto, criaremos um estudo experimental para demonstrar ao usuário e aos desenvolvedores de aplicações web em PHP, as eventuais incidências de ataques de injeção SQL em aplicações que utilizam a API MySQL e API PDO com conexão do banco de dados MySQL.

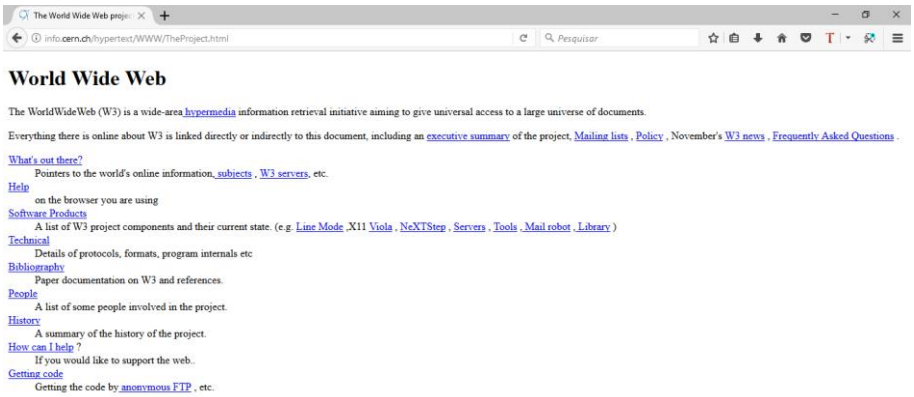
CAPÍTULO 1 – REFERENCIAL TEÓRICO-CONCEITUAL

Quando a Internet foi iniciada, era constituída apenas por páginas web estáticas, ou seja, os conteúdos dessas páginas eram basicamente constituídos por dados estagnados que não podiam ser alterados pelo visitante (STUTTARD; PINTO, 2011, p. 2).

Para que o usuário conseguisse ter acesso a uma página web, foi necessária a criação de uma aplicação que recuperasse e exibisse esses dados de uma forma mais interativa. Dessa maneira, foi dado a essa aplicação o nome "Navegador" ou "Browser".

Abaixo são apresentadas dois tipos de páginas web, sendo que a imagem 1 é de uma página web estática obtida da CERN (Conseil Européen pour la Recherche Nucléaire), enquanto que a imagem 2 é de uma página web dinâmica:

Imagem 1 – Página web estática



Fonte: CERN, 2017

Especificamente, conforme Stuttard e Pinto (2011), todos os usuários eram iguais perante ao acesso a página estática, pois não era possível fazer um *login* utilizando um usuário e senha, visto que todas as requisições traziam a mesma informação ao acessar este tipo de página. Consequentemente, as vulnerabilidades de segurança estavam relacionadas apenas ao servidor web. Se um intruso comprometesse o servidor que hospedava o sítio eletrônico, na maioria das vezes, não era comprometido nenhum dado importante, pois toda a informação do sítio já estava disponível para o público. O máximo que o invasor poderia fazer, era modificar ou apagar os arquivos que continham as páginas estáticas ou utilizar a rede do servidor web para distribuir programas piratas (STUTTARD; PINTO, 2011, p. 2).

Imagem 2 – Página web dinâmica

Fonte: Wikipédia, 2017

Atualmente a Internet está muito diferente de quando foi lançada. Uma grande parte dos *sites* existentes na web são páginas dinâmicas ou aplicações web, as quais geram um fluxo de dados entre o cliente (navegador) e o servidor. Essas aplicações permitem a realização de cadastros e acessos com usuário e senha, pesquisas, transações financeiras, etc. Com efeito, diferente do acesso à página estática, o usuário passou a receber, neste novo molde de página eletrônica, uma capacidade de modificar, inserir, editar, e, às vezes, até excluir o conteúdo da página, muitas vezes, com informações privadas e altamente sigilosas, o que gera uma grande preocupação com a segurança nas aplicações web (STUTTARD; PINTO, 2011, p. 3).

1.1. Arquitetura web

Conforme Stuttard e Pinto (2011, p. 5-6; 39-58), para que as funcionalidades de uma aplicação web sejam implementadas e disponibilizadas para a Internet são necessárias algumas tecnologias, tais como:

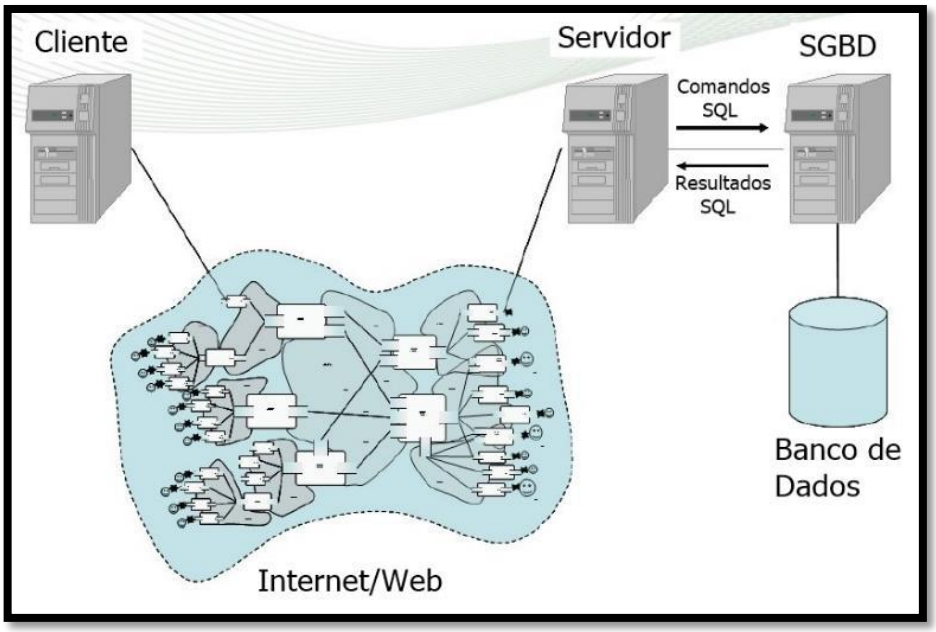
- Protocolo HTTP (*Hypertext transfer protocol*, que traduzido significa “Protocolo de transferência de hipertexto): é o principal protocolo de comunicação manuseado entre o navegador e o servidor web. Comumente este tipo de Protocolo é executado na porta 80 e envia mensagens em modo texto;
- Navegador: é uma aplicação do lado do cliente, que traduz a codificação da aplicação web para uma interface de fácil visualização e interação do usuário.

Podemos citar alguns navegadores existentes, como Firefox, Chrome, Safari, etc.;

- Servidor web: é um programa responsável pela hospedagem e execução dos arquivos na aplicação do cliente, retornando as requisições dos navegadores em páginas da web. Esses servidores web utilizam o Protocolo de comunicação HTTP e são um dos principais elementos responsáveis pela distribuição de sítios eletrônicos e vários conteúdos na web. Há vários tipos de servidores web, como o Apache, que é um programa gratuito, o IIS da Microsoft, entre outros;
- Interpretadores de linguagens de *script:* é um serviço implementado no lado do servidor web para que os arquivos escritos em linguagens tais como, PHP, VBScript e Perl, possam ser executados para que gerem as respostas requisitadas pelos usuários, a partir da aplicação web;
- Aplicação Web: é composta por vários arquivos codificados que serão interpretados e executados pelo servidor web, no qual enviará como saída um código HTML e que será interpretado pelo navegador, gerando uma interface manuseável pelo usuário (STUTTARD; PINTO, 2011, p. 5-6; 39-58);
- O SGDB, ou seja, sistema gerenciador de banco de dados, é um servidor que gerencia banco de dados relacional, por meio do qual podemos criar vários bancos de dados para inumeráveis aplicações. O termo relacional indica que no banco de dados as tabelas e entidades possuem relacionamentos entre as informações, gerando um recurso de controle na consistência dos dados, o que impede cadastros não coerentes na aplicação. Outro benefício de um banco de dados relacional é a manipulação das *querys* (consultas), pois, como existem relacionamentos entre as diversas tabelas, é possível consultar essas informações (*Cf.* MILANI, 2006, p. 313). MySQL, Oracle, MS-SQL, por exemplo, utilizados por pequenas, médias e grandes empresas, são alguns tipos de sistemas gerenciadores de bancos de dados.
- Banco de Dados: é uma aplicação que armazena as informações enviadas pelos usuários, através de formulários contidos nas páginas, como cadastros, *logins*, etc (STUTTARD; PINTO, 2011, p. 5-6; 39-58).

Na imagem 3 são esboçados os elementos computacionais que estão associados a uma arquitetura de uma aplicação web, segundo Holanda e Fernandes (2011, p. 12).

Imagem 3 – Esboço de uma arquitetura web



Fonte: Holanda; Fernandes, 2011, p. 12

1.2. Linguagem de Programação PHP e sua história

Em 1994, Rasmus Lerdorf criou alguns códigos *scripts* em Perl (Practical Extraction and Report Language) com o intuito de capturar dados estatísticos sobre os acessos em seu currículo profissional, que estava acessível na Internet. Com o passar do tempo, Ledorf aprimorou os *scripts* inserindo novas funcionalidades. Então, em um certo momento, ele reescreveu esses *scripts* em outra linguagem, que seria a linguagem C, para que pudesse ser instrumentalizada em construções de novas aplicações web; o nome dado a esse projeto de novas funcionalidades foi PHP/FI (Personal Home Page/Forms Interpreter) (MILANI, 2016, p. 21).

O projeto de Lerdorf ganhou popularidade na Internet por apresentar facilidades na implementação e por possuir várias funcionalidades dinâmicas, que estavam sendo manipuladas por vários usuários em suas páginas web. Em 1997, a versão 2.0 do PHP/FI foi lançada, e foi um sucesso, tanto que "se estima que 1% do total dos domínios existentes na internet na época usavam esse projeto" (MILANI, 2016, p. 21).

No ano de 1998, segundo Milani (2016), foi lançada a versão 3.0 criada por Andi Gutmans e Zeev Suraski, com apoio de Rasmus Ledorf, com a finalidade de resolver demandas existentes de alguns projetos desses novos colaboradores. Então,

reduziram o nome para PHP e reescreveram todos os códigos-fontes da tecnologia (MILANI, 2016, p. 21-22).

Visando atender projetos de grande complexidade, o que tornaria o PHP mais poderoso, foi desenvolvido em 1999 o PHP 4.0. E devido à estabilidade dessa tecnologia, "somente em 2004 foi lançada a nova versão do PHP (5.0), baseada em correções de bugs encontradas nesse período e com várias outras características" (MILANI, 2016, p. 22).

Devido a alguns problemas ocorridos no desenvolvimento do PHP 6.0, essa versão não foi oficialmente distribuída, e por isso foi mantido o PHP 5.x até o final do ano de 2015, quando foi lançado o PHP 7.0. Até o presente momento em que este trabalho está sendo escrito, a versão atual do PHP é 7.1.11, conforme informação colhida em 30 de outubro de 2017 (THE PHP GROUP, 2001-2017b).

Em linhas gerais, para Converse e Park, acerca do PHP afirmam que:

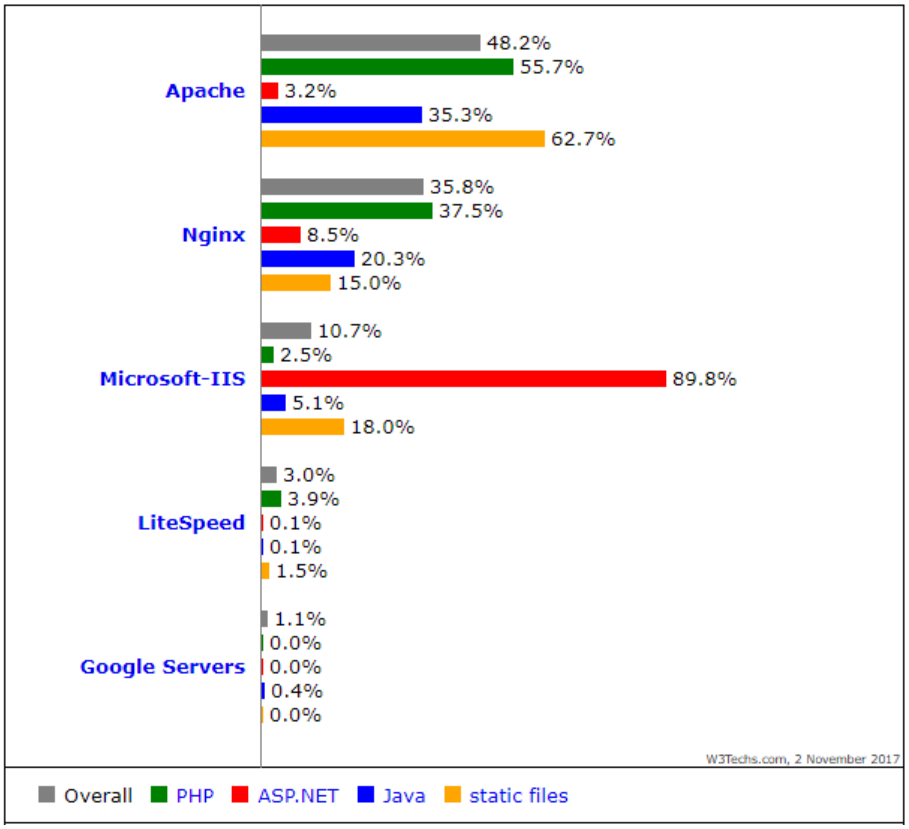
> É uma linguagem para a criação de scripts para a Web do lado servidor embutidos em HTML, cujo código-fonte é aberto, e que é compatível com os mais importantes servidores Web (especialmente o Apache). O PHP permite incorporar fragmentos de código em páginas de HTML normais — código esse que é interpretado à medida que suas páginas são oferecidas aos usuários. O PHP também serve como uma linguagem de "cola", facilitando a conexão de suas páginas Web com o banco de dados do lado servidor (CONVERSE; PARK, 2002, p. XXVII).

De acordo com o sítio eletrônico da PHP, tradicionalmente, esta linguagem de programação é muito mais instrumentalizada em aplicações web com *scripts* do lado do servidor (*server-side*), que precisam da estrutura navegador, interpretador PHP e servidor web, para que funcione. Mas, por outro lado, também pode ser empregado em códigos de *scripts* de linha de comando, que é utilizado para automatizar um processo dentro de um sistema operacional utilizando o agendamento de tarefas, necessitando apenas do serviço de interpretação do PHP. Além disso, pode ser utilizada em aplicações desktop (THE PHP GROUP, 2001-2017f).

A linguagem PHP é suportada por vários tipos de sistemas operacionais, tais como Linux, Microsoft Windows, Mac OS X, RISC OS, entre outros. E também vários servidores web, tais como Apache, Nginx, Microsoft-IIS, LiteSpeed, Google Servers, etc.

Segundo o Gráfico 1 obtido do *site* W3techs, 55,7% das aplicações PHP estão hospedadas em servidores Apache:

Gráfico 1 – Estatísticas das aplicações web hospedadas em servidores web



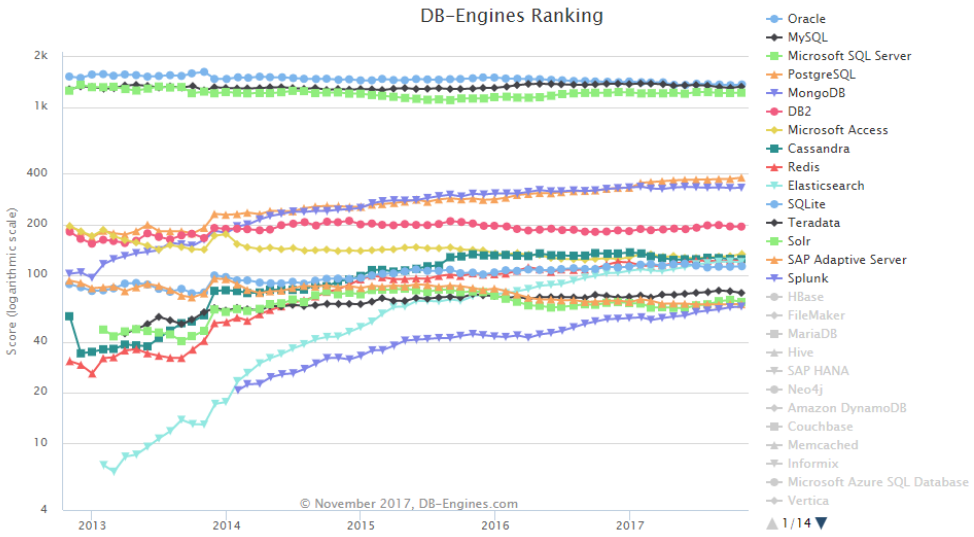
Fonte: W3techs, 2009-2017

Portanto, ao fazer uso da linguagem PHP, nós temos uma maior flexibilidade na escolha tanto do sistema operacional quanto do servidor web. No tocante ao estilo de programação, esta pode ser do tipo estruturada ou orientada a objeto, ou a junção das duas. Outro aliado do PHP é o suporte a amplas opções de banco de dados, haja vista que as extensões específicas auxiliam na conexão entre a aplicação e o banco de dados, conforme assegura o Manual PHP (THE PHP GROUP, 2001-2017f).

1.3. Banco de Dados MySQL

Graças ao acesso via Internet houve um encurtamento na relação entre as pessoas do mundo. A maioria dos negócios foi sendo migrado para a web, gerando uma grande quantidade de informações e de armazenamento de dados. Visando esse mercado crescente e altamente potencial, nasce o MySQL, que é atualmente o banco de dados gratuito mais utilizado no mundo, segundo o ranking do *site* DB-Engines (Gráfico 2), entre o período de 2013 e 2017, que classifica os sistemas de gerenciamento de banco de dados, conforme a sua popularidade.

Gráfico 2 – Estatística da popularidade de uso do Banco de Dados (2013-2017)



Fonte: DB-Engines, 2017

Conforme Milani, o Mysql:

> (...) é um banco de dados completo, robusto e extremamente rápido, com todas as características existentes nos principais bancos de dados pagos existentes no mercado. Uma de suas peculiaridades são suas licenças para uso gratuito, tanto para fins acadêmicos como para realização de negócios, possibilitando que na maioria dos casos as empresas o utilizem livremente (MILANI, 2006, p. 21).

Muitas empresas e organizações conhecidas fazem uso deste sistema gerenciador de banco de dados MySQL, segundo consta no *site* MySQL, na seção denominada "clientes" (customers); entre estes estão a NASA (National Aeronautics and Space Administration ou Administração Nacional da Aeronáutica e Espaço), Facebook (Rede social), Banco da Finlândia (Instituição financeira), Youtube (Rede social de vídeos), Netflix (Streaming de filmes), Spotify (Streaming de músicas), entre outros (MYSQL, 2017).

O SGBD MySQL é um sistema de gerenciamento de banco de dados que foi disponibilizado para o público em 1996 por funcionários da TcX DataKonsult AB, instituída na Suécia. O programa não se mostrou popular no começo, mas em 2001 foi fundada a empresa MySQL AB, que era totalmente focada no projeto do MySQL (GILMORE, 2010, p. 477).

A MySQL AB sempre foi lucrativa desde o seu início, o que motivou a empresa criar escritórios em vários países e, assim, angariar substanciais recursos financeiros para o capital de risco e atrair parcerias com várias grandes empresas da época, como Red Hat, Veritas, Novell e Rackspace (GILMORE, 2010, p. 477).

Em 2008, a empresa Sun Microsystems comprou a MySQL AB, e, em 2009, a Oracle Corporation adquiriu os direitos da empresa Sun Microsystems (GILMORE, 2010, p. 477).

1.4. API de conexão entre PHP e o banco de dados MySQL

1.4.1. O que é API?

API (Aplication Programming Interface, cuja tradução para o português significa “Interface de Programação de Aplicativo”) é um conjunto de padrões e rotinas de programação para o acesso a uma plataforma web ou aplicativo de software. Geralmente criada por empresas de software que têm como objetivo que outros programadores de software desenvolvam aplicativos associados ao seu serviço. Existem várias empresas que disponibilizam as próprias instruções e os códigos para serem utilizados em outros *sites* de uma maneira adequada para seus usuários (CANALTECH, 2017).

A fim de exemplificação de uso do recurso API, recorremos ao portal brasileiro da empresa de telégrafos e correios, visto que para calcular os prazos e preços de fretes, muitas lojas virtuais fazem uso corrente dessa ferramenta. Essa API é um serviço disponibilizado pelos correios, o qual o proprietário da loja virtual poderá agregá-lo ao código do seu site, para que o cliente possa inserir o CEP (Código de Endereçamento Postal) de sua casa e verificar o preço e o prazo de entrega da encomenda:

> O webservice de cálculo remoto de preços e prazos de encomendas dos Correios é destinado aos clientes SEDEX e PAC que necessitam calcular o preço e o prazo de entrega de uma encomenda em seus websites, de forma personalizada. Os clientes que não possuem contrato de encomenda com os Correios podem usar esta ferramenta, porém os preços apresentados serão os praticados no balcão da agência (CORREIOS, 2017).

O PHP disponibiliza três diferentes tipos de API para a conexão com o banco de dados MySQL. As extensões existentes são: 1) mysqli; 2) PDO; e 3) mysql.

Na imagem 4 pode ser visto os três tipos de APIs para o banco de dados MySQL, codificadas em PHP, conforme retirado do *site* Manual do PHP:

Imagem 4 – Três tipos de conexão API do banco de dados MySQL:

```
<?php
    // mysqli
    $mysqli = new mysqli("example.com", "user", "password",
    "database");
    $result = $mysqli->query("SELECT 'Hello, dear MySQL user!' AS
    _message FROM DUAL");
    $row = $result->fetch_assoc();
    echo htmlentities($row['_message']);

    // PDO
    $pdo = new PDO('mysql:host=example.com;dbname=database',
    'user', 'password');
    $statement = $pdo->query("SELECT 'Hello, dear MySQL user!'
    AS _message FROM DUAL");
    $row = $statement->fetch(PDO::FETCH_ASSOC);
    echo htmlentities($row['_message']);

    // mysql
    $c = mysql_connect("example.com", "user", "password");
    mysql_select_db("database");
    $result = mysql_query("SELECT 'Hello, dear MySQL user!' AS
    _message FROM DUAL");
    $row = mysql_fetch_assoc($result);
    echo htmlentities($row['_message']);
?>
```

Fonte: The PHP Group, 2001-2017a

1.4.2. API MySQL

A “extensão mysql” foi originalmente projetada para auxiliar a conexão de aplicações PHP com o banco de dados MySQL, essa extensão provê uma interface procedural, e pode ser utilizada nas versões 4.1.3 ou anteriores do MySQL, pois nas versões posteriores alguns dos recursos antigos do servidor MySQL não estarão mais disponíveis.

Segundo o *site* Manual do PHP, não é aconselhável utilizar a “extensão mysql” para novos projetos de sistemas, devido à falta de manutenção nas versões do PHP 5.x e sua remoção a partir da versão PHP 7, por isso para novos desenvolvimentos de aplicações, o portal recomenda o uso das API MySQLi ou PDO (THE PHP GROUP, 2001-2017a).

O fim da manutenção nas funções da API MySQL, a partir da versão PHP 5.x, provoca um risco enorme nas aplicações web, haja vista que as possíveis falhas de segurança descobertas não mais receberão as devidas correções, tornando-as vulneráveis, podendo assim abrir brechas de segurança na aplicação, como a injeção SQL.

Outra grave consequência gerada pela remoção das funções da API MySQL na versão PHP 7 é que, caso o servidor web realize uma atualização da versão PHP 5.x para a versão PHP 7, isso acarretará no não funcionamento de todas as aplicações que contenham essa API, por conseguinte, todas as conexões com o banco de dados serão perdidas, além disto, nenhuma autorização de alteração, correção, atualização, exclusão, inserção, edição, quando requisitadas ao banco de dados, não serão atendidas.

Em seguida, na Tabela 1 são fornecidas informações sobre os recursos de cada API:

Tabela 1 – Comparação de recursos entre as APIs de conexão PHP

	ext / mysqli	PDO_MySQL	ext / mysql
Introdução de versão PHP	5.0	5.1	2.0
Incluído com PHP 5.x	sim	sim	sim
Incluído com PHP 7.x	sim	sim	Não
Estado de Desenvolvimento	Ativo	Ativo	Manutenção apenas em 5.x; removido em 7.x
Ciclo da vida	Ativo	Ativo	Desempenhado em 5.x; removido em 7.x
Recomendado para novos projetos	sim	sim	Não
Interface OOP	sim	sim	Não
Interface processual	sim	Não	sim
API suporta consultas não-bloqueadoras e assíncronas com mysqlnd	sim	Não	Não
Conexões persistentes	sim	sim	sim
API suporta Charsets	sim	sim	sim
API suporta instruções preparadas para o lado do servidor	sim	sim	Não
API suporta instruções preparadas no lado do cliente	Não	sim	Não
API suporta procedimentos armazenados	sim	sim	Não
API suporta várias declarações	sim	A maioria	Não
API suporta Transações	sim	sim	Não
Transações podem ser controladas com SQL	sim	sim	sim
Suporta todas as funcionalidades do MySQL 5.1+	sim	A maioria	Não

Fonte: The PHP Group, 2001-2017a

A seguir descreveremos as funcionalidades de cada função da API MySQL utilizadas no Sistema de empregos criado pelo estudante André Luís Pizza Basso, o qual é objeto desta análise:

1.4.2.1. Função mysql_connect()

A função *mysql_connect()* serve para abrir uma conexão com um servidor MySQL.

Na imagem 5 e demonstrado a sintaxe da função *mysql_connect()* conforme Manual do PHP:

Imagem 5 – Sintaxe da função *mysql_connect()*

```
resource mysql_connect ([ string $server [, string $username [, string $password [, bool $new_link [, int $client_flags ]]]]] )
```

Fonte: The PHP Group, 2001-2017d

Com a alteração da sintaxe da função efetuada pelo pesquisador, do comando originário do Manual PHP usa-se apenas a expressão *mysql_connect(string $server, string $username, string $password)*, conforme será detalhado na imagem 6, com a explicitação do seguinte comando:

Imagem 6 – Sintaxe da função *mysql_connect()* utilizada no Sistema de Empregos

```
mysql_connect ('$endereço_do_servidorMySQL', '$usuarioBD', '$senhaBD')
```

Fonte: Autoria própria, 2017

Os parâmetros utilizados na função *mysql_connect()* são definidos:

- *$endereço_do_servidorMySQL:* é o endereço do servidor MySQL que pode ser um IP ou o nome do servidor e que também pode conter um número de porta, como por exemplo, “127.0.0.1:3306“, ou “localhost:3306”.
- *$usuarioBD*: é o nome de usuário do banco de dados. Geralmente o usuário padrão é o nome do usuário do proprietário do processo do servidor MySQL, o “root”.
- *$senhaBD*: é a senha do MySQL. Geralmente a senha padrão é uma senha vazia.

Com esta função podemos estabelecer uma conexão no servidor MySQL que, ao se conectar, retorna um identificador da conexão MySQL, (*TRUE*) em caso de sucesso, ou, contrariamente, *FALSE*, em caso de falha.

Segue-se na imagem 7 um exemplo da função *mysql_connect(),* extraído do Manual PHP, com adaptação para a nossa língua vernácula, com nome do servidor e a porta:

Imagem 7 – Codificação da função *mysql_connect()* com servidor e porta

```
<?php
   $link = mysql_connect("localhost:3306", "usuario", "senha")
      or die("Não consegui conectar: " . mysql_error());
   print("Conectado");
?>
```

Fonte: The PHP Group, 2001-2017d

Na imagem 8 e demonstrado outro exemplo da função *mysql_connect(),* extraído do Manual PHP, com IP do servidor e a porta:

Imagem 8 – Codificação da função *mysql_connect()* com IP e porta

```
<?php
   $link = mysql_connect("127.0.0.1:3306", "usuario", "senha")
      or die("Não consegui conectar: " . mysql_error());
   print ("Conectado");
?>
```

Fonte: The PHP Group, 2001-2017d

1.4.2.2. Função *mysql_select_db()*

A função *mysql_select_db()* é utilizada para definir a seleção de um banco de dados MySQL.

Na imagem 9 e demonstrado a sintaxe da função *mysql_select_db()* conforme Manual do PHP:

Imagem 9 – Sintaxe da função *mysql_select_db()*

```
bool mysql_select_db ( string $database_name [, resource $ link_identifier ] )
```

Fonte: The PHP Group, 2001-2017d

Com a alteração da sintaxe do método efetuada pelo pesquisador, do comando originário do Manual PHP usa-se apenas a expressão *mysql_select_db(string $database_name)*, conforme será detalhado na imagem 10, com a explicitação do seguinte comando:

Imagem 10 – Sintaxe da função *mysql_select_db()* do Sistema de empregos

```
mysql_select_db (nome_do_banco_de_dados)
```

Fonte: Autoria própria, 2017

Esta função indica o nome do banco de dados ativo no servidor que será associado ao identificador de conexão (*link_identifier*) especificado. Caso nenhum identificador de conexão seja especificado, assumi-se a última conexão aberta. Se não houver nenhuma conexão aberta a função *mysql_connect()* tentará estabelecer uma conexão sem nenhum argumento. Caso nenhuma conexão for estabelecida ou encontrada, um erro do nível *E_WARNING* é gerado.

Esta função retorna *TRUE* (verdadeiro) em caso de sucesso ou *FALSE* (falso) em caso de falhas.

Na imagem 11 e demonstrado um exemplo da função *mysql_select_db()*:

Imagem 11 – Codificação da função *mysql_select_db()*

```
<?php
   mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
?>
```

Fonte: The PHP Group, 2001-2017d

Segue-se a imagem 12 com um exemplo da função *mysql_select_db()* com *link_identifier*:

Imagem 12 – Codificação da função *mysql_select_db()* com *link_identifier*

```
<?php
   $link = mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
$banco_dados = mysql_select_db('banco', $link);
if (!$banco_dados) {
   die ('Não consegui usar o banco de dados: ' . mysql_error());}
?>
```

Fonte: The PHP Group, 2001-2017d

1.4.2.3. Função *mysql_query()*

A função *mysql_query()* é utilizada para realizar uma consulta SQL no banco de dados MySQL.

Na imagem 13 pode ser visto a sintaxe da função *mysql_query()*:

Imagem 13 – Sintaxe da função *mysql_query()*

```
mysql_query (“comando SQL”, link_identifier )  ou
mysql_query (“comando SQL”)
```

Fonte: Autoria própria, 2017

Com esta função enviamos uma consulta para o banco de dados no servidor em que informamos no *link_identifier* (variável de conexão da função *mysql_connect()*). Se o parâmetro da variável de conexão não estiver especificado, será usada a última conexão aberta. Se não houver nenhuma conexão aberta a função *mysql_connect()* tentará estabelecer uma conexão sem nenhum argumento. Caso nenhuma conexão for estabelecida ou encontrada, um erro do nível *E_WARNING* é gerado.

Para os comandos SQL *SELECT*, *SHOW*, *EXPLAIN* ou *DESCRIBE* a função *mysql_query()* retorna o identificador de recurso ou *FALSE* se a consulta não foi executada corretamente. Em outros tipos de comandos SQL, como, *UPDATE*, *DELETE*, *DROP*, etc., a função *mysql_query()* retorna *TRUE* em caso de sucesso e *FALSE* no caso de erro. O valor não *FALSE* mostra que uma consulta foi legal e poderá ser executada pelo servidor.

Na imagem 14 e demonstrado um exemplo da função *mysql_query():*

Imagem 14 – Codificação da função mysql_query

```
<?php
$resultado = mysql_query("SELECT nome FROM produtos")
   or die ("Query invalida: " . mysql_error());
?>
```

Fonte: The PHP Group, 2001-2017d

Segue-se na imagem 15 outro exemplo *mysql_query()* com *link_identifier*:

Imagem 15 – Codificação da função *mysql_query()* com *link_identifier*

```
<php
$resultado = mysql_query("SELECT * FROM produtos", $conexao)
   or die("Query invalida: " . mysql_error());
?>
```

Fonte: The PHP Group, 2001-2017d

1.4.2.4. Função *mysql_fetch_array()*

A função *mysql_fetch_array()* procura o resultado da linha e coloca-o como uma matriz numérica, associativa ou ambas.

Na imagem 16 e demonstrado a sintaxe da função *mysql_fetch_array()*:

Imagem 16 – Sintaxe da função *mysql_fetch_array()*

```
mysql_fetch_array (resultado da consulta, tipo da matriz)
```

Fonte: Autoria própria, 2017

Esta função retorna uma matriz que corresponde a linha adquirida, ou *FALSE* se não existirem mais linhas. O parâmetro “resultado da consulta” é o resultado da chamada da função *mysql_query()*. No parâmetro “tipo da matriz” é definido qual o tipo de índice que terá na matriz; ela pode ser do tipo *MYSQL_ASSOC*, na qual os índices serão do tipo associativo, que serão usados como chave os nomes de campos da tabela do banco de dados, ou do tipo *MYSQL_NUM*, na qual os índices serão do tipo numérico, que é usado como chave a numeração da matriz, ou pode ser do tipo *MYSQL_BOTH,* que corresponde a uma matriz com ambos os índices, ou seja, o numérico e o associativo.

A seguir na imagem 17 pode ser visto um exemplo da função *mysql_fetch_array()* com *MYSQL_ASSOC*:

Imagem 17 – Codificação da função *mysql_fetch_array()* com *MYSQL_ASSOC*

```
<?php
   mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
   $resultado = mysql_query("SELECT codigo, nome FROM
nome_tabela");
   while ($linha = mysql_fetch_array($resultado, MYSQL_ASSOC)) {
      printf ("Código: %s  Nome: %s", $linha["codigo"], $linha["nome"]);
   }
?>
```

Fonte: The PHP Group, 2001-2017d

A seguir na imagem 18 e demonstrado outro exemplo da função *mysql_fetch_array()* com *MYSQL_NUM*:

Imagem 18 – Codificação da função *mysql_fetch_array()* com *MYSQL_NUM*

```
<?php
   mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
   $resultado = mysql_query("SELECT codigo, nome FROM
nome_tabela");
   while ($linha = mysql_fetch_array($resultado, MYSQL_NUM)) {
      printf ("Código: %s  Nome: %s", $linha[0], $linha[1]);
   }
?>
```

Fonte: The PHP Group, 2001-2017d

Segue-se na imagem 19 outro exemplo da função *mysql_fetch_array()* com *MYSQL_BOTH*:

Imagem 19 – Codificação da função *mysql_fetch_array()* com *MYSQL_BOTH*

```
<?php
   mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
   $resultado = mysql_query("SELECT codigo, nome FROM
nome_tabela");
   while ($linha = mysql_fetch_array($resultado, MYSQL_BOTH)) {
      printf ("Código: %s  Nome: %s", $linha[0], $linha["nome"]);
   }
?>
```

Fonte: The PHP Group, 2001-2017d

1.4.2.5. Função *mysql_insert_id()*

A função *mysql_insert_id()* retorna o ID gerado pela operação do comando SQL *INSERT* anterior.

Na imagem 20 segue-se a sintaxe da função *mysql_insert_id()*:

Imagem 20 – Sintaxe da função *mysql_insert_id()*

```
mysql_insert_id (link_identifier)
```

Fonte: Autoria própria, 2017

Com a função *mysql_insert_id()* o ID gerado pela última consulta SQL *INSERT* usando o *link_identifier* do dado. Se não for especificado o *link_identifier*, a última conexão aberta será utilizada.

Na imagem 21 e demonstrado um exemplo da função *mysql_insert_id()*:

Imagem 21 – Codificação da função *mysql_insert_id()*

```
<?php
   mysql_connect("localhost", "usuario", "senha") or
      die("Não pude conectar: " . mysql_error());
   mysql_select_db("bancoDeDados");
   mysql_query("INSERT INTO tabela (produto) values ('arroz')");
   printf ("O ultimo registro incluído tem id %d\n", mysql_insert_id());
?>
```

Fonte: The PHP Group, 2001-2017d

1.4.3. API PDO

Segundo Lockhart (2015, p. 128) a extensão PDO (PHP Data Objects) é uma coleção de classes PHP que se conectam com vários tipos de banco de dados SQL por meio de uma única interface de usuário, na qual a finalidade é abstrair as interações e a conexão com o banco de dados MySQL, isso significa que independente do banco que estivermos usando, os métodos implementados serão sempre os mesmos.

Além do MySQL, a API PDO suporta os seguintes bancos de dados relacionais, conforme pode ser visto na Tabela 2:

Tabela 2 – Lista dos drivers exemplificados pelo *site* PHP

DRIVER	BANCO DE DADOS SUPORTADOS
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x / 4.x / 5.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC e win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 e SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure
PDO_4D	4D

Fonte: The PHP Group, 2001-2017c

Para habilitar a extensão PDO no PHP é necessária a edição do arquivo de configuração *php.ini*, localizando as linhas *extension=php_pdo.dll* e *extension=php_pdo_mysql.dll* do documento, e, para descomentar, é preciso retirar a pontuação ";" do início dessas linhas. Caso queira utilizar outro tipo de SGBD, basta descomentar a linha inerente ao SGBD. É importante salientar que o exemplo citado é de um sistema operacional Windows; caso o sistema operacional seja Linux, será necessário procurar as linhas *extension=pdo.so* e a *extension=pdo_mysql.so*.

> O PDO representa uma camada de abstração de acesso aos dados, as mesmas funções utilizadas para manipular dados ou recuperar informações do banco serão as mesmas, independentemente do banco de dados que esteja sendo usado (MISCELÂNIA DO CONHECIMENTO, 2017).

A construção das aplicações web torna-se mais segura, em termos de banco de dados, utilizando a API PDO, visto que uma das vantagens é o uso de *prepared statement* (instruções preparadas), que auxilia no combate de injeções indesejadas nas instruções SQL.

> *Prepared Statemens* são consultas pré-preparadas onde dividimos em partes a inserção do código SQL a ser executado e os valores a serem utilizados (*bind parans* ou *bind values).* Com os *prepareds statements* utilizamos na *query* os marcadores de lugar e depois informamos quais serão os valores que deverão ser utilizados em cada um dos lugares. Isso fornece mais segurança ao nosso banco de dados e nos previne de ações maliciosas como o SQL Injection (MISCELÂNIA DO CONHECIMENTO, 2017).

A seguir apresentaremos as aplicabilidades dos métodos da classe da API PDO que foram utilizadas para realizar a migração da API MySQL, garantindo-lhe mais segurança, portabilidade e performance ao sistema Empregos.

1.4.3.1. Conexão com o Banco de dados utilizando PDO

Para se criar uma conexão do PHP com o banco de dados utilizando o PDO, é necessário criar uma instância da classe PDO, inserindo os parâmetros do tipo string $dsn, $user, $pass no método construtor da classe, conforme pode ser visto na imagem 22 um exemplo extraído do Manual PHP:

Imagem 22 – Codificação da classe PDO para criar conexão entre o PHP e o MySQL

```
<?php
        $dbh = new PDO('mysql:host=localhost;dbname=test', $user,
$pass);
?>
```

Fonte: The PHP Group, 2001-2017g

Os parâmetros utilizados na classe PDO são definidos, de acordo com o Portal da PHP (2001-2017):

- *$dsn:* o argumento DSN (Nome da fonte de dados ou *Data Source Name*), e o que fornece detalhes para a conexão com o banco de dados. O DSN começa com o nome do driver do banco de dados utilizado, por exemplo, mysql ou sqlite, após inserir o nome do driver, colocar ":" e o restante da *string* de conexão, que geralmente é diferente para cada tipo de banco de dados, mas normalmente, é constituída por:
    - Endereço IP ou nome do servidor (host);
    - Número da porta;
    - Nome do banco de dados;
    - Conjunto de caracteres.
- *$user:* é inserido o nome de usuário do banco de dados. Que geralmente o usuário padrão é o "root" que é o proprietário do processo do servidor MySQL, mas esse usuário deve ser alterado por motivos de segurança.
- *$pass:* esse parâmetro recebe a senha do servidor MySQL, que geralmente a senha padrão é uma senha vazia (THE PHP GROUP, 2001-2017g).

Caso haja algum erro de conexão, será lançada um objeto *PDOException*. Essa exceção pode ser capturada para tratar melhor a condição do erro.

1.4.3.2. Método PDO::prepare

O método *PDO::prepare* apronta um comando para execução e retorna um objeto de declaração.

Na imagem 23 e demonstrado a sintaxe do método PDO::prepare conforme Manual de PHP:

Imagem 23 – Sintaxe do método PDO::prepare

```
public PDOStatement PDO:: prepare (string $Statement)
```

Fonte: The PHP Group, 2001-2017g

Com a alteração da Sintaxe do método efetuada pelo pesquisador, do comando originário do Manual PHP usa-se apenas a expressão *prepare (string $Statement*), conforme será detalhado na imagem 24, com a explicitação do seguinte comando:

Imagem 24 – Sintaxe do método PDO::prepare utilizado no sistema de empregos

```
$declaracao = $conexao->prepare($comandoSQL);
```

Fonte: Autoria própria, 2017

Este método prepara um comando SQL para ser executado pelo método *PDOStatement::execute*, que será citado no decorrer deste Trabalho.

No comando SQL que é inserida no parâmetro *$comandoSQL* pode conter marcadores de parâmetro dos valores reais, que podem ser de três tipos: 1) nomeado (:nome), 2) ponto de interrogação (?) ou 3) não conter nenhum marcador.

Não é possível utilizar dentro do mesmo comando SQL os dois marcadores de parâmetros, o que implica que será necessário escolher um tipo de marcador de parâmetro apenas, isto é, combinações 1) ou 3) ou 2) ou 3). Esses parâmetros são utilizados para vincular a entrada de dados de qualquer informação que será inserida na instrução SQL.

Ilustraremos a seguir os métodos 1) *PDO::prepare*, 2) *PDOStatement::BindValue* e 3) *PDOStatement::execute*, que são utilizados para definir uma instrução preparada (prepared statement), otimizando o desempenho para instruções que serão emitidas várias vezes com diferentes valores de parâmetros, eliminando a necessidade de faze-los manualmente, permitindo ao driver *PDO_MySQL* a negociação do cliente

com o servidor sobre a consulta realizada, e assim ajudando a evitar ataques de injeção SQL.

O parâmetro utilizado no método *PDO::Prepare* da classe *PDO* é definido:

- *$ comandoSQL*: é o comando SQL adequado para o servidor de banco de dados alvo.

Se o servidor de banco de dados retornar a preparação da instrução com sucesso, o método *PDO::prepare* retornará um objeto *PDOStatement*. Ora, caso o servidor de banco de dados falhar na preparação da instrução, o *PDO::prepare* retornará *FALSE* (falso) ou emitirá uma exceção *PDOException* (dependendo do tratamento de erros).

Segue-se na imagem 25 um exemplo do método *PDO::prepare,* extraído do Manual PHP, com adaptação para a nossa língua vernácula, com marcadores de parâmetros do tipo nomeados (tipo 1 do parâmetro *$comandoSQL)*:

Imagem 25 – Codificação do método *prepare()* com parâmetros nomeados

```
<?php
        $calorias = 95;
        $cor = ‘Amarela’;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < :calorias AND cor = :cor';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->bindValue(':calorias', $calorias);
        $declaracao->bindValue(':cor’, $cor);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

Segue-se na imagem 26 um outro exemplo de *PDO::prepare* com marcadores de parâmetros de ponto de interrogação (tipo 2 do parâmetro *$comandoSQL)*:

Imagem 26 – Codificação do método *prepare()* com parâmetros ponto de interrogação

```
<?php
        $calorias = 95;
        $cor = ‘Amarela’;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < ? AND cor = ?';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->bindValue(1, $calorias);
        $declaracao->bindValue(2, $cor);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

Segue-se na imagem 27 um terceiro exemplo de *PDO::prepare* sem marcador de parâmetro (tipo 3 do parâmetro *$comandoSQL)*:

Imagem 27 – Codificação do método *prepare()* sem parâmetros

```
<?php
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

1.4.3.3. Método PDOStatement::bindValue

O método *PDOStatement::bindValue* vincula um valor real para um marcador de parâmetro do tipo nomeado ou ponto de interrogação, utilizada na preparação da declaração da instrução SQL.

Na imagem 28 e demonstrado a sintaxe do método *PDOStatement::bindValue* conforme Manual PHP:

Imagem 28 – Sintaxe do método *PDOStatement::bindValue*

```
public bool PDOStatement::bindValue ( mixed $parameter , mixed $value , int $data_type = PDO::PARAM_STR)
```

Fonte: The PHP Group, 2001-2017g

Os parâmetros utilizados na método *PDOStatement::BindValue* da classe *PDOStatement*, são definidos de acordo com o Portal da The PHP Group (2001-2017g):

- *$parameter:* é o Identificador do marcador de parâmetro. Para uma instrução SQL que está utilizando uma declaração preparada usando marcadores de parâmetros do tipo nomeado, este será um nome como parâmetro, como por exemplo, “:nome”.

  Para uma instrução que está utilizando uma declaração preparada usando marcadores do tipo *ponto de interrogação* (?), esta será uma indicação numérica progressiva que indexará a posição do parâmetro.
- $value: é o valor real que se liga ao parâmetro de uma declaração preparada.
- $data_type: é o tipo de dados para o parâmetro usado, como por exemplo, se o valor real for do tipo *string* usaremos o método *PDO::PARAM_STR* da classe

PDO, se for do tipo inteiro (número) o método utilizado é *PDO::PARAM_INT*, além desses dois tipos, temos também o método *PDO::PARAM_BOOL* para o tipo booleano e o método *PDO::PARAM_NULL* para o tipo nulo (THE PHP GROUP, 2001-2017g).

O método retornará *TRUE* (verdadeiro) em caso de sucesso, ou *FALSE* (falso) em caso de falha.

Segue-se abaixo dois exemplos do método *PDO::BindValue,* extraído do Manual PHP, com adaptação para a nossa língua vernácula, com marcadores de parâmetros do 1) tipo nomeados e 2) ponto de interrogação.

A seguir na imagem 29 e demonstrado um exemplo do método *PDOStatement::bindValue* com marcador de parâmetro do tipo 1:

Imagem 29 – Codificação do método *bindValue()* com parâmetros nomeados

```
<?php
        $calorias = 150;
        $cor = 'vermelha;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < :calorias AND cor = :cor';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->bindValue(:calorias,$calorias,
        PDO::PARAM_INT);
        $declaracao->bindValue(:cor, $cor, PDO::PARAM_STR);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

Segue-se na imagem 30 um outro exemplo do método *PDOStatement::bindValue* com marcador de parâmetro do tipo 2:

Imagem 30 – Codificação do método *bindValue()* com parâmetros ponto de interrogação

```
<?php
        $calorias = 150;
        $cor = 'vermelha;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < ? AND cor = ?';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->bindValue(1, $calorias, PDO::PARAM_INT);
        $declaracao->bindValue(2, $cor, PDO::PARAM_STR);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

1.4.3.4. Método PDOStatement::execute

O método *PDOStatement::execute* executa um comando preparado.

Na imagem 31 e demonstrado a sintaxe do método *PDOStatement::execute*, conforme Manual PHP:

Imagem 31 – Sintaxe do método *PDOStatement::execute*

```
public bool PDOStatement::execute(array $input_parameters)
```

Fonte: The PHP Group, 2001-2017g

Na imagem 32 pode ser visto a sintaxe do método supracitado, conforme usado no Sistema de Empregos:

Imagem 32 – Sintaxe do método *PDOStatement::execute* do Sistema de Empregos

```
$declaracao->execute();
```

Fonte: Autoria própria, 2017

Este método executa uma declaração preparada pelo método *PDO::prepare*. Se a declaração preparada estiver utilizando o método *PDOStatement::bindValue()*, basta executar o método sem nenhum parâmetro, como pode ser avaliado na imagem 21.

No parâmetros $*input_parameters,* podemos utilizar uma matriz com os valores, sendo que todos os valores serão tratados como do tipo *PDO:PARAM_STR*.

O método *PDOStatement::execute* retornará *TRUE* (verdadeiro) em caso de sucesso ou *FALSE* em caso de falha.

Segue-se abaixo dois exemplos do método *PDOStatement::execute,* extraído do Manual PHP, com adaptação para a nossa língua vernácula, um com uma declaração preparada utilizando 1) *PDOStatement::bindValue* e a outra usando 2) Matriz de valores.

A seguir na imagem 33 pode ser visto um exemplo executando uma declaração preparada utilizando 1) *PDOStatement::bindValue:*

Imagem 33 – Codificação do método *execute()* com o método *bindValue()*

```
<?php
        $calorias = 20;
        $cor = ‘verde;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < ? AND cor = ?';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->bindValue(1, $calorias, PDO::PARAM_INT);
        $declaracao->bindValue(2, $cor, PDO::PARAM_STR);
        $declaracao->execute();
?>
```

Fonte: The PHP Group, 2001-2017g

Segue-se na imagem 34 um outro exemplo executando uma declaração preparada utilizando 2) Matriz de valores*:*

Imagem 34 – Codificação do método execute*()* com Matriz de valores

```
<?php
        $calorias = 20;
        $cor = ‘verde;
        $comandoSQL = 'SELECT nome, cor, calorias FROM fruta
        WHERE calorias < ? AND cor = ?';
        $declaracao = $conexao->prepare($comandoSQL);
        $declaracao->execute(array($calorias, $cor);
?>
```

Fonte: The PHP Group, 2001-2017g

1.4.3.5. Método PDOStatement::fetch

O método *PDOStatement::fetch* é utilizado para obter a próxima linha de um conjunto de resultados.

Na imagem 35 e demonstrado a sintaxe do método *PDOStatement::fetch*, conforme usado no Sistema de Empregos:

Imagem 35 – Sintaxe do método *PDOStatement::fetch* do Sistema de Empregos

```
$resultado = $declaracao->fetch(PDO::FETCH_NUM);
```

Fonte: Autoria própria, 2017

Este método obtém uma linha a partir de um conjunto de resultados associada a um objeto *PDOStatement*.

O parâmetro *fetch_style* determina como o objeto retornará a linha. Esse parâmetro controla como a próxima linha será devolverá a chamada. Se no método não for

atribuído nenhum *fetch_style* o padrão é *PDO::FETCH_BOTH*, que retornará uma matriz indexada com número e o nome da coluna como retorno no conjunto de resultados; a título de informação, há muito outros tipos de *fetch_style*, por exemplo, *PDO::FETCH_ASSOC, PDO::FETCH_NUM, PDO::FETCH_OBJ, etc*.

O retorno do valor deste método, em caso de sucesso, dependerá do tipo da busca. Em caso de falha, independente do tipo de busca, conforme determinado no parâmetro *fetch_style*, o valor devolvido será *FALSE* (falso).

Segue-se na imagem 36 um exemplo do método *PDOStatement::fetch,* extraído do Manual PHP, com adaptação para a nossa língua vernácula, utilizando o *fetch_style PDO::FETCH_NUM*:

Imagem 36 – Codificação do método *fetch()* com parâmetro FETCH_NUM

```
<?php
        $declaracao = $conexao->prepare("SELECT nome, cor FROM
        fruta");
        $declaracao->execute();
        $resultado = $declaracao->fetch(PDO::FETCH_NUM);
        print_r($resultado); //mostra o resultado
?>
```

Fonte: The PHP Group, 2001-2017g

No exemplo ilustrado na imagem 36, o resultado exibido pelo código, pode ser visto na imagem 37:

Imagem 37 – Resultado da codificação do método *fetch()* com parâmetro *FETCH_NUM*

```
Array (
        [0] => Abacate
        [1] => Verde
)
```

Fonte: The PHP Group, 2001-2017g

1.4.3.6. Método PDO::lastInsertId

O método *PDO::lastInsertId* retornará um ID (identificador) da última linha inserida ou o valor da sequência.

Na imagem 38 e demonstrado a sintaxe do método *PDO::lastInsertId*, conforme usado no Sistema de Empregos:

Imagem 38 – Sintaxe do método *PDO::lastInsertId* do Sistema de Empregos

```
lastInsertId ()
```

Fonte: Autoria própria, 2017

Este método retorna um ID (identificador) da última linha inserida recentemente em uma tabela no banco de dados. No exemplo que veremos, foi inserida no banco de dados a entrada de um usuário com seu código, o IP da rede do servidor e o tipo do usuário; ao fazer estas introduções será gerado um ID que poderá ser resgatado, desde que se utilize o método *PDO::lastInsertID*, a fim de registrar o último *login* acessado.

Segue-se na imagem 39 um exemplo do método *PDO::lastInsertID,* instrumentalizado no Sistema de Empregos:

Imagem 39 – Codificação do método *lastInsertID()*

```
<?php
        $sql = "INSERT INTO login (ID, IP, TIPO_USUARIO) VALUES
        (:id,:server, :status);";
        $declaracao = $conexao->prepare($sql);
        $declaracao->bindValue(':id',$row[0]);
        $declaracao->bindValue(':server',$_SERVER["REMOTE_ADDR"]);
                $declaracao->bindValue(':status','U');
        $declaracao->execute();
        //registrando ultimo login
        if($declaracao = $conexao->prepare($sql)){
                $sqlCod = "UPDATE aluno SET ULTIMO_LOGIN = :insertId
WHERE ALUNO = :id";
                $insertId = $conexao->lastInsertId();
        $declaracao = $conexao->prepare($sqlCod);
        $declaracao->bindValue(':insertId', $insertId);
        $declaracao->bindValue(':id',$row[0]);
        $declaracao->execute();
        }
?>
```

Fonte: Autoria própria, 2017

1.4.4. API MySQLi

A API MySQLi ou MySQL improved (aprimorada) foi criada para aperfeiçoar os novos recursos descobertos nas versões do banco de dados MySQL 4.1.3 e posteriores, inclusa na versão 5 do PHP e ativa, até o presente momento em todas as versões 7.x do PHP. Por suportar a interface processual, que é bem parecida com a API MySQL, muitos desenvolvedores resolvem utilizar a API MySQLi para realizar a migração das funções MySQL para MySQLi.

Segundo o *site* Manual do PHP, foram incorporados vários benefícios na extensão MySQLi são:

- Interface orientada a objetos ou processual;
- Suporte para declarações preparadas;
- Suporte para várias declarações;
- Suporte para Transações;
- Capacidades de depuração aprimoradas;
- Suporte de servidor incorporado (THE PHP GROUP, 2001-2017f)

As informações aqui disponibilizadas foram apresentadas apenas a título de conhecimento, pelo fato de muitos desenvolvedores utilizarem a API MySQLi. No entanto, elas não são objeto de análise deste Trabalho de Graduação, que tem como base da reflexão técnico-computacional apenas as APIs MySQL e PDO.

1.5. Injeção SQL

A injeção SQL tem a capacidade de expor todas as informações relevantes armazenadas em uma aplicação banco de dados, incluindo itens confidenciais, como senhas ou detalhes do cartão de crédito ou ainda dados de interesse mais específico ao que pretende o atacante, como um simples nome, endereço ou telefone, que podem ser utilizados para identificar pessoas ou localidades. Por esse motivo o código SQL injetado em qualquer campo de formulário web é considerado uma das vulnerabilidades mais danosas às interações comerciais (CLARKE, 2009, p. 2). A *Open Web Application Security Project* (OWASP), uma comunidade aberta dedicada a permitir que as organizações desenvolvam, comprem e mantenham aplicativos e API que possam ser confiáveis, elencou em seu Top 10 2017 os dez riscos de

segurança da aplicação mais críticos da web, no qual cita que a injeção SQL continua em primeiro lugar entre os principais causadores de danos em aplicações web, posição esta ocupada desde de 2013, quando esse Top 10 foi publicado pela última vez.

Essa organização se define como uma inovação no mercado tecnológico, conforme divulgado:

> A OWASP Foundation é a entidade sem fins lucrativos que garante o sucesso a longo prazo do projeto. Quase todos os associados com a OWASP são voluntários, incluindo o Conselho da OWASP, Líderes do Capítulo, Líderes do Projeto e membros do projeto. Apoiamos pesquisa de segurança inovadora com bolsas e infraestrutura. (...). Nossa liberdade de pressões comerciais nos permite fornecer informações imparciais, práticas e econômicas sobre segurança de aplicativos. A OWASP não é afiliada a nenhuma empresa de tecnologia, embora apoiemos o uso informado da tecnologia de segurança comercial. OWASP produz muitos tipos de materiais de forma colaborativa, transparente e aberta (THE OWASP FOUNDATION, 2017a, p. 2).

Muito conhecida como SQL *injection*, a injeção SQL oferece ao invasor a possibilidade de manipular as consultas SQL que um aplicativo transfere para uma base de dados, desse modo o invasor pode utilizar a sintaxe e os recursos da linguagem SQL[1], o que configura numa vulnerabilidade.

Conforme conceitua o documento Top 10 2017 da OWASP,

> As falhas de injeção, como SQL, (...), ocorrem quando os dados não confiáveis são enviados para um intérprete como parte de um comando ou consulta. Os dados hostis do invasor podem enganar o intérprete para executar comandos não intencionais ou acessar dados sem a devida autorização (THE OWASP FOUNDATION, 2017a, p. 7).

Em 1998 Rain Forest Puppy publicou um artigo para Phrack, intitulado "NT Web Technology Vulnerabilities", cujo público alvo era constituído por hackers. Nesse artigo, Puppy ganhou notoriedade pública pela descoberta da vulnerabilidade oferecida pela injeção SQL, uma vez que, possivelmente, essa fragilidade já existia desde a primeira conexão entre a aplicação e a base de dados (CLARKE, 2009, p. 4).

Puppy alertou, no início de 2000, em outro artigo, nomeado “How I hacked PacketStorm”, publicado no endereço eletrônico “www.wiretrip.net/rfp/txt/rfp2k01.txt”,

[1] *Structured Query Language* é uma linguagem baseada na álgebra relacional, que é utilizada para manipular informações contidas em bancos de dados relacionais (UTO, 2013, p. 1).

como a injeção de SQL foi utilizada para atacar um site popular. A partir desse alerta, muitos pesquisadores estudam a injeção SQL, entretanto ela ainda representa um desafio para os profissionais da Segurança da Informação (CLARKE, 2009, p. 4).

Os três principais tipos de ataques comuns à injeção SQL são a) a injeção de SQL cega, baseada em Boolean, ou inferencial; b) a injeção de SQL cega baseada em tempo; e c) a injeção de SQL em erro, de acordo com Dougherty (2011):

- Injeção de SQL cega baseada em Boolean (às vezes referida como injeção de SQL inferencial): múltiplas declarações válidas que avaliam como verdadeiro e falso são fornecidas no parâmetro afetado na solicitação HTTP. Ao comparar a página de resposta entre ambas as condições, a ferramenta pode inferir se a injeção foi ou não bem-sucedida.
- Injeção de SQL cega baseada em tempo (às vezes referida como injeção de SQL completo): as declarações de SQL válidas são fornecidas no parâmetro afetado na solicitação de HTTP que faz com que o banco de dados faça uma pausa por um período de tempo específico. Ao comparar os tempos de resposta entre solicitações normais e solicitações injetadas várias vezes, uma ferramenta pode determinar se a execução da instrução SQL foi bem-sucedida.
- Injeção SQL baseada em erro: as declarações SQL inválidas são fornecidas ao parâmetro afetado na solicitação HTTP. A ferramenta, em seguida, monitora as respostas HTTP para mensagens de erro, que se sabe que se originaram no servidor do banco de dados (DOUGHERTY, 2011, p. 3).

As prevenções para esses tipos de ataques podem ser minimizadas, conforme sugestões da OWASP (2017), da seguinte maneira:

1) Ao usar APIs seguras e parametrizadas no acesso à base de dados;
2) Caso os comandos SQL sejam por meio de *strings* concatenadas, deve-se analisar minuciosamente todas as entradas de informações inseridas pelos usuários, tratando todos os tipos de caracteres especiais na codificação da aplicação web;
3) Uso do método de validação de entrada ou “lista branca”, que procura bloquear todas as entradas de informações, menos àquelas que adotarem o padrão definido como válido (THE OWASP FOUNDATION, 2017a, p. 8).

1.6. Conceituação de Segurança da Informação

A Segurança da Informação tem o objetivo de minimizar as ocorrências e a gravidade de incidentes como desastres, erros (intencionais ou não) e manipulação não autorizada de informações ou sistemas.

A base de um sistema seguro se constitui a partir de princípios essenciais da segurança da informação como confidencialidade, integridade e disponibilidade que devem ser garantidos para que a informação permaneça livre de qualquer intervenção externa.

Esses princípios são definidos, segundo Coelho, Araújo e Bezerra (2014), como:

- Confidencialidade: é a proteção de dados contra acessos não autorizados, para garantir o sigilo de uma informação, como por exemplo uma senha de um usuário, que deve ser acessada apenas pelo próprio e ser restrita aos outros;
- Integridade: é a manutenção da informação livre de alterações ou remoções não autorizadas, que podem ocorrer quando uma determinada informação fica exposta ao acesso e manipulação não autorizados;
- Disponibilidade: é a garantia de acesso à informação sempre que necessário a quem ela é permitida, para isso essa informação deve estar protegida contra perdas ou degradações (*Cf*. COELHO; ARAÚJO; BEZERRA, 2014, p. 6-7).

Sob a intenção de assegurar que os princípios essenciais da Segurança da Informação sejam aplicados e de proteger os dados que interagem nas aplicações web usam-se controles estabelecidos, implementados, monitorados e analisados criticamente, a fim de garantir a segurança desses dados, como o teste de segurança e penetração que será dissertado na próxima seção.

1.7. Ferramenta de Teste de Penetração

1.7.1. Teste de segurança

O teste de segurança de uma aplicação web é um processo de teste e avaliação, na qual se descobre vulnerabilidades e riscos de segurança no sistema.

Segundo o documento introdutório do Programa ZAP, a palavra teste está definida como a descoberta e tentativa de exploração de vulnerabilidades e a palavra avaliação

como a análise e descoberta de vulnerabilidades sem a exploração dessas vulnerabilidades (THE OWASP FOUNDATION, 2017b, p. 1).

Muitas vezes exposto de uma forma arbitrária o teste de segurança tem vários tipos de saídas, de acordo com a Fundação OWASP, o tipo do teste ou a vulnerabilidade testada são:

- Avaliação de Vulnerabilidade – A aplicação web é escaneada e analisada por problemas de segurança;
- Teste de penetração – A aplicação web passa por análise e simulação de ataques mal-intencionados;
- Teste de tempo de execução – A aplicação web é submetida a análises e testes de segurança a partir de um usuário final;
- Revisão de Código – A codificação da aplicação web é submetida por uma minuciosa revisão e análise visando especificamente as fraquezas de segurança (THE OWASP FOUNDATION, 2017b, p. 1).

A avaliação de risco não será listada como parte do teste de segurança, porque a avaliação de risco não é verdadeiramente um teste, mas sim uma análise das importâncias de diferentes riscos (segurança do pessoal, hardware e software, etc.) e algumas medidas de reduções para esses riscos.

1.7.2. Testes de Penetração

Pentesting (teste de penetração) simula um invasor malicioso com o objetivo de acessar o sistema, podendo assim realizar alterações e roubos dos dados. É utilizado para verificar os mecanismos de defesa, os planos de respostas e a confirmação da adesão à política de segurança. Há três tipos de *pentesting*: 1) o manual, 2) automático ou 3) a junção dos dois; todos examinam tudo dos servidores, as redes e até os dispositivos, mas esta documentação se concentra no teste de penetração em aplicações web (THE OWASP FOUNDATION, 2017b, p. 2).

O teste automatizado é um componente importante da validação de integração contínua, o que significa que, ajuda a descobrir novas vulnerabilidades em um ambiente que se altera rapidamente, auxilia também nas regressões das

vulnerabilidades anteriores, na qual o desenvolvimento pode ser extremamente colaborativo e distribuído (THE OWASP FOUNDATION, 2017b, p. 2).

Geralmente as etapas para um teste de penetração são, de acordo com a Fundação OWASP:

- Explorar – O testador tenta conhecer sobre o sistema que está sendo explorado, capturando todos os pontos de extremidades existentes, quais os softwares que estão sendo usados, quais patches (remendos ou correções) estão instalados, etc. É incluído também as pesquisas dos conteúdos escondidos no site, as vulnerabilidades conhecidas e outras fraquezas.
- Ataque – O testador tenta atacar as vulnerabilidades suspeitas ou conhecidas, provando assim a existência do mesmo.
- Relatório – O testador cita os resultados dos testes, inserindo as vulnerabilidades, como foi explorada e o nível de rigidez da exploração (THE OWASP FOUNDATION, 2017b, p. 2).

A meta final de um teste de penetração é encontrar as vulnerabilidades da aplicação web para que possam ser analisadas e mitigadas.

1.7.3. Zed Attack Proxy (ZAP)

O programa ZAP (Zed Attack Proxy) é uma ferramenta de teste de penetração open source (código aberto), mantida pela organização OWASP. Este programa foi projetado para testar a segurança das aplicações web.

Também muito conhecido como um "*proxy* de interceptação", pois todas as requisições e respostas enviadas entre o navegador (browser) e a aplicação web (Web Application) são interceptadas e examinadas, podendo assim modificar o conteúdo, e enviar os pacotes para o destino (THE OWASP FOUNDATION, 2017b, p. 2), conforme observado na imagem 40.

Imagem 40 – Entrada o programa ZAP entre o navegador e a aplicação

Fonte: The OWASP Foundation, 2017b

CAPÍTULO 2 – ESTUDO DE CASO

Para a realização do estudo de caso, utilizamos o programa XAMPP, que tem como propósito a simulação de um servidor web local, e que será utilizado para a hospedagem das aplicações apiMySQL e apiPDO.

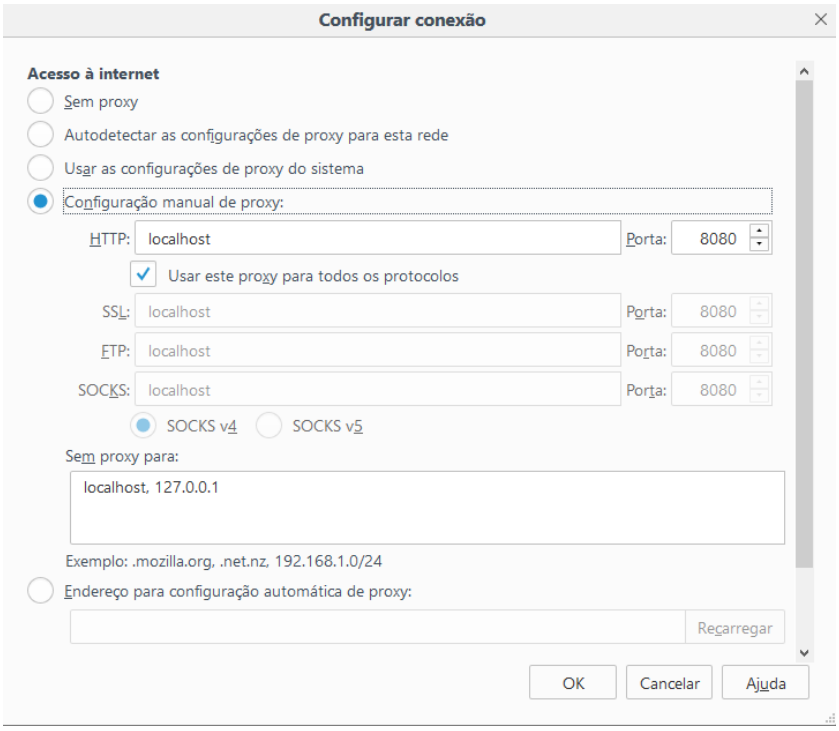
O significado do nome XAMPP, de acordo com o sítio eletrônico deste programa supracitado, origina-se da abreviação de **X**, ou seja, instalável em diferentes sistemas operacionais, como, Windows, Linux e OS X, **A**pache, que é o Servidor HTTP, **M**ySQL ou **M**ariaDB, ou seja, ambos são Sistemas Gerenciadores de Banco de Dados, **P**HP, isto é, o interpretador da linguagem PHP, e **P**erl, que corresponde ao interpretador da linguagem Perl (APACHE FRIENDS, 2017).

Conforme mencionado, foi instalado o programa ZAP, uma ferramenta de testes de penetração, que tem como objetivo encontrar vulnerabilidades nas aplicações Web.

Após a instalação desses dois programas, é necessária a configuração do proxy no navegador Firefox, a fim de capturar a estrutura da navegação com o auxílio do programa ZAP.

Para configurar o proxy manual do navegador Firefox requer-se do usuário o cumprimento das seguintes etapas: assinalar “Opções”, depois, “Avançado” e, posteriormente, na aba “Rede”; selecionar “Configuração manual de proxy:”, inserir localhost em “HTTP” e 8080 na “Porta” e, por último, recomenda-se fazer um check em “Usar este proxy para todos os protocolos”, conforme descrito na imagem 41:

Imagem 41 – Configuração da conexão proxy



Fonte: Autoria própria (2017) com base no Navegador Firefox

Com o programa ZAP iniciado, inserimos na barra de endereço do navegador o IP do servidor ou o host, juntamente com o nome da pasta no qual se encontra o sistema, conforme imagem 42:

Imagem 42 – Endereço eletrônico da aplicação web apiMySQL

Fonte: Autoria própria (2017) com base no Navegador Firefox

O mesmo procedimento técnico foi adotado para a realização da configuração manual do *proxy* no navegador indicado, mas agora na aplicação web apiPDO, conforme ilustração 43:

Imagem 43 – Endereço eletrônico da aplicação web apiMySQL

Fonte: Autoria própria (2017) com base no Navegador Firefox

Para que a ferramenta de scanner de vulnerabilidades web capture todas as requisições POST e GET é necessário percorrer todas as funcionalidades do sistema, cadastrando e preenchendo todos os formulários das páginas, só assim a ferramenta coletará toda a estrutura da aplicação.

Em seguida, indicaremos uma imagem da página inicial (imagem 44) da aplicação web, sendo que no topo se encontra os campos para acesso dos alunos e empresas cadastradas; abaixo da barra de login, teremos o logotipo da instituição Fatec Americana, posteriormente, há o menu e as imagens de cada curso oferecido por esta Instituição de Ensino Superior, com as vagas disponíveis por curso.

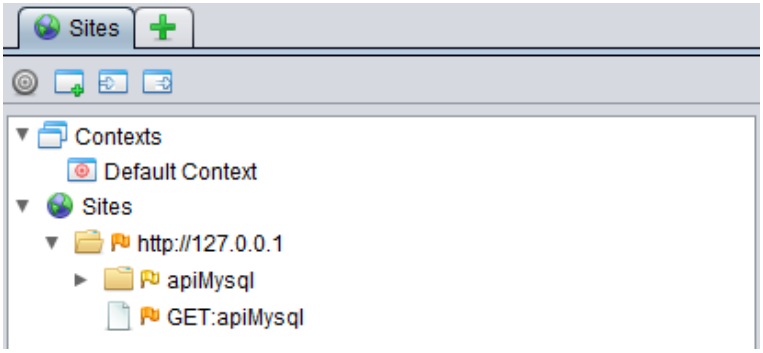
Imagem 44 – Página inicial do sistema de empregos da Fatec Americana

Fonte: Autoria própria (2017) baseada em Fatec-Am (2015)

Percorridas todas as páginas do sistema indicado e após o cadastramento de todos os formulários da aplicação, podemos verificar no programa ZAP o começo da estrutura do *site*, conforme indicada na Imagem 45.

Em função do caráter sigiloso das informações contidas na estrutura, não disponibilizaremos toda a árvore do *site* coletada pelo programa.

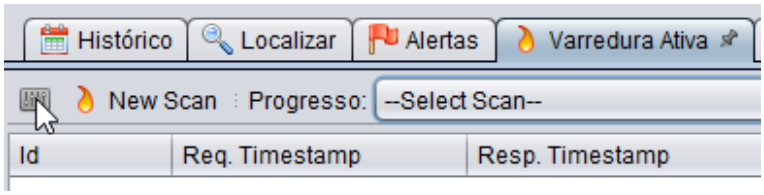
Imagem 45 – Estrutura do sistema de empregos da Fatec Americana com API MySQL



Fonte: ZAP *In*. OWASP (2017)

Com toda a estrutura capturada podemos fazer uma varredura ativa na aplicação, a qual irá realizar um teste de penetração em todas as funções POST e GET existentes nas páginas, inserindo os ataques de injeção SQL. Para iniciar uma varredura na aplicação, basta o usuário escolher a aba “Varredura Ativa”. Como faremos uma varredura apenas com os ataques de injeção SQL, será necessária a criação de uma política de varredura, que se localiza ao lado esquerdo do botão “New Scan”, conforme a imagem 46:

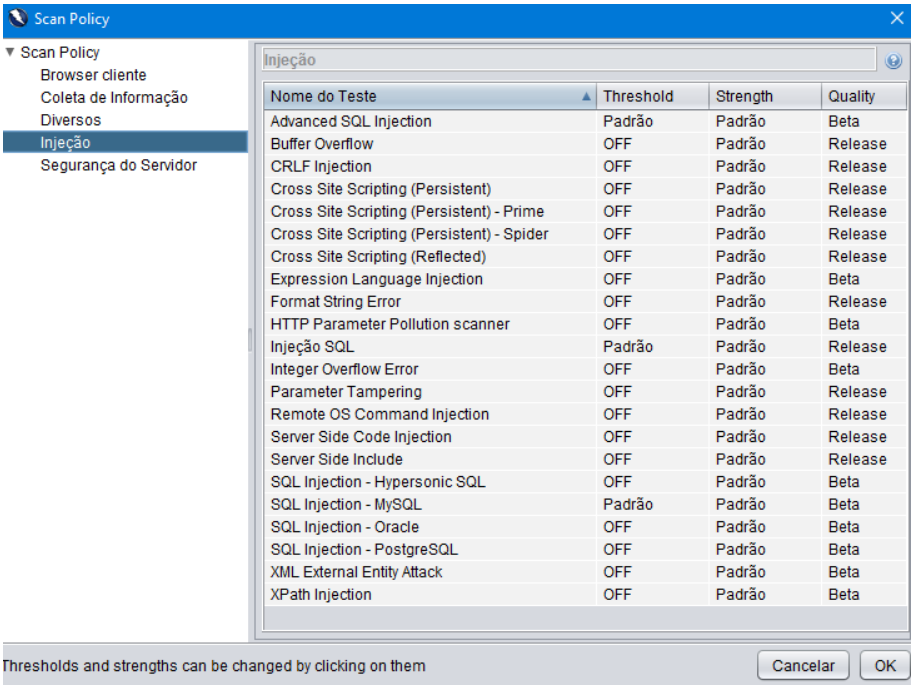
Imagem 46 – Botão de configuração para a Política de Varredura



Fonte: ZAP *In*. OWASP (2017)

Na política de varredura foi criada uma política apenas com os ataques de injeção SQL, o restante dos ataques foi desabilitado, conforme visto na imagem 47, e o nome dado à política foi “SQLInjection”.

Imagem 47 – Política de varredura do Programa ZAP

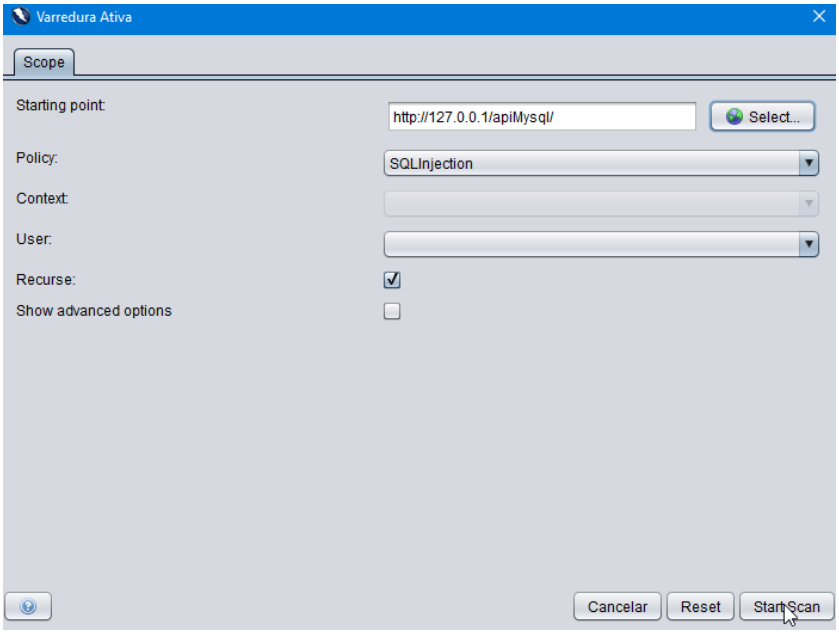


Fonte: ZAP *In.* OWASP (2017)

Com a política de varredura criada, podemos gerar a varredura ativa acionando o botão “New Scan” e selecionar o ponto de partida, cujo enderenço é

“http://127.0.0.1/apiMysql”, e adicionar a política anteriormente denominada “SQLInjection” na política de varredura, e assim, iniciar o processo de escaneamento, conforme imagem 48 a seguir:

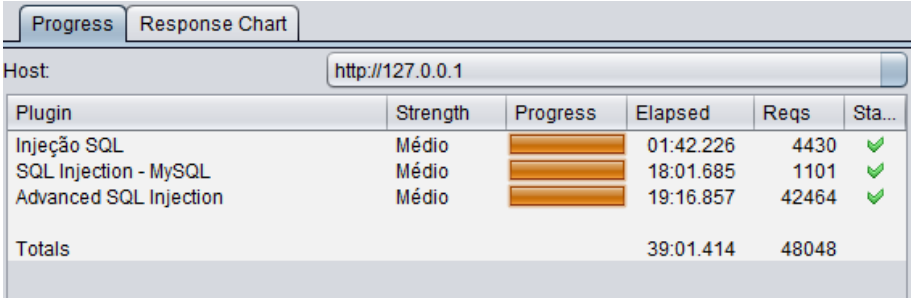
Imagem 48 – Varredura ativa do Programa ZAP



Fonte: ZAP *In.* OWASP (2017)

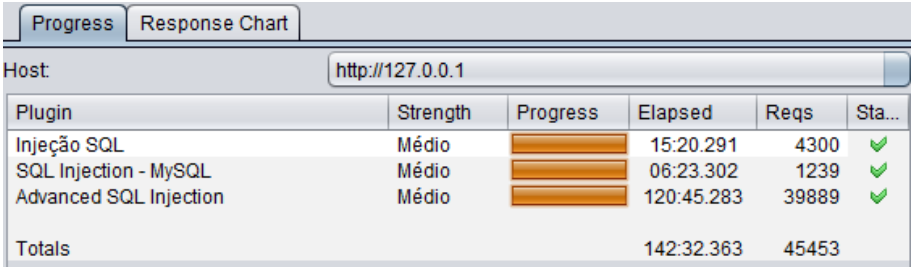
Nos escaneamentos realizados nas duas aplicações, cada um foi gerado uma tabela na qual é possível visualizar o tipo de ataque (*plugin*), a força do ataque (*Strength*), o progresso, o tempo, a quantidade de requisições enviadas e o status, de acordo com a imagem 49 que é da apiMysql e a imagem 50 da apiPDO:

Imagem 49 – Progresso da Varredura ativa do Programa ZAP da apiMysql



Fonte: ZAP *In.* OWASP (2017)

Imagem 50 – Progresso da Varredura ativa do Programa ZAP da apiPDO



Fonte: ZAP *In.* OWASP (2017)

Na tabela 3 pode ser visto os dados da comparação das varreduras ativas entre as aplicações web apiMysql e apiPDO, foi levado em consideração o tempo e a quantidade de requisições injetadas em cada tipo de ataque, assim como o total de tempo e requisições da varredura ativa:

Tabela 3 – Comparação das aplicações web em relação ao tempo e requisições

App	**Ataques**						**Total de Tempo e Requisições**	
	Injeção SQL		**SQL I.- MySQL**		**Advanced SQL I.**			
	T	**R**	**T**	**R**	**T**	**R**	**TT**	**TR**
apiMysql	1min42s	4430	18min1s	1101	19min16s	42464	39min1s	48048
apiPDO	15min20s	4300	6min23s	1239	120min45s	39889	142min32s	45453
Legendas	• App = Aplicação • I. = Injection • T = Tempo • R = Requisições • TT = Total de Tempo • TR = Total Requisições							

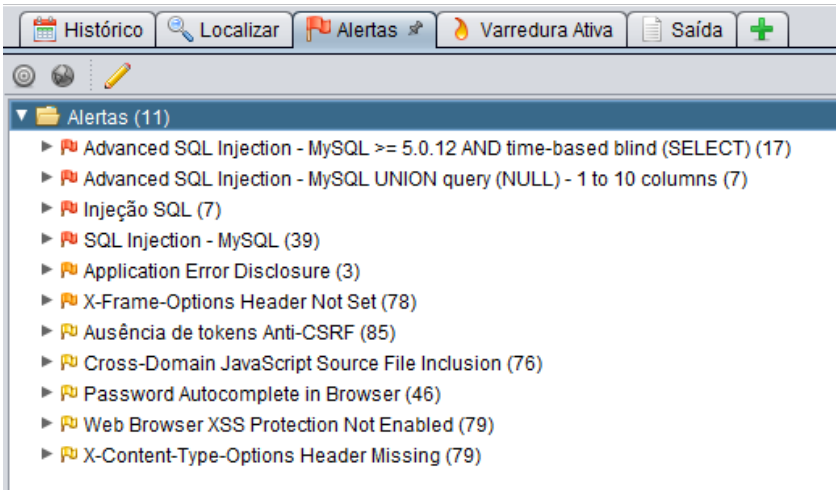
Fonte: Autoria própria (2017) com base no Programa ZAP

Em seguida, ao verificar a aba “Alertas”, visualizamos o tipo e a quantidade de ataques que a aplicação pode estar suscetível, sendo assim, com a aplicação apiMysql, foram encontradas as seguintes vulnerabilidades:

1) 17 do tipo Advanced SQL Injection – MySQL;
2) 7 do tipo Advanced SQL Injection – MySQL UNION query (NULL);
3) 7 do tipo Injeção SQL e, por último,
4) 39 do tipo SQL Injection – MySQL.

As informações supracitadas, podem ser conferidas na imagem 51:

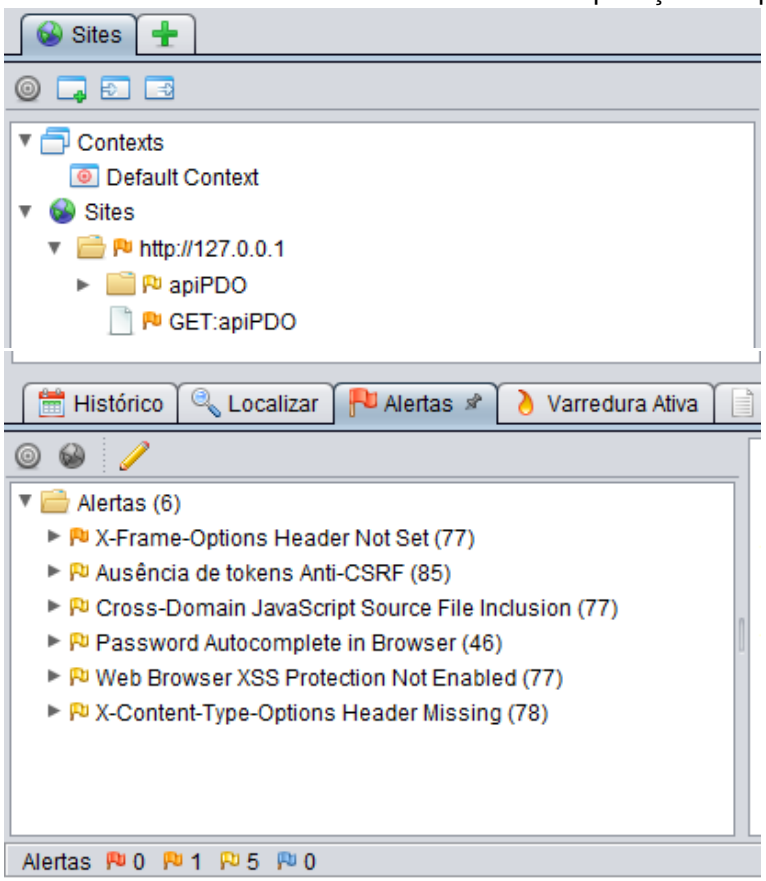
Imagem 51 – Resultado das vulnerabilidades com a aplicação da apiMysql



Fonte: ZAP *In.* OWASP (2017)

Não obstante, com a aplicação apiPDO, a ferramenta de escaneamento de vulnerabilidades web, não conseguiu encontrar nenhuma entrada utilizando o ataque de injeção SQL, conforme pode ser visualizado na imagem 52:

Imagem 52 – Resultado das vulnerabilidades com a aplicação da apiPDO



Fonte: ZAP *In.* OWASP (2017)

Abaixo é apresentada a Tabela 4 que ilustra a quantidade de vulnerabilidade nas aplicações web.

Tabela 4 – Quantidade de vulnerabilidades nas aplicações web

Tipos de Ataques	Quantidade de vulnerabilidades encontradas:	
	apiMysql	apiPDO
Advanced SQL I. (baseado em tempo)	17	0
Advanced SQL I. (baseado em erro)	7	0
Injeção SQL (baseado em Boolean)	7	0
SQL I. - MySQL (baseado em Boolean)	39	0

Fonte: Autoria própria (2017) com base no Programa ZAP

Nesse sentido, podemos verificar na tabela 4 que a aplicação web apiMysql está muito mais suscetível aos ataques de injeção SQL por fazer o uso de uma API já depreciada e descontinuada, ao passo que na aplicação web apiPDO, por instrumentalizar uma API com o *prepared stantements*, se mostrou mais blindada a estes tipos de ataques.

CONSIDERAÇÕES FINAIS

O presente Trabalho de Graduação teve como propósito demonstrar que uma aplicação web implementada com a API MySQL pode estar vulnerável a ataques de injeção SQL, não sendo possível, por conseguinte, garantir a confiabilidade, integridade e a disponibilidade das informações armazenadas na base de dados.

Verificamos ainda a eficácia da API PDO que, ao ser implementada na aplicação web, se mostrou 100% segura na proteção contra injeções SQL.

Como demonstrado, no início dos primórdios da aplicação web na Internet, apenas as páginas estáticas já continham disponíveis todos os dados relevantes para os usuários. Com a evolução da página web estática para as páginas web dinâmicas foi necessária a criação de um banco de dados para a realização dos armazenamentos das informações.

No intuito de disponibilizar para os usuários uma aplicação web, foram conceituados na Arquitetura web as tecnologias utilizadas na implementação. Ainda foi definido a linguagem de programação PHP, usada para a codificação da aplicação web, o banco de dados MySQL para o armazenamento dos dados, além das APIs MySQL e PDO, que foram instrumentalizadas na migração de uma para outra. Explanamos também sobre os ataques de injeção SQL mais conhecidos, assim como orientamos algumas formas de prevenções que podem minimizar esses tipos de ataques.

Por conseguinte, foi considerado os conceitos da segurança da informação para conscientizar os desenvolvedores de sistemas web, que vulnerabilidades de injeção SQL podem acarretar em perdas, erros e até na disseminação de dados privados.

Foi utilizado, por fim, uma ferramenta de teste de penetração que auxiliou na simulação de um invasor que injetou os ataques SQL nas aplicações com as APIs MySQL e PDO. Como resultado, constatamos que o uso de uma API desatualizada, depreciada e que não faz o uso de consultas parametrizadas, no caso a API MySQL, podem abrir brechas de segurança na aplicação web, ocasionando indisponibilidade, erros, alterações, furtos dos dados armazenados no banco de dados. Contudo, a utilização da API PDO, a qual recebe todas as correções atuais e detém as consultas parametrizadas, é uma aliada do desenvolvedor no impedimento de injeções SQL.

Entretanto, a título de complementação e de futuras pesquisas, vale ressaltar que o uso do *prepared statemen*t não é o único meio de defesa do ataque de injeção SQL,

conforme ilustra o artigo “SQL Injection Prevention Cheat Sheet”, da OWASP. A título de informação o artigo publicado pela OWASP fornece orientações claras, simples e acionáveis para prevenir falhas de Injeção SQL em seus aplicativos, visto que estes ataques são, infelizmente, muito comuns, em função de dois fatores: a) a prevalência significativa de vulnerabilidades de injeção de SQL; e b) a atratividade do alvo, isto é, o banco de dados geralmente contém todos os dados interessantes e/ou críticos para sua aplicação. Com efeito, podemos avaliar como algo vexatório que haja tantos ataques de Injeção de SQL bem-sucedidos, sobretudo quando nós constatamos o quanto é simples evitar vulnerabilidades de Injeção SQL em seu código. Ademais, as falhas de injeção de SQL são introduzidas quando os desenvolvedores de software criam consultas dinâmicas de banco de dados, que incluem entrada fornecida pelo usuário. Para evitar falhas de injeção SQL os desenvolvedores precisam: a) parar de escrever consultas dinâmicas; e/ou b) impedir que a entrada fornecida pelo usuário que contenha SQL mal-intencionado afete a lógica da consulta executada.

Este trabalho teve como intenção fornecer um conjunto de técnicas simples para prevenir vulnerabilidades de Injeção de SQL, objetivando evitar os problemas supracitados. Essas técnicas, por exemplo, podem ser usadas com praticamente qualquer tipo de linguagem de programação com qualquer tipo de banco de dados. Não obstante, acreditamos que essas perspectivas podem ser objeto de análise de outros Trabalhos de Graduação, haja vista que estas linhas não exaurem a compreensão e a complexidade que esta temática envolve.

REFERÊNCIAS

APACHE FRIENDS. **Sobre**, 2017. Disponível em: <https://www.apachefriends.org/pt_br/about.html>. Acesso em: 13 nov. 2017

CANALTECH. **O que é API?**, 2017. Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 1 nov. 2017

CONSEIL EUROPÉEN POUR LA RECHERCHE NUCLÉAIRE. **World Wide Web**, 2017. Disponível em: <http://info.cern.ch/hypertext/WWW/TheProject.html>. Acesso em: 8 ago. 2017. (Imagem 1 - World Wide Web)

CLARKE, J. **SQL Injection Attacks and Defense**. Burlington: Elsevier, 2009.

COELHO, F. E. S.; ARAÚJO, L. G. S. DE; BEZERRA, E. K. **Gestão da Segurança da Informação**: NBR 27001 e NBR 27002. Rio de Janeiro: RNP/ESR, 2014.

CONVERSE, T.; PARK, J. **PHP5 and MySQL Bible**. 2ª ed. São Paulo: Elsevier, 2002.

CORREIOS. **Calculador Remoto de Preços e Prazos**, 2017. Disponível em: <https://www.correios.com.br/para-sua-empresa/servicos-para-o-seu-contrato/precos-e-prazos>. Acesso em: 10 out. 2017.

DB-ENGINES. **DB-Engines Ranking:**Trend Popularity, 2017. Disponível em: <https://db-engines.com/en/ranking_trend>. Acesso em: 1 nov. 2017. (Gráfico 2 - DB-Engines Ranking - Trend Popularity)

DOUGHERTY, C. **Practical Identification of SQL Injection Vulnerabilities**. 15 p., 2011. Disponível em <https://www.us-cert.gov/sites/default/files/publications/Practical-SQLi-Identification.pdf>. Acesso em: 13 nov. 2017.

GILMORE, W. J. **Beginning PHP and MySQL**: From Novice to Professional. 4ª ed. New York: Apress, 2010.

HOLANDA, M. T.; FERNANDES, J. H. C. **Segurança no desenvolvimento de aplicações**. Disponível em: <http://mauriciolyra.pro.br/site/wp-

content/uploads/2015/12/22-Seguranca_Desenvolvimento_Aplicacoes.pdf>. Acesso em: 20 ago. 2017.

LOCKHART, J. **PHP Moderno**: Novos Recursos e Boas Práticas. São Paulo: Novatec, 2015.

MILANI, A. **MySQL**: Guia do Programador. São Paulo: Novatec, 2006.

_____. **Construindo aplicações web com PHP e MySQL**. 2ª ed. São Paulo: Novatec, 2016.

MISCELÂNEA DO CONHECIMENTO. **PHP Data Objects**: PDO, 2017. Disponível em: <http://www.miscelaneadoconhecimento.com.br/mysql/sql/iframepdo.html>. Acesso em. 02 de nov. 2017.

MYSQL. **MySQL Customers**, 2017. Disponível em: <https://www.mysql.com/customers/>. Acesso em: 11 ago. 2017.

STUTTARD, D.; PINTO, M. **The web application hacker's handbook**: Finding and Exploiting Security Flaws. 2ª ed. Indianapolis: Wiley Publishing, 2011.

THE OWASP FOUNDATION. **OWASP Top 10 2017**: The Ten Most Critical Web Application Security Risks. 26p., 2017a. Disponível em: <https://www.owasp.org/images/b/b0/OWASP_Top_10_2017_RC2_Final.pdf>. Acesso em: 13 nov. 2017.

_____. **OWASP Zap 2.6**: Getting Started Guide. 10p., 2017b. Disponível em: <https://github.com/zaproxy/zaproxy/releases/download/2.6.0/ZAPGettingStartedGuide-2.6.pdf>. Acesso em: 13 nov. 2017.

THE PHP GROUP. **Choosing an API**, 2017a. Disponível em: <http://php.net/manual/pt_BR/mysqlinfo.api.choosing.php>. Acesso em: 8 mar. 2017

_____. **Current Stable PHP 7.1.11**, 2017b. Disponível em: <https://secure.php.net/downloads.php>. Acesso em: 30 out. 2017

_____. **Extensões de Banco de Dados**, 2017c. Disponível em:

<http://php.net/manual/pt_BR/refs.database.php>. Acesso em: 10 ago. 2017

_____. **Funções da MySQL**, 2017d. Disponível em: <http://php.net/manual/pt_BR/ref.mysql.php>. Acesso em: 10 ago. 2017

_____. **MySQL Improved Extension**, 2017e. Disponível em: <http://php.net/manual/en/book.mysqli.php>. Acesso em: 10 ago. 2017

_____. **O que o PHP pode fazer?**, 2017f. Disponível em: <http://php.net/manual/pt_BR/intro-whatcando.php>. Acesso em: 8 ago. 2017

_____. **The PDO Class**, 2017g. Disponível em: <http://php.net/manual/pt_BR/class.pdo.php>. Acesso em: 8 ago. 2017

UTO, N. **Teste de Invasão de Aplicações Web**. Rio de Janeiro: RNP/ESR, 2013.

W3TECHS. **Usage of web servers broken down by server-side programming languages**, 2017. Disponível em: <https://w3techs.com/technologies/cross/web_server/programming_language>. Acesso em: 20 out. 2017 (Gráfico 1 - Usage of web servers broken down by server-side programming languages)

WIKIPÉDIA. **Página Principal**, 2017. Disponível em: <https://pt.wikipedia.org/wiki/Wikipédia:Página_principal>. Acesso em: 20 out. 2017 (Imagem 2 - Página Principal)