

CENTRO UNIVERSITÁRIO SENAC

Flávio Ferreira da Cunha

Hardening para aplicações web baseadas em LAMP (Linux, Apache, Mysql e PHP)

São Paulo

2020

Flávio Ferreira da Cunha

Hardening para aplicações web baseadas em LAMP (Linux, Apache, Mysql e PHP)

Trabalho de Conclusão de Curso apresentado ao Centro Universitário Senac – Campus Santo Amaro, como exigência parcial para obtenção do grau de Especialista em Segurança da Informação. Este trabalho foi desenvolvido nas disciplinas Trabalho de Conclusão de Curso I, II e III e mediado pela professora Nathalia Sautchuk Patrício.

São Paulo

2020

Elaborada pelo sistema de geração automática de ficha catalográfica do Centro
Universitário Senac São Paulo com dados fornecidos pelo autor(a).

Ferreira da Cunha, Flávio

Hardening para aplicações web baseadas em LAMP Linux, Apache,
Mysql e PHP / Flávio Ferreira da Cunha - São Paulo (SP), 2020.
101 f.: il. color.

Mediador(a): Nathalia Sautchuk Patrício

Trabalho de Conclusão de Curso (Pós-Graduação em Segurança da
Informação) - Centro Universitário Senac, São Paulo, 2020.

1. Hardening. 2. Linux. 3. PHP. 4. Segurança em aplicações web. 5.
LAMP. I. Sautchuk Patrício, Nathalia (Mediad.) II. Título

Flávio Ferreira da Cunha

Hardening para aplicações web baseadas em LAMP (Linux, Apache, Mysql e PHP)

Trabalho de Conclusão de Curso
apresentado ao Centro Universitário
Senac – Campus Santo Amaro, como
exigência parcial para obtenção do grau
de Especialista em Segurança da
Informação.

Mediadora: Profª Nathalia Sautchuk
Patrício

A banca examinadora dos Trabalhos de Conclusão, em sessão pública realizada em
____/____/____, considerou o(a) candidato(a):

1) Examinador(a)

2) Examinador(a)

3) Presidente

DEDICATÓRIA

Dedico este trabalho as comunidades *Hacker*, *Linux* e *Open Source* pelo trabalho de divulgação do conhecimento e por ajudarem a construir a Internet que conhecemos hoje.

AGRADECIMENTOS

Agradeço à minha esposa Marcilene pela companhia e apoio incondicional durante o desenvolvimento deste trabalho.

Agradeço ao meu pai Fábio F Cunha por ter me ensinado tudo o que eu precisava aprender sobre a vida.

Agradeço ao meu amigo Felipe Siqueira pelo apoio e parceria.

Agradeço a todas as empresas que confiam no meu trabalho pela oportunidade de crescimento moral e profissional.

Agradeço também aos professores do Senac pelo aprendizado que este curso proporcionou e pelas lições valiosas que foram compartilhadas.

“ Não se deve honrar um homem acima da verdade. ” (Platão)

RESUMO

O objetivo desta pesquisa foi demonstrar como as falhas de segurança em aplicações *web* baseadas em LAMP (Linux, Apache, Mysql e PHP) podem comprometer o funcionamento esperado destas aplicações e de toda a sua infraestrutura. As análises e pesquisas bibliográficas realizadas demonstraram que dentre os principais fatores que contribuem para a insegurança e indisponibilidade destas aplicações estão os códigos mal escritos, configurações padrão, falta de atualizações e o uso de componentes vulneráveis. Ainda, se todos estes fatores estiverem associados com a falta de sistemas de gestão de segurança da informação dentro do ambiente corporativo, a situação fica ainda mais crítica. A segurança da informação exige um trabalho contínuo, desta forma, as práticas utilizadas para aumentar e fortalecer a segurança de um sistema, comumente conhecidas como *Hardening*, são de fundamental importância para diminuir as vulnerabilidades e manter a disponibilidade dentro do ambiente das aplicações *web*. Uma empresa pode ter sua reputação comprometida se uma de suas aplicações disponibilizadas na Internet for vítima de um incidente de segurança, pois, falhas podem ocasionar modificação, roubo e perda de dados, lentidão e indisponibilidade total ou parcial da aplicação. Um documento importante que serve como referência para manter a segurança das aplicações *web* é o OWASP *Top Ten*, disponibilizado pela fundação OWASP (*Open Web Application Security Project*). Muitas vezes, os *softwares* desenvolvidos são colocados em ambiente de produção sem o conjunto de testes necessários e as tratativas e validações de código não são realizadas. Por outro lado, uma quantidade enorme de informações sobre segurança para estas aplicações são disponibilizadas na Internet e fundações como a OWASP contribuem significativamente para melhorar a segurança e a qualidade do *software* neste ambiente. E por fim, os resultados apresentados com as técnicas de *Hardening* recomendadas pelas referências pesquisadas foram satisfatórios para aumentar de forma significativa a segurança das aplicações LAMP.

Palavras-chave: 1. *Hardening*. 2. Linux. 3. PHP. 4. Segurança em aplicações *web*. 5. LAMP.

ABSTRACT

The objective of this research was to demonstrate how the security flaws in LAMP-based web applications (Linux, Apache, Mysql and PHP) can compromise the expected functioning of these applications and their entire infrastructure. Analyzes and bibliographic research conducted have shown that among the main factors that contribute to the insecurity and unavailability of these applications are poorly written codes, default settings, lack of updates and the use of vulnerable components. Still, if all these factors are associated with the lack of information security management systems within the corporate environment, the situation is even more critical. Information security requires continuous work, thus, the practices used to increase and strengthen the security of a system, commonly known as Hardening, are of fundamental importance to reduce vulnerabilities and maintain availability within the environment of web applications. A company can have its reputation compromised if one of its applications made available on the Internet is the victim of a security incident, as failures can cause modification, theft and loss of data, slowness and total or partial unavailability of the application. An important document that serves as a reference for maintaining the security of web applications is the OWASP Top Ten, made available by the OWASP foundation (Open Web Application Security Project). Often, the software developed is placed in a production environment without the necessary set of tests and code dealings and validations are not carried out. On the other hand, a huge amount of security information for these applications is made available on the Internet and foundations such as OWASP contribute significantly to improving the security and quality of the software in this environment. And finally, the results presented with the Hardening techniques recommended by the researched references were satisfactory to significantly increase the security of LAMP applications.

Keywords: 1. Hardening. 2. Linux. 3. PHP. 4. Web application security. 5. LAMP.

LISTA DE ILUSTRAÇÕES

Figura 1 - Participação de mercado de servidores web na Internet segundo a Netcraft.....	15
Figura 2 - Modelo de referência OSI, TCP/IP e protocolos da pilha TCP/IP.....	22
Figura 3 - Instalação do pacote pwgen.....	35
Figura 4 - Geração de senhas com o comando pwgen.....	35
Figura 5 - Instalação do SSH.....	36
Figura 6 - Atualizando as configurações do servidor SSH.....	37
Figura 7 - Arquitetura de firewall <i>screened subnet</i>	39
Figura 8 - Tipos de firewalls e atuação na pilha TCP/IP.....	42
Figura 9 - Fluxo de pacotes nas tabelas do netfilter/iptables.....	46
Figura 10 - Política padrão para o firewall de host.....	47
Figura 11 - Acesso total para 127.0.0.1 (loopback).....	47
Figura 12 - Permitindo a entrada de pacotes ICMP <i>echo request</i> (ping).....	47
Figura 13 - Permitindo acesso ao servidor SSH na porta TCP 35777.....	48
Figura 14 - Permitindo acesso ao servidor web (HTTP e HTTPS).....	48
Figura 15 - Acesso ao servidor Mysql somente para os ips 192.168.0.180 e 192.168.0.181.....	48
Figura 16 - Garantindo respostas para as solicitações do próprio firewall.....	49
Figura 17 - <i>Script Shell</i> para <i>firewall</i> de <i>host</i>	49
Figura 18 - Atribuindo permissão de execução para o <i>script</i> de <i>firewall</i>	52
Figura 19 - Habilitando o script para carregar na inicialização do sistema.....	52
Figura 20 - Instalação do Apache com suporte a PHP.....	52
Figura 21 - Informações sobre a versão do Apache e da distribuição Linux.....	53
Figura 22 - Ativando as novas configurações do Apache.....	53
Figura 23 - Ocultando informações sobre a versão do Apache e da distribuição Linux.....	54
Figura 24 - Nenhuma informação sobre versões de serviço é exibida.....	54
Figura 25 - Controlando o acesso na pasta projetos.....	55
Figura 26 - Utilização do comando htpasswd para criação do arquivo de usuários.....	56
Figura 27 - Criação do segundo usuário.....	56
Figura 28 - Visualização do conteúdo do arquivo de usuários.....	56
Figura 29 - Atualizando as configurações do Apache.....	57
Figura 30 - Usuário realizando autenticação para acessar pasta do Apache.....	57
Figura 31 - Permitindo acesso na pasta projetos somente para o ip 192.168.0.26.....	58
Figura 32 - Acesso proibido para endereço ip não autorizado.....	58
Figura 33 - Arquivo de <i>logs</i> do Apache mostra o bloqueio do ip indesejável.....	59
Figura 34 - Instalação do pacote openssl.....	60
Figura 35 - Ativação do módulo <i>mod_ssl</i> para o Apache.....	60
Figura 36 - Geração do arquivo de certificado e chave privada.....	61
Figura 37 - Edição do arquivo <i>default-ssl.conf</i>	61
Figura 38 - Indicação do caminho do arquivo de certificado e da chave privada.....	61
Figura 39 - Reinicialização do Apache.....	61
Figura 40 - Acesso em servidor <i>web</i> Apache com certificado autoassinado.....	62
Figura 41 - Arquivo <i>insecure_code.php</i>	63
Figura 42 - Arquivo <i>/etc/passwd</i> sendo visualizado de forma arbitrária.....	64

Figura 43 - Codificação segura do arquivo <code>insecure_code.php</code>	64
Figura 44 - Vulnerabilidade de <i>Directory Traversal</i> corrigida.....	65
Figura 45 - Arquivo <code>monitora.php</code>	65
Figura 46 - Programa <code>monitora.php</code> sendo executado da maneira esperada.....	66
Figura 47 - Programa <code>monitora.php</code> sendo usado para executar comandos do sistema.....	67
Figura 48 - Codificação segura do arquivo <code>monitora.php</code>	68
Figura 49 - Programa <code>monitora.php</code> sem executar comandos do sistema.....	68
Figura 50 - Desativação de funções PHP que podem executar comandos de sistema.....	69
Figura 51 - Reinicialização do Apache.....	69
Figura 52 - <i>Download</i> do <i>Mysql Community</i>	70
Figura 53 - Instalando os pacotes do <i>Mysql Community</i>	70
Figura 54 - Seleção dos pacotes do <i>Mysql Community</i>	71
Figura 55 - Instalação do <i>Mysql</i> e do <i>PHP</i>	71
Figura 56 - Escolha da senha de <i>root</i> do <i>Mysql</i>	71
Figura 57 - Escolha do método de autenticação para o <i>Mysql</i>	72
Figura 58 - Verificação do status do servidor <i>Mysql</i>	72
Figura 59 - verificação do estado da porta do servidor <i>Mysql</i>	73
Figura 60 - Acesso ao banco como usuário <i>root</i>	73
Figura 61 - Comandos <i>Mysql</i> necessários para simulação de ataque <i>SQL Injection</i>	73
Figura 62 - Arquivo <code>conecta.php</code>	75
Figura 63 - Arquivo <code>pesquisa.php</code>	75
Figura 64 - Programa <code>pesquisa.php</code> sendo executado da maneira esperada.....	76
Figura 65 - Aplicação <code>pesquisa.php</code> sofrendo um ataque de <i>SQL injection</i>	77
Figura 66 - <i>Query</i> (consulta) <i>SQL</i> que mostra informações sobre o usuário.....	77
Figura 67 - <i>String</i> maliciosa construída pela invasor.....	77
Figura 68 - Trecho de código vulnerável.....	78
Figura 69 - A função <code>htmlspecialchars</code> sendo utilizada para proteger contra o <i>SQL injection</i>	78
Figura 70 - Execução do comando <code>mysql_secure_installation</code>	79
Figura 71 - <code>mysql_secure_installation</code> desativando o <i>login</i> remoto para o usuário <i>root</i>	80
Figura 72 - Estrutura para demonstração de esquema de permissões.....	81
Figura 73 - Criação dos usuários e definição das permissões de acesso com o comando <code>GRANT</code>	82
Figura 74 - Usuário <i>Boris</i> com acesso negado para <code>UPDATE</code> na tabela <i>produtos</i>	82
Figura 75 - Senhas dos usuários armazenadas de forma insegura.....	83
Figura 76 - Senhas armazenadas de forma segura com uso do algoritmo de <i>Hash Bcrypt</i>	84
Figura 77 - <i>Backup</i> de uma base de dados com <code>mysqldump</code>	85
Figura 78 - Restauração de uma base de dados com o comando <code>mysql</code>	85
Figura 79 - Código <code>HTML</code> com formulário usando o método <code>HTTP GET</code>	87
Figura 80 - Arquivo <code>teste_risco.php</code>	87
Figura 81 - Entrada de dados no formulário.....	88
Figura 82 - Dados processados e exibidos pelo arquivo <code>teste_risco.php</code>	88
Figura 83 - Injeção de código <code>JavaScript</code> malicioso no formulário.....	89
Figura 84 - Sucesso na execução do ataque <i>XSS</i>	89
Figura 85 - Arquivo <code>teste_risco.php</code> com função <code>htmlspecialchars</code>	90

Figura 86 - Vulnerabilidades <i>web</i> mais graves de acordo com relatório Acunetix 2019.....	94
Figura 87 - Falhas de segurança XSS em aplicações <i>web</i> de acordo com relatório Acunetix 2019.....	95

LISTA DE TABELAS

Tabela 1 - Classificação das aplicações <i>web</i>	19
Tabela 2 - Atributos de segurança da informação.....	23
Tabela 3 - OWASP Top 10 - 2017.....	24
Tabela 4 - Particionamento de disco para um servidor Linux com LAMP.....	32
Tabela 5 - Fortalecimento da segurança no arquivo de configuração do SSH.....	36
Tabela 6 - Atuação do <i>firewall packet filtering</i> com relação a pilha TCP/IP.....	40
Tabela 7 - Tabela de estado de um <i>firewall stateful</i>	41
Tabela 8 - Política padrão de <i>firewall de host</i> com iptables.....	44
Tabela 9 - Autores e correções propostas para ataques XSS.....	96

SUMÁRIO

1 INTRODUÇÃO	14
1.1 Tema	14
1.2 Justificativa	15
1.3 Objetivos	16
1.3.1 Objetivo geral	16
1.3.2 Objetivos específicos	16
1.4 Problema de pesquisa	16
1.5 Delimitação do problema de pesquisa	17
2 REFERENCIAL TEÓRICO	18
2.1 Aplicações <i>web</i> e seus componentes	18
2.1.2 Sistema Operacional	19
2.1.3 Servidor <i>web</i> ou HTTP	20
2.1.4 Aplicação <i>web</i>	20
2.1.5 Banco de dados	20
2.2 Arquitetura e características das aplicações <i>web</i>	20
2.3 Introdução à segurança de aplicações <i>web</i>	22
2.4 Pilha LAMP	28
2.5 <i>Hardening</i> para sistemas Linux	31
2.5.1 Particionamento de disco	32
2.5.2 Política de senhas	33
2.5.3 <i>Hardening</i> no SSH	36
2.5.4 <i>Firewalls</i> em ambientes Linux	37
2.6 <i>Hardening</i> para o servidor <i>web</i> Apache	52
2.6.1 Ocultando Informações sobre o servidor	53
2.6.2 Autenticação, autorização e controle de acesso	55
2.6.3 Acesso seguro com SSL/TLS	59
2.7 <i>Hardening</i> para a aplicação PHP	63
2.7.1 <i>Directory Traversal</i>	63
2.7.2 <i>Code Injection</i>	65
2.7.3 <i>SQL injection</i>	69
2.7.4 <i>Hardening</i> no Mysql	79
2.7.5 <i>Cross-site scripting</i> (XSS)	86
3 METODOLOGIA	91
3.1 Método	91
3.2 Tipo de pesquisa	91
3.3 Coleta de dados	92
3.4 Tratamento de dados	92
4 ANÁLISE DOS DADOS	93
4.1 Tema – <i>Cross-site Scripting</i> (XSS)	93
4.2 Considerações sobre a análise de dados	96
5 CONSIDERAÇÕES FINAIS	97
6 REFERÊNCIAS	98

1 INTRODUÇÃO

Nos últimos anos a disseminação na utilização de dispositivos como *firewalls* e sistemas de detecção de intrusão melhorou significativamente a segurança de redes e de servidores, entretanto, apesar destes equipamentos realizarem uma proteção razoável na camada de rede, pouco fazem para manter a segurança nas aplicações *web*. Isto posto, os *hackers* passaram a atacar a mesma, uma vez que estas aplicações interagem diretamente com todos os sistemas internos de uma empresa, como o servidor de banco de dados, por exemplo (PAULI, 2014).

Muitas vezes a equipe responsável pelo desenvolvimento de uma aplicação *web* se preocupa mais com as funcionalidades da aplicação do que com os aspectos relacionados a segurança. Negligenciar a segurança pode expor a aplicação *web* a diversos riscos. Vulnerabilidades podem ser exploradas para deixar a aplicação indisponível, obter informações confidenciais como usuários e senhas, modificar as páginas, executar comandos arbitrários, entre outras ameaças.

Diante do cenário apresentado os métodos de *hardening* que visam o aumento da segurança de um sistema são importantes para que as aplicações *web* e todos os seus componentes estejam de acordo com as normas e padrões de segurança da informação reconhecidos e recomendados pela indústria, funcionem com um bom desempenho, mantenham a integridade, autenticidade e a confidencialidade de seus dados e fiquem disponíveis durante todo o tempo que seus usuários precisarem dela.

1.1 Tema

Estratégias e ferramentas de *hardening* para fortalecer a segurança da informação na implementação e no gerenciamento de aplicações *web* baseadas em LAMP (Linux, Apache, Mysql e PHP).

1.2 Justificativa

O propósito deste trabalho é demonstrar como fortalecer a segurança de aplicações *web* baseadas em LAMP (Linux, Apache, Mysql e PHP) usando técnicas de *hardening* desde a sua concepção inicial até o seu gerenciamento depois de colocada em ambiente de produção. Segundo Pauli (2014, p.27), "A segurança deve permanecer como uma força direcionadora do projeto durante todas as fases de *design*, construção, implementação e testes".

Devido a grande popularidade das implementações LAMP na Internet e nas intranets de empresas e instituições ao redor do mundo é importante que se faça um estudo e que se desenvolvam métodos e padrões para que as mesmas possam ser disponibilizadas e utilizadas com mais segurança.

De acordo com pesquisa da Netcraft, de abril de 2019, o Apache, juntamente com o Nginx e o IIS da Microsoft são os servidores *web* com mais participação na Internet atualmente. Como o Apache é nativo da plataforma Linux é muito mais comum que seja executado neste sistema operacional em conjunto com aplicações como o PHP e o banco de dados Mysql ou Mariadb.

Figura 1 - Participação de mercado de servidores web na Internet segundo a Netcraft.



Fonte: Netcraft (2019).

1.3 Objetivos

1.3.1 Objetivo geral

Demonstrar como as práticas de *hardening* podem ajudar no fortalecimento da segurança de aplicações web baseadas em LAMP (Linux, Apache, Mysql e PHP).

1.3.2 Objetivos específicos

Aumentar a segurança de todos os componentes que compoem uma aplicação *web* baseada em LAMP, são eles: sistema operacional Linux, servidor *web* Apache, linguagem de programação PHP e banco de dados Mysql.

Reforçar a segurança de componentes específicos da aplicação *web* e do processo de interação entre eles, como: campos para entrada de dados de usuário, processos de autenticação, sessão, comunicação entre aplicação e banco, funções *Hash*, protocolos HTTP (*Hypertext Transfer Protocol*) e HTTPS (*Secure Hypertext Transfer Protocol*), entre outros.

Demonstrar as principais vulnerabilidades associadas as aplicações *web*, bem como, as respectivas contramedidas.

1.4 Problema de pesquisa

"[...]Uma aplicação *web* está suscetível a ataques ou a acessos dos mais diferentes tipos de usuários" (MILETTO; BERTAGNOLLI, 2014, p.10).

Assim sendo, de que maneira processos e estratégias de *hardening* aplicadas ao uso, desenvolvimento e gerenciamento de aplicações *web* baseadas em LAMP podem contribuir para aumentar a autenticidade, confidencialidade, integridade e a disponibilidade destas aplicações?

1.5 Delimitação do problema de pesquisa

O primeiro capítulo, Introdução, apresenta uma introdução as aplicações *web* baseadas em LAMP. Como, as mesmas, estão inseridas dentro do contexto da Internet e um pouco sobre a importância da segurança da informação para estas aplicações.

O segundo capítulo aborda o referencial teórico. Discorre sobre a arquitetura e as características das aplicações *web* e seus componentes e demonstra técnicas para fortalecimento da segurança (*hardening*) que podem ser usadas em todos os componentes da pilha LAMP (Linux, Apache, Mysql e PHP).

O terceiro capítulo mostra a metodologia utilizada para desenvolvimento deste trabalho, que por sua vez, é dividida em quatro partes: método, tipo de pesquisa, coleta e tratamento de dados.

O quarto capítulo, análise dos dados, analisa os dados sobre o tema *Cross-site Scripting* (XSS), mostrando dados sobre a sua popularidade na Internet, bem como, as respectivas contramedidas para proteger as aplicações *web* LAMP desta ameaça.

Por fim, o quinto capítulo faz as considerações finais e reflete sobre possíveis trabalhos futuros que podem ser desenvolvidos.

2 REFERENCIAL TEÓRICO

2.1 Aplicações *web* e seus componentes

Segundo Pauli (2014), muitas pessoas em diferentes contextos se referem as aplicações *web* com outras nomenclaturas como: sistema baseado em *web*, *web site*, *software* baseado em *web* e *web*. Todos estes nomes podem ter o mesmo significado, entretanto, devido a generalização do termo aplicação *web* fica complicado diferenciar as aplicações que realizam alguma ação e são interativas dos *sites* que apenas exibem conteúdo HTML (*Hypertext Markup Language*) estático e não interativo.

Portanto, "Quando um usuário interage com um *web site* para realizar alguma ação, por exemplo, fazer *login*, fazer compras ou acessar o banco, temos uma aplicação *web*" (PAULI, 2014, p.26).

Outras abordagens também podem ser utilizadas para definir as aplicações *web*. Conforme Miletto e Bertagnolli (2014), as aplicações *web* podem ser classificadas de acordo com a sua orientação. Na tabela 1, alguns exemplos:

Tabela 1 - Classificação das aplicações *web*.

Síte de conteúdo	Exibe apenas conteúdo HTML estático e não interativo.
Síte de registro de dados ou entrada do usuário	Apresenta apenas formulários que devem ser preenchidos pelos usuários de acordo com os objetivos do <i>síte</i> .
Portal	É constituído por várias páginas e <i>links</i> que convergem para um mesmo contexto. Por exemplo: um portal somente de notícias.
Aplicação orientada a transação	Neste sistema o usuário envia solicitações para a aplicação processar em um banco de dados. Depois de processada a resposta desta solicitação é enviada ao usuário.

Fonte: Mileto e Bertagnolli (2014). Adaptado pelo autor.

De forma geral, a aplicação *web* é composta por quatro componentes principais, são eles: sistema operacional, servidor *web*, aplicação *web* e banco de dados.

2.1.2 Sistema Operacional

É a base para implementação de uma aplicação *web* e nele serão instalados os programas do servidor *web*, da aplicação e do banco de dados.

De acordo com Tanenbaum (2010), um sistema computacional moderno é constituído por vários componentes como processadores, memória, interfaces de rede, discos e outros dispositivos de entrada e saída. O sistema operacional é o *software* responsável por gerenciar todos estes elementos e também por fornecer uma interface para que os programas de usuário possam ser escritos e executados.

2.1.3 Servidor *web* ou HTTP

O protocolo da camada de aplicação da *web* é o HTTP. Ele é dividido em duas partes: o lado cliente e o lado servidor. Estes lados executam em sistemas finais diferentes e se comunicam pela troca de mensagens HTTP. Os clientes *web* (*browsers*) fazem requisições para os servidores *web* solicitando objetos como páginas e arquivos de imagem. O servidor *web*, por sua vez, realiza a transferência destes objetos (KUROSE; ROSS, 2003).

2.1.4 Aplicação *web*

A aplicação *web* é uma linguagem de programação que pode ser executada tanto no cliente quanto no servidor *web*.

“[...] É caracterizada por construir dinamicamente o seu conteúdo, com dados provenientes de um banco de dados, a partir da interação dos usuários com as páginas, via navegadores” (MILETTO; BERTAGNOLLI, 2014, p.18). Este trabalho se baseia na aplicação *web* que fica no servidor.

2.1.5 Banco de dados

O banco de dados, por sua vez, é um local para armazenar dados. O banco de dados armazena os dados que serão utilizados pela aplicação *web*.

2.2 Arquitetura e características das aplicações *web*

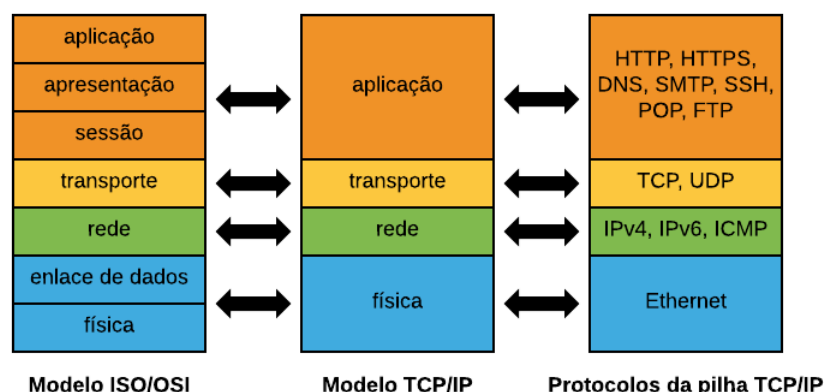
Convém ressaltar que além dos componentes citados uma aplicação *web* normalmente utiliza outras tecnologias para aumentar o seu nível de funcionalidades. Podemos destacar: *frameworks*, *softwares* CMS (*Content Management System*), Ferramentas estatísticas, API (*Application Programming Interface*), *plugins*, linguagens de programação para interfaces de usuário (*frontend*), entre outras tecnologias.

Como menciona Stevens, Fenner e Rudoff (2005), aplicações *web* são aplicações de rede que utilizam um modelo conhecido como cliente/servidor. Neste modelo o cliente envia requisições para o servidor e o servidor sempre fica aguardando por solicitações do cliente. Dentro deste cenário, o servidor *web* atuaria como servidor e o navegador *web* como cliente. Para que programas possam se comunicar através de uma rede de computadores é necessário que utilizem as mesmas regras de comunicação. Estas regras são conhecidas como protocolos. Muitos protocolos de rede são utilizados e o conjunto de todos eles são denominados como: protocolos da pilha TCP/IP (*Transmission Control Protocol/Internet Protocol*). O servidor e o cliente *web* comunicam-se utilizando TCP, o TCP, por sua vez, utiliza o protocolo IP e o IP utiliza protocolos da camada física para transmitir os dados através do meio físico.

O modelo utilizado como referência para descrição e estudo dos protocolos de comunicação em rede de computadores é conhecido como OSI (*Open Systems Interconnection*) da organização ISO (*International Organization for standardization*), contudo, na prática o modelo utilizado é o TCP/IP.

Os protocolos da pilha TCP/IP são divididos em 4 camadas: **aplicação**, **transporte**, **rede** e **física** e possuem vários protocolos que atuam em cada uma delas, conforme mostra a figura 2.

Figura 2 - Modelo de referência OSI, TCP/IP e protocolos da pilha TCP/IP.



Fonte: Stevens, Fenner e Rudoff (2005). Adaptado pelo autor.

Nos dias atuais, a maioria das aplicações *web* são desenvolvidas para serem responsivas, ou seja, se adequarem tanto para exibição em máquinas *desktops* quanto para dispositivos móveis, interativas e repletas de funcionalidades. *Sites* de conteúdo que exibem apenas páginas HTML estáticas e não interativas quase não existem, portanto, a grande maioria dos métodos utilizados para fortalecimento da segurança da informação em aplicações *web* compreendem as aplicações orientadas a transação onde comunicações são feitas com um banco de dados por meio de entradas de usuário.

2.3 Introdução à segurança de aplicações *web*

Quando a *World Wide Web* (WWW) ou simplesmente *web* foi inventada por Tim Berners-Lee no período de 1989 a 1991, ele e sua equipe desenvolveram versões bem simplificadas da linguagem HTML, do protocolo HTTP, um servidor *web* e um navegador, os quatro componentes fundamentais da *web* (KUROSE; ROSS, 2003). Era o início da Internet como a conhecemos hoje e naquela época não existia nenhuma preocupação com a segurança das ferramentas que a compoem. Nos dias atuais a *web* e seus componentes evoluíram e embora existam muitas medidas e tecnologias para aumentar a segurança das aplicações *web*, as

mesmas, continuam apresentando muitas falhas que podem ser exploradas para os mais diferentes propósitos.

Conforme Miletto e Bertagnolli (2014), a fim de evitar comprometimentos de segurança em aplicações *web*, é fundamental que se conheçam primeiramente, os atributos de segurança que devem ser garantidos. São eles:

Tabela 2 - Atributos de segurança da informação.

Autenticidade	Se refere ao controle de acesso baseado em mecanismos de autenticação. Este atributo é essencial para garantir a identidade do usuário.
Confidencialidade	Garantir que somente as pessoas autorizadas possam ter acesso aos dados. A proteção dos dados deve ser garantida tanto quando os mesmos estiverem trafegando pela rede quanto armazenados em bancos de dados, servidores de arquivos, <i>backup</i> , entre outros.
Integridade	Garantia de que os dados serão alterados somente pelas pessoas autorizadas. Tanto os dados em trânsito pela rede quanto armazenados devem possuir controles com relação a modificações indevidas. Qualquer alteração deve ser identificada.
Disponibilidade	É a garantia de que os dados e os sistemas estarão disponíveis sempre que solicitados. Mecanismos de contingência como <i>clusters</i> de servidores, podem ser utilizados para garantir a disponibilidade de um sistema. Caso um deles falhe, um outro servidor assume as suas funções.

Fonte: Miletto e Bertagnolli (2014). Adaptado pelo autor.

Várias comunidades de segurança se dedicam a fortalecer a segurança das aplicações *web*, dentre elas, podemos destacar a Wasc (*Web Application Security Consortium*) e a Owasp (*Open Web Application Security Project*).

De acordo com a Owasp (2017), a infraestrutura de serviços críticos como: defesa, saúde, financeira, energia e outras são comprometidas por *softwares* inseguros. As dificuldades para manter a segurança do *software* aumentam a medida que o mesmo se torna mais intrincado, complexo e interligado. Os processos atuais de desenvolvimento de *software* que visam colocar o mesmo em produção o mais rápido possível também ajudam a dificultar a detecção destes riscos de forma rápida e precisa.

Uma variedade enorme de *softwares* e técnicas foram desenvolvidas para explorar as vulnerabilidades das aplicações *web* e o uso de metodologias e práticas seguras recomendadas por normas de segurança e organizações como a Owasp podem contribuir de forma considerável para o aumento da segurança nestas aplicações, uma vez que, identificam os principais riscos e ameaças que as afetam.

Entre as recomendações da Owasp destaca-se o documento “**OWASP Top 10 - Ano - “The Ten Most Critical Web Application Security Risks”**”. Para elaboração desta pesquisa foi utilizado o OWASP Top 10 - 2017. Este documento apresenta os dez riscos de segurança mais críticos para as aplicações *web* e se tornou uma referência padrão mundial para profissionais de segurança e desenvolvedores de aplicações *web*. Segundo a Owasp (2017), as principais ameaças são as seguintes:

Tabela 3 - OWASP Top 10 - 2017.

Injeção	Falhas de injeção como SQL, NoSQL, OS e LDAP <i>injection</i> podem ocorrer quando dados maliciosos ou não confiáveis são enviados para um interpretador como parte de um comando ou <i>query</i> . Se forem interpretados como legítimos estes dados poderão executar comandos arbitrários ou mesmo acessar dados sem a devida autorização.
Quebra de autenticação	Aplicativos relacionados a autenticação e gerenciamento de sessão implementados de forma incorreta podem comprometer senhas, chaves, <i>tokens</i> de sessão ou permitir que o invasor assuma a identidade de outros usuários de forma temporária ou permanente.

Exposição de dados sensíveis	Muitas aplicações <i>web</i> e APIs não protegem de forma adequada dados confidenciais sensíveis relacionados a serviços críticos como financeiro, saúde e dados pessoais. Quando não possuem uma proteção adicional adequada, como o uso de criptografia quando armazenados ou em trânsito, eles podem ser roubados ou modificados a fim de serem usados em crimes que envolvam fraudes com cartões de crédito, roubo de identidade, entre outros.
Entidades externas de XML (XXE)	Processadores de XML mais antigos ou mal configurados podem fazer referências a entidades externas dentro dos documentos XML. Estas entidades externas, por sua vez, podem ser utilizadas para revelar arquivos internos usando o processador de URI de arquivos, executar código remoto e ataques de negação de serviço.
Quebra de controle de acessos	Ocorre quando usuários autenticados acessam mais recursos ou possuem mais privilégios do que deveriam. Invasores podem tirar proveito destes controles de acesso mal configurados para acessar contas de outros usuários, obter funcionalidades ou dados não autorizados, modificar permissões de acesso, entre outras violações.
Configurações de segurança incorretas	Problemas relacionados com configurações de segurança incorretas são comuns e normalmente são resultado de configurações padrão inseguras, incompletas ou <i>ad hoc</i> . Mensagens de erro detalhadas contendo informações sensíveis, armazenamento em nuvem sem autenticação e pontos de acesso <i>wi-fi</i> abertos são alguns exemplos.
Cross site scripting (XSS)	Quando a aplicação inclui dados não confiáveis de entradas de usuário sem realizar a validação adequada uma falha de XSS pode ocorrer. Códigos <i>JavaScript</i> normalmente são utilizados neste tipo de ataque. O XSS permite que o invasor execute <i>scripts</i> no navegador da vítima, sequestre sua sessão, desfigure o <i>web site</i> ou redirecione o usuário para um site malicioso.

Desserialização insegura	Falhas de desserialização insegura normalmente permitem execução remota de código, entretanto, podem ser exploradas para realização de outros tipos de ataques como: <i>replay attacks</i> , injeção e escaladas de privilégios.
Utilização de componentes vulneráveis	Muitos componentes como bibliotecas, <i>frameworks</i> e módulos de <i>software</i> executam com os mesmos privilégios da aplicação, portanto, aplicações e APIs que usam elementos com vulnerabilidades conhecidas podem passar por diversas violações de segurança.
Registro e monitoração insuficiente	A ausência de registros e monitoração dos eventos juntamente com uma boa integração com os mecanismos de detecção e respostas de incidentes, permitem que os invasores ataquem mais os sistemas, modifiquem, obtenham ou destruam dados. Muitos estudos mostram que as violações normalmente são detectadas por agentes externos ao invés de processos de monitoramento interno.

Fonte: Owasp (2017).

As ameaças mais predominantes podem servir como ponto de partida para analistas de segurança e desenvolvedores *web* definirem estratégias de defesa para suas aplicações.

Segundo Basso (2010), quando se trata de segurança em aplicações *web* muitos pontos de fragilidades são encontrados. Classificar estas fragilidades em categorias ajuda a entender melhor o cenário a fim de aplicar as contramedidas mais adequadas.

De acordo com Tsipenyuk (apud Basso, 2010), as categorias que retratam os pontos de fragilidade comumente identificados, são:

Validação e representação de entrada de dados: Problemas de segurança podem ser ocasionados pela ausência no tratamento de validação e representação dos dados de entrada contra inserção de conteúdos maliciosos, como por exemplo: metacaracteres, representações numéricas e codificações inusitadas. Dentro destas circunstâncias ataques de XSS e injeção de SQL podem ser executados.

Exploração de API: Uma API permite a integração entre diferentes aplicações. Elas estabelecem um acordo entre um método solicitante e um provedor. Uma forma comum de exploração de APIs ocorre quando o solicitante não cumpre a sua parte neste acordo.

Recursos de segurança: Diferentes recursos de segurança podem ser implementados em um mesmo sistema. Controle de acesso, criptografia, autenticação e gerenciamento de privilégios são alguns exemplos. Todas estas funcionalidades de segurança tornam o sistema mais complexo aumentando as dificuldades de gerenciamento e dentro deste contexto vulnerabilidades podem surgir.

Comunicação: A comunicação na computação distribuída compartilha tempo e estado. Problemas entre as interações de processos, tempos e *threads* podem ocasionar falhas de autenticação e gerenciamento de sessão.

Erros: O tratamento inadequado dos erros pode expor as fragilidades do sistema.

Qualidade do código: *Softwares* de má qualidade apresentam comportamento imprevisível e fornecem oportunidades para ataques e explorações do sistema.

Encapsulamento: Significa implementar proteção em torno de recursos. Por exemplo, a partir de um navegador não podemos acessar o disco rígido de um servidor *web* de forma arbitrária. Uma autenticação deve ser necessária para acessar recursos protegidos.

Ambiente: É responsável por armazenar o *software* e fazer sua conexão com um ambiente externo. O Gerenciamento inapropriado deste ambiente pode causar falhas na configuração do sistema.

Identificar os riscos e as ameaças mais predominantes que afetam as aplicações *web* podem servir como ponto de partida para analistas de segurança e desenvolvedores definirem estratégias de segurança mais eficazes e materiais como o OWASP Top 10 - 2017 - “*The Ten Most Critical Web Application Security Risks*” contribuem de forma significativa para esta tarefa.

2.4 Pilha LAMP

A pilha LAMP (Linux, Apache, Mysql e PHP) é composta pelo sistema operacional Linux, o servidor web Apache, o banco de dados relacional Mysql e a linguagem de programação para ambientes *web* PHP. A seguir, um breve histórico de cada um:

LINUX: É um sistema operacional baseado no UNIX que teve origem em 1991 como um projeto pessoal de um estudante finlandês chamado Linux Torvalds. Ele foi concebido tendo como base um outro sistema “*UNIX Like*” conhecido como Minix, que era utilizado para fins educativos em universidades. Podemos ressaltar ainda que o Linux segue o padrão POSIX (*Portable Operating System Interface*), pode ser executado em várias plataformas de *hardware* diferentes e é compatível com a maior parte do *software* UNIX existente. É gratuito, possui código fonte aberto e conta com o apoio de centenas de indivíduos e organizações (comunidade Linux)

dentro de um modelo de desenvolvimento colaborativo (NEMETH; SNYDER; HEIN, 2004).

APACHE: “O Apache *HTTP Server Project* é um esforço colaborativo de desenvolvimento de *software* destinado a criar uma implementação de código fonte robusta, comercial, com recursos e disponível gratuitamente de um servidor HTTP (*Web*)”. (APACHE SOFTWARE FOUNDATION, 2019). Atualmente o Apache é mantido por uma organização beneficente dos EUA conhecida como ASF. Sua diretoria totalmente voluntária supervisiona centenas de projetos de código aberto que são disponibilizados gratuitamente sob a licença do Apache beneficiando milhões de usuários em todo o mundo. O projeto é gerenciado de forma colaborativa por um grupo de voluntários espalhados por diferentes países e que usam a internet para se comunicar. O Apache possui um bom suporte para aplicações PHP e módulos podem ser acrescentados a fim de expandir sua lista de funcionalidades.

PHP: Segundo Milani (2010), em meados de 1995 o programador Rasmus Lerdorf queria obter estatísticas sobre os acessos de seu currículo que estava disponibilizado *on-line*, e para tanto, escreveu alguns *scripts* em linguagem Perl. Com o passar do tempo ele foi aprimorando estes códigos e adicionando novas funcionalidades. Posteriormente ele desenvolveu algo em Linguagem C que foi capaz de generalizar a construção de novas aplicações para a *web* e este novo projeto recebeu o nome de PHP/FI - *Personal Home Page/Forms Interpreter*. Assim nasceu o PHP (acrônimo recursivo para PHP: *Hypertext Preprocessor*) que todos conhecem atualmente. O PHP é uma linguagem de *script open source* especialmente adequada para o ambiente *web* e que pode ser embutida dentro do HTML. O código PHP é executado no servidor *web* (*server-side*) e gera o HTML que é, por sua vez, enviado de volta ao navegador. Praticamente tudo que é feito dentro de um contexto *web* com o uso de *scripts* CGI (*Common Gateway Interface*) pode ser executado com o PHP. Recuperação de dados de formulários, criação de páginas dinâmicas, envio e recebimento de *cookies* e validação de dados de entrada, são alguns exemplos (PHP, 2019).

MYSQL: É um sistema gerenciador de banco de dados (SGBD) SQL (*Structured Query Language*) *open source* e nos dias atuais é mantido pela Oracle Corporation. Apesar de ser mantido por uma empresa e possuir versões comerciais, a Oracle disponibiliza de forma gratuita a versão *Mysql community edition*.

O SQL se tornou a linguagem padronizada mais comum para acessar bancos de dados relacionais e, portanto, é fundamental que os principais SGBDs suportem a mesma. Segundo Milleto e Bertagnolli (2014), a linguagem SQL foi padronizada pela ISO/ANSI (*International Organization for standardization/American National Standards Institute*) em 1986 e é utilizada atualmente por uma grande variedade de sistemas gerenciadores de bancos de dados.

De acordo com Mysql (2019), dentre as principais características do Mysql, podemos destacar:

- Suporte a linguagem padronizada SQL (*Structured Query Language*).
- As bases de dados do servidor Mysql são relacionais.
- O servidor Mysql é muito rápido, confiável, escalável e fácil de usar.
- Usa o modelo de conectividade cliente/servidor em cima da pilha TCP/IP.
- A comunidade Mysql e os seus contribuidores disponibilizam uma grande quantidade de *softwares* para o Mysql.
- Possui comandos para *backup* e recuperação de dados.
- Suporte a recursos de segurança como gerenciamento de contas, controle de acesso de usuários e uso de conexões criptografadas.

Aplicações *web* baseadas na pilha LAMP são muito populares na Internet e possuem uma grande comunidade de usuários que contribuem para uma ampla documentação de todos os seus recursos.

Este estudo considera as implementações de *hardening* para um sistema Linux que sirva de sustentação para hospedar uma base LAMP e a distribuição utilizada é o Debian Linux 10.

2.5 *Hardening* para sistemas Linux

Hardening significa fortalecer a segurança de um sistema. É a prática que visa aumentar a segurança de um sistema com a utilização de técnicas, customizações, metodologias e ferramentas.

De acordo com Arora et al. (2014), o termo "*hardening*" faz referência a segurança do sistema. Como qualquer outra plataforma operacional, falhas em nível de aplicação deixam o Linux vulnerável a uma grande variedade de ataques. O ambiente de código aberto do Linux permite um alto grau de customização e personalização, por outro lado, esta mesma flexibilidade pode se tornar um problema de segurança, pois qualquer pessoa pode criar um programa para o sistema e aumentar suas possibilidades de configuração. Portanto, o objetivo das práticas de *hardening* em sistemas Linux consiste em utilizar estas mesmas características para criar um sistema mais seguro e que atenda melhor às necessidades dos usuários.

Qualquer distribuição Linux deve ser ajustada pós instalação do sistema a fim de aumentar a sua segurança. As configurações de segurança devem levar em consideração os serviços que o sistema vai disponibilizar, ou seja, como este Linux vai atuar? será um servidor de arquivos? E-mail? servidor *web*?. Por exemplo: se a máquina tiver que suportar uma infraestrutura LAMP, somente os pacotes necessários para prover esta implementação devem ser instalados, não faz sentido instalar programas para um servidor de arquivos ou servidor FTP (*File transfer Protocol*). Todo programa pode apresentar falhas de segurança e frequentemente deve ser atualizado para que as mesmas sejam corrigidas, portanto, quanto maior o número de programas maior o número de vulnerabilidades que podem surgir.

Além destas preocupações iniciais, o gerenciamento e controle de acesso dos usuários, o controle de acesso para os serviços de rede, a segurança do *host* e dos programas instalados também são questões que devem ser tratadas com atenção.

2.5.1 Particionamento de disco

O esquema de criação de partições também é afetado pelo tipo de serviço que o sistema irá oferecer. Por exemplo, se o objetivo for utilizar o Linux como um servidor LAMP, pode-se criar uma partição para o diretório “*document root*” do servidor *web* Apache ***/var/www/html*** e outra para o diretório onde ficam as bases de dados do Mysql ***/var/lib/mysql***.

Conforme Fernández Sanguino Peña e Reelsen (2013), devem ser criadas ainda, partições para os diretórios ***/home***, ***/tmp*** e ***/var/tmp***, onde todos os usuários tem permissão de gravação, desta maneira, evitamos que todo o espaço da partição “*/*” seja preenchido indevidamente por algum usuário e deixe o sistema inoperante. Todos os diretórios que não forem criados como partição, ficarão dentro da partição “*/*” e serão limitados pelo seu espaço.

Uma sugestão de particionamento para uma implementação LAMP é apresentada na tabela 4.

Tabela 4 - Particionamento de disco para um servidor Linux com LAMP.

Partição	Conteúdo
<i>/</i>	Comandos (binários), bibliotecas de sistema e arquivos de configuração.
<i>/boot</i>	Local da imagem do <i>Kernel</i> e de arquivos de inicialização (boot).
<i>/usr</i>	Arquivos relacionados aos programas instalados: comandos, bibliotecas, arquivos de configuração e documentação.
<i>/var</i>	Arquivos de log de programas e de sistemas.

/var/www/html	Diretório raiz (<i>document root</i>) do servidor <i>web</i> Apache. Arquivos da aplicação <i>web</i> .
/var/lib/mysql	Arquivos da base de dados e tabelas do servidor Mysql.
/tmp	Arquivos temporários. Todos os usuários tem permissão de gravação.
/var/tmp	Arquivos temporários. Todos os usuários tem permissão de gravação.
/home	Local onde ficam os diretórios pessoais (<i>home</i>) de cada usuário.
Swap	Partição de memória virtual.

Fonte: Fernández Sanguino Peña e Reelsen (2013).

2.5.2 Política de senhas

Segundo a ABNT (2013), recomenda-se que os sistemas para gerenciamento de senhas sejam interativos e garantam senhas de qualidade. Outras garantias importantes devem incluir:

- Uso individual de identificação de usuário (ID) e senha. Necessário para que cada usuário seja responsável pelo próprio acesso.
- Estabelecimento de intervalos de tempo regulares para mudança das senhas.
- Formas seguras para transmissão e armazenamento das senhas.

De acordo com Kissel (2017), os principais fatores que influenciam na força de uma senha são o seu comprimento, o conjunto de caracteres utilizados e a sua aleatoriedade.

Comprimento: Se refere ao tamanho da senha. Uma senha que utilize os 26 caracteres (somente minúsculos) do alfabeto da língua portuguesa oferece a possibilidade de 26 combinações possíveis por caractere, logo, uma senha de 4 caracteres oferece um total de $26 \times 26 \times 26 \times 26$ combinações possíveis, ou seja, 456.976. Considerando agora uma senha de 5 caracteres teremos um total de $26 \times 26 \times 26 \times 26 \times 26$ possibilidades possíveis, portanto, 11.881.376. Conclui-se que quanto maior o número de caracteres, maior a quantidade de combinações possíveis e maior a força da senha.

Conjunto de caracteres: Diz respeito a diversidade dos caracteres utilizados, como: letras maiúsculas, letras minúsculas, números e caracteres especiais. Por exemplo: uma senha que utilize os 26 caracteres minúsculos e os 26 caracteres maiúsculos do alfabeto, possibilita 52 combinações possíveis por caractere, portanto, uma senha de 4 caracteres oferece um total de $52 \times 52 \times 52 \times 52$ combinações possíveis, que é igual a 7.311.616. E uma senha de 5 caracteres oferece 380.204.032 variações possíveis. Na sequência, vamos adicionar números de 0 a 9 (10 dígitos) para a senha de 5 caracteres, ficando com um total de 62 combinações por caractere, logo teremos 916.132.832 possibilidades possíveis. Adicionando-se mais 12 caracteres especiais na senha de 5 caracteres teremos 74 combinações por caractere, o que resulta em 2.219.006.624 variações possíveis. Quanto mais conjuntos de caracteres (letras maiúsculas, letras minúsculas, números, caracteres especiais) utilizamos, mais forte a senha se torna. Em um ataque de *brute force* contra usuários e senhas, por exemplo, quanto maior o número de variações possíveis por caractere o computador tiver que analisar, maior será o tempo para concluir a tarefa.

Aleatoriedade: Ausência de um padrão compreensível. Por exemplo: Uma palavra contida em um dicionário como *house*, segue um padrão, já a sequência pfnwxi é um conjunto aleatório de caracteres. Na realidade, sequências de caracteres que parecem aleatórias para uma pessoa, são facilmente reconhecidas por algoritmos de “quebra” de senhas. Para um ser humano é praticamente impossível elaborar um conjunto verdadeiramente aleatório de caracteres, sendo extremamente difícil até para um computador. De fato, o que um computador faz é

criar conjuntos pseudoaleatórios de senhas. Praticamente imprevisíveis, mas não totalmente aleatórias.

Geração de senhas seguras em linha de comando

O comando `pwgen` pode ser utilizado a partir do *Shell* Linux para criação de senhas seguras. Para instalação do pacote, como usuário `root`, entre com o comando:

Figura 3 - Instalação do pacote `pwgen`.

```
# apt-get install pwgen
```

Fonte: O autor.

Para gerar as senhas:

Figura 4 - Geração de senhas com o comando `pwgen`.

```
$ pwgen -s -y -B 12 4  
  
p4_9a]hUvinN  J!=,4q.tfLWi  gRJgnT@=3=ne  V)Vy<$b_hs3:
```

Fonte: O autor.

Significado dos parâmetros do comando:

- s:** Para geração de senhas aleatórias.
- y:** Inclui caracteres especiais.
- B:** Não inclui caracteres ambíguos na senha.
- 12:** Quantidade de caracteres da senha.
- 4:** Quantidade de senhas geradas.

2.5.3 Hardening no SSH

O acesso remoto aos sistemas Linux deve ser feito com a utilização do serviço SSH (*Secure Shell*). De acordo com Fernández Sanguino Peña e Reelsen (2013), o SSH possui suporte a criptografia ao contrário de outros protocolos para acesso remoto como o Telnet, onde as informações trafegam em modo “*clear text*”.

Com a utilização de SSH mesmo que o tráfego entre cliente e servidor seja interceptado o invasor não poderá entendê-los.

Para instalar o SSH em um sistema Debian Linux 10, como usuário root:

Figura 5 - Instalação do SSH.

```
# apt-get install ssh
```

Fonte: O autor.

É importante não efetuar *logon* no SSH como usuário root. Deve-se entrar no sistema como usuário comum e utilizar comandos como “su” ou “sudo” para obter privilégios de root.

Para fortalecer a segurança do SSH o arquivo de configuração **/etc/ssh/sshd_config** deve ser modificado. A tabela 5, mostra alguns parâmetros que devem ser alterados ou acrescentados no arquivo:

Tabela 5 - Fortalecimento da segurança no arquivo de configuração do SSH.

Parâmetro	Função
Port 45999	Indica em qual porta o SSH irá funcionar. A porta padrão é a 22. É interessante alterar a porta padrão para confundir os invasores e as varreduras automatizadas feitas a partir da Internet.

ListenAddress 0.0.0.0	Especifica em qual endereço ip o SSH irá funcionar se a máquina estiver conectada em várias redes. indique apenas o endereço ip pertencente a rede que você deseja permitir acesso. O valor 0.0.0.0 deixa a porta do SSH aberta (<i>estado listen</i>) em todas as interfaces de rede da máquina.
PermitRootLogin prohibit-password	Bloqueia o <i>login</i> no SSH como usuário root. É importante não permitir o <i>login</i> como root, desta maneira, mesmo que a senha de root seja descoberta o invasor não poderá acessar a máquina via SSH. Este bloqueio também impede ataques de força bruta como usuário root. O valor “no” também tem o mesmo efeito.
PermitEmptyPasswords no	Não permite a utilização de usuário sem senha.
PasswordAuthentication yes	Permite autenticação com usuário e senha. Para utilizar autenticação com chaves públicas e privadas, configure esta opção como “no”. Ao desabilitar a autenticação por usuário elimina-se a possibilidade de ataques de força bruta no servidor.
AllowUsers tux pigu	Indica que somente os usuários “tux” e “pigu” poderão <i>logar</i> no SSH.

Fonte: Fernández Sanguino Peña e Reelsen (2013).

Para que as modificações entrem em vigor o SSH deve ser recarregado:

Figura 6 - Atualizando as configurações do servidor SSH.

```
# systemctl force-reload sshd.service
```

Fonte: O autor.

2.5.4 Firewalls em ambientes Linux

“*Firewalls* são dispositivos ou programas que controlam o fluxo do tráfego de rede entre redes ou *hosts* que adotam diferentes posturas de segurança” (NIST,

2009) . O *firewall* permite o bloqueio do tráfego de rede indesejável, bem como, a liberação de fluxos de pacotes legítimos. Entre os tipos de *firewalls* mais populares estão o “*firewall* de Internet” e o “*firewall* baseado em *host*”.

Firewall de Internet

Além de atuar como um roteador encaminhando pacotes entre uma ou mais redes distintas e a Internet, também opera realizando filtragem de pacotes entre estas redes de acordo com regras previamente definidas. Outro recurso importante consiste na possibilidade de compartilhar o acesso de Internet para todas as estações da rede interna a partir de um ou mais endereços ips públicos com o uso de regras de NAT (*Network Address Translation*).

Firewall de host

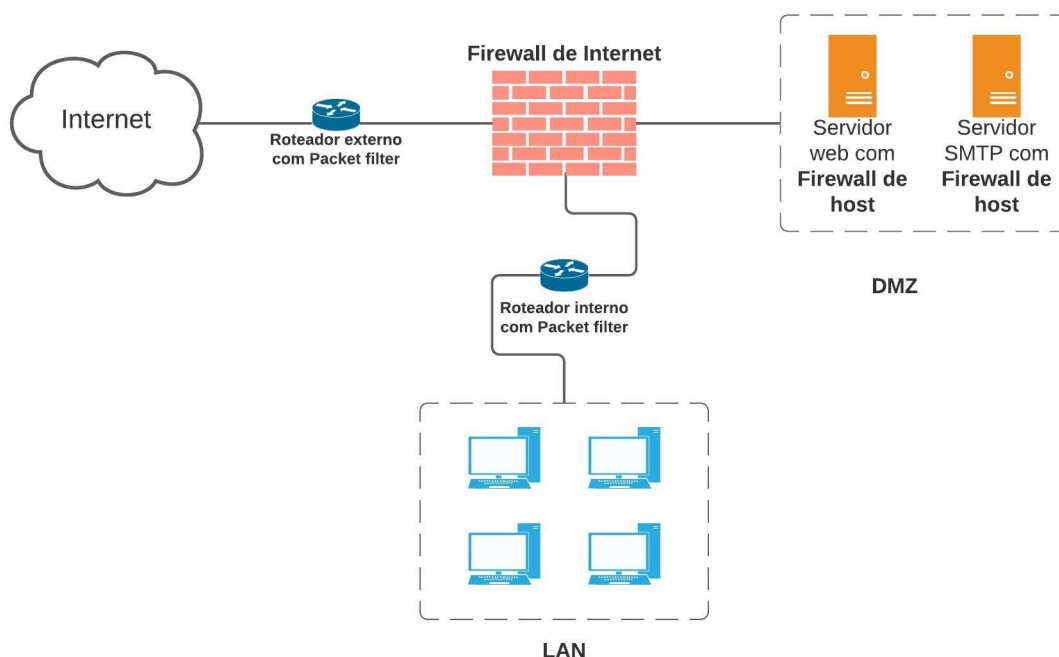
De acordo com NIST (2009), o *firewall* baseado em *host* (*host-based firewall*) reside no próprio *host* e pode bloquear ou liberar tráfego de entrada e saída baseado em suas regras de configuração. O tráfego de entrada tem como destino o *host* e o tráfego de saída, por sua vez, origina-se no *host* com destino a uma outra máquina.

Enquanto o *firewall* de Internet atua como um roteador e controla o tráfego da rede interna para a Internet e vice-versa, o *firewall* de *host* é baseado em *software* e protege a si próprio.

Arquiteturas de firewall

Os *firewalls* de Internet e de *host* são os mais utilizados quanto as características de atuação e podem ser posicionados em diferentes arquiteturas de *firewall*. A figura 7 mostra como o *firewall* de Internet e o *firewall* de *host* podem coexistir na arquitetura *screened subnet*.

Figura 7 - Arquitetura de firewall *screened subnet*.



Fonte: O autor.

Na figura 7, o *firewall* de Internet trabalha controlando o tráfego entre as redes enquanto os *firewalls* de *host* oferecem uma proteção exclusiva somente nos servidores onde estão instalados.

Técnicas de *firewall*

Segundo NIST (2009), as principais tecnologias de *firewall* comumente encontradas são as seguintes:

Packet filtering

Funcionalidade fundamental de qualquer *firewall* o *packet filtering* realiza filtragens básicas sobre os protocolos da pilha TCP/IP, entretanto, não possui recursos para inspecionar o conteúdo dos pacotes e não é capaz de detectar o estado dos fluxos de tráfego que passam por ele, por isso, também é conhecido como *stateless firewall*.

Considerando o modelo TCP/IP de 4 camadas (física, rede, transporte e aplicação) como referência para suas características de atuação, pode-se dizer que a filtragem de pacotes opera da seguinte maneira:

Tabela 6 - Atuação do *firewall packet filtering* com relação a pilha TCP/IP.

PILHA TCP/IP	EXEMPLOS
CAMADA FÍSICA	Filtros em MAC <i>address</i> origem e destino.
CAMADA DE REDE	Filtros em endereços IP origem, destino e protocolo ICMP.
CAMADA DE TRANSPORTE	Filtros em portas TCP e UDP. Como: 80/tcp, 443/tcp, 53/udp, 123/udp. Sendo um <i>firewall</i> sem estado (<i>stateless</i>), não consegue fazer regras que considerem o estado da conexão. Por exemplo: uma conexão TCP que tenha uma sessão estabelecida.
CAMADA DE APLICAÇÃO	Não atua nesta camada.

Fonte: NIST (2009). Adaptado pelo autor.

Stateful firewall

A verificação *stateful* amplia as capacidades do *packet filtering* pois monitora o estado das conexões, desta forma, pode bloquear pacotes que não estejam dentro do padrão de estado esperado.

A inspeção *stateful* aumenta a abrangência da filtragem na camada de transporte, principalmente com relação ao protocolo TCP que é orientado a conexão.

Na filtragem *stateful* cada conexão é controlada com base em uma tabela de estado. As informações contidas nesta tabela podem variar de acordo com a solução de *firewall*, mas normalmente incluem: endereço ip de origem, endereço ip de destino, números de porta e informações sobre o estado da conexão.

Tabela 7 - Tabela de estado de um *firewall stateful*.

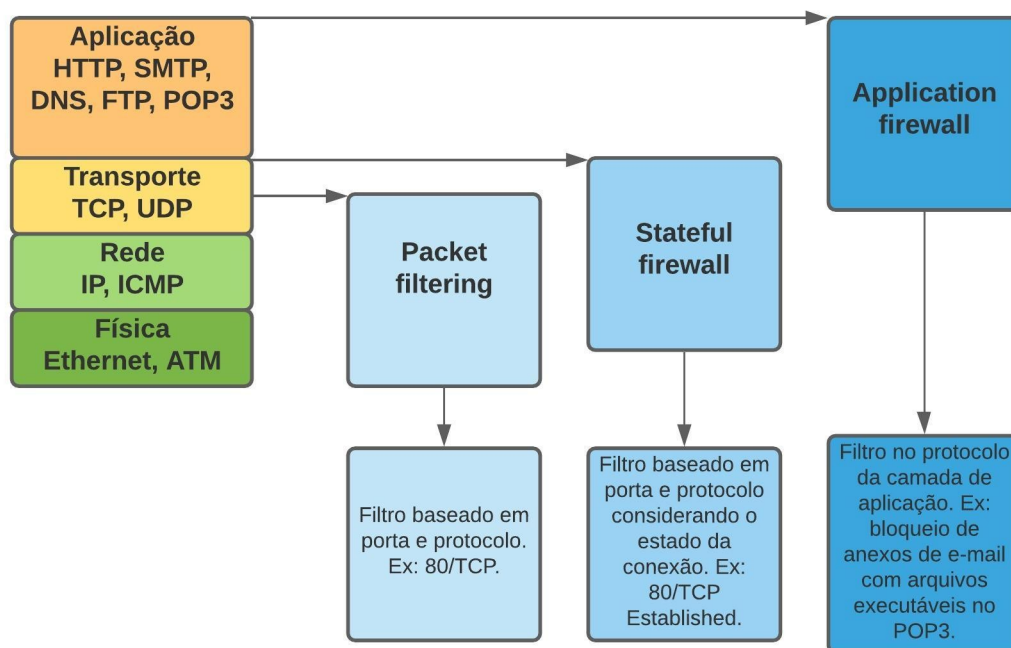
IP de origem	Porta de origem	IP de destino	Porta de destino	Estado da conexão
192.168.15.3	1050	192.0.5.8	443	<i>Initiated</i>
192.168.15.7	1052	172.16.6.7	80	<i>Established</i>
192.168.15.9	1055	172.16.6.19	22	<i>Established</i>
192.168.15.34	1057	172.16.6.39	587	<i>Established</i>

Fonte: NIST (2009). Adaptado pelo autor.

Application firewalls

Apesar de ampliar a capacidade de filtragem do *packet filtering* tradicional com a monitoração do estado das conexões, o filtro *stateful* não é capaz de identificar o conteúdo dos pacotes das aplicações que trafegam pela rede. Por exemplo: ele não tem como saber se o tráfego com destino a porta 80/TCP realmente faz uso do protocolo HTTP, ou se o fluxo da porta 110/TCP realmente está associado ao POP3. Um *application firewall*, por outro lado, pode averiguar os dados de um protocolo da camada de aplicação da pilha TCP/IP como o SMTP, para saber se uma mensagem de e-mail contém em anexo alguma extensão de arquivo proibida pela política de segurança da informação da empresa, como um executável, por exemplo.

Figura 8 - Tipos de firewalls e atuação na pilha TCP/IP.



Fonte: O autor.

Web Application firewalls (WAF)

Servidores *web* utilizam o HTTP. Invasores podem explorar este protocolo juntamente com falhas relacionadas a aplicação, a fim de descobrir informações privadas, ou mesmo, para inserir *software* malicioso no computador de algum usuário que esteja navegando no *site* deste servidor. Ataques de injeção, XSS e sequestro de sessão são comuns em aplicações *web*. Os *firewalls* conhecidos como *Web Application Firewalls* (WAF) são capazes de detectar muitas destas ameaças e podem trabalhar em conjunto com os sistemas de detecção de intrusão para mitigar estes ataques e enviar alertas para o administrador. Pode-se dizer que o WAF é um *firewall* de aplicação específico para o protocolo HTTP e aplicações *web*.

Netfilter/iptables

Um *firewall* de software nativo para ambientes Linux que possui recursos de *packet filtering* e *stateful firewall* é o Netfilter/iptables.

Conforme The Netfilter.Org Project (2020), o netfilter.org é um *framework* usado para realização de filtragem de pacotes em ambientes Linux e está presente desde a versão 2.4 do *kernel*. O programa regularmente associado ao netfilter.org é o iptables.

O netfilter já vem habilitado por padrão na maioria das distribuições Linux e suas principais características incluem:

- Recursos para implementação de *firewalls* de Internet e de *host* baseado em filtros de pacotes *stateless* e *stateful*.
- NAT e *masquerading* para compartilhar o acesso a Internet a partir de um único endereço ip público.
- Estrutura flexível e expansível.
- NAT para implementação de *transparent proxies*.
- NAPT (*network address port translation*).
- Manipulação de pacotes (*packet mangling*).

Conforme Gheorghe (2006), o netfilter trabalha com tabelas, sendo que, em cada tabela pode-se ter um conjunto padrão de regras conhecidas como *chains*. A tabela padrão carregada pelo *kernel* Linux é a tabela *filter*. As chains desta tabela também são utilizadas para definir a política padrão do *firewall*. São elas:

- **INPUT:** Possui regras para pacotes que tem como destino a própria máquina (pacotes entrantes).
- **FORWARD:** Contém regras para pacotes que serão “roteados” (encaminhados) pela máquina com destino a outros *hosts* em outras redes.
- **OUTPUT:** Regras para pacotes que são originados pela máquina, ou seja, que tem como origem a máquina com destino a outros *hosts*.

Os valores preestabelecidos para as chains de *INPUT*, *FORWARD* e *OUTPUT* é *ACCEPT*. isto significa que, por padrão, todo pacote pode entrar, ser encaminhado e sair de um *host* que adote a política *ACCEPT* para estas três *chains*. Neste cenário todos os pacotes são permitidos, desta forma, a fim de aumentar a segurança, outra política deve ser adotada para compor as regras do *firewall* de *host*.

A ABNT (2013), recomenda que sejam estabelecidas regras de controle de acesso baseadas na premissa de que **"Tudo é proibido a menos que expressamente permitido"**, isto posto, a política padrão sugerida para o *firewall* de *host* neste estudo é apresentada na tabela 8.

Tabela 8 - Política padrão de *firewall* de *host* com iptables.

CHAIN	VALOR
<i>INPUT</i>	<i>DROP</i>
<i>FORWARD</i>	<i>DROP</i>
<i>OUTPUT</i>	<i>ACCEPT</i>

Fonte: O autor.

Com esta política, nenhum pacote entra ou é encaminhado pelo *firewall*. Somente a *chain OUTPUT* permanece com o valor *ACCEPT* por se tratar de pacotes que tem como origem o próprio *firewall*, entretanto, em alguns cenários que

exijam mais controle e segurança para o tráfego de saída, a *chain OUTPUT* também deve ser ajustada como *DROP*.

A partir deste ponto, deve-se trabalhar com as exceções e liberar acesso somente para os serviços e protocolos que o servidor vai disponibilizar para os usuários. Lembrando que, tudo aquilo que não for permitido já está bloqueado.

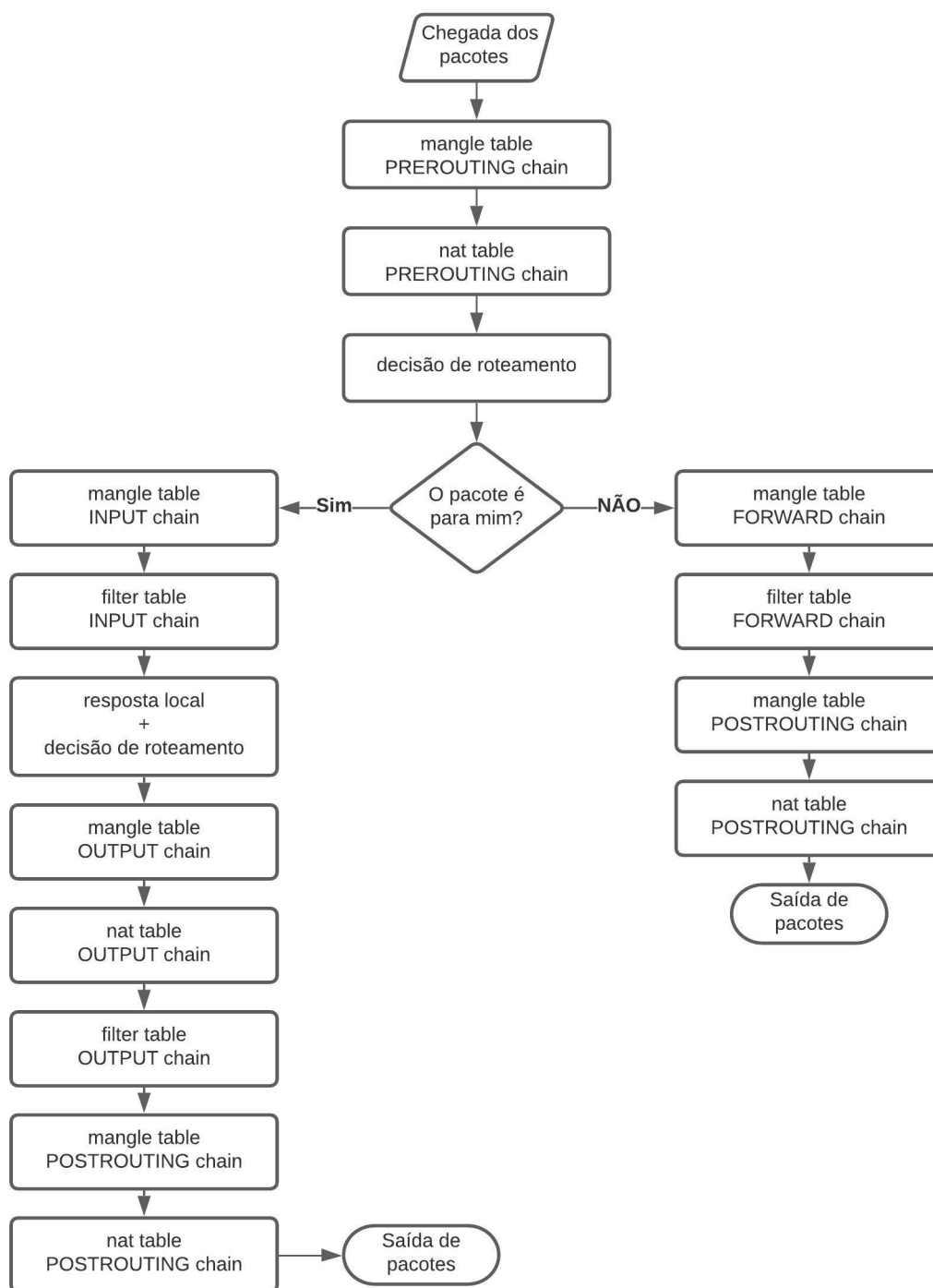
Firewall de host para o servidor LAMP com utilização de recursos de *packet filtering* e *stateful firewall*

O *firewall* de *host* vai oferecer mais segurança para os serviços de rede da infraestrutura LAMP, uma vez que, estabelece regras para controle de acesso. O conhecimento das características dos protocolos de rede de cada serviço servem de base para estabelecimento das regras de filtragem, isto posto, com relação as portas de serviço o Apache normalmente “escuta” (estado *listen*) na porta 80/TCP (HTTP) ou na porta 443/TCP (quando tem suporte para HTTPS) e o servidor Mysql “ouve” na porta 3306/TCP.

Para administrar e gerenciar o servidor é utilizado o serviço de acesso remoto SSH (*Secure Shell*) que tem suporte a criptografia, portanto, mais seguro que outros serviços que não suportam encriptação e são usados para a mesma finalidade, como o Telnet. Por padrão, o SSH faz uso da porta 22/TCP, mas por questões de segurança é recomendado que se utilize uma outra porta.

Os pacotes de rede que chegam no *kernel* Linux e passam pelo *framework* netfilter com suas *tabelas* e *chains*, seguem o fluxo mostrado na figura 9.

Figura 9 - Fluxo de pacotes nas tabelas do netfilter/iptables.



Fonte: Gheorghe (2006).

Várias regras de controle de acesso são utilizadas para constituir um *firewall*.

Exemplos de regras de *firewall* de *host* que podem ser digitadas na linha de comando a partir de um *shell* Linux no servidor LAMP:

Definição da política padrão:

Figura 10 - Política padrão para o firewall de host.

```
# iptables -P INPUT DROP  
# iptables -P FORWARD DROP  
# iptables -P OUTPUT ACCEPT
```

Fonte: O autor.

Liberação de acesso total para o endereço de localhost:

Figura 11 - Acesso total para 127.0.0.1 (loopback).

```
# iptables -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

Fonte: O autor.

Para permitir que o *firewall* responda ao comando ping (pacotes ICMP *echo request*). O módulo *limit* é utilizado para limitar a entrada de pacotes em cinco por segundo:

Figura 12 - Permitindo a entrada de pacotes ICMP *echo request* (ping).

```
# iptables -A INPUT -p icmp --icmp-type 8 -i <interface de rede> -m limit --limit 5/sec -j ACCEPT
```

Fonte: O autor.

Permissão de acesso ao servidor SSH na porta 35777 (somente pacotes de início de conexão):

Figura 13 - Permitindo acesso ao servidor SSH na porta TCP 35777.

```
# iptables -A INPUT -p tcp --syn -i <interface de rede> --dport 35777 -m state --state NEW -j ACCEPT
```

Fonte: O autor.

Permissão de acesso ao servidor web Apache nas portas 80 e 443/TCP (somente pacotes de início de conexão):

Figura 14 - Permitindo acesso ao servidor web (HTTP e HTTPS).

```
# iptables -A INPUT -p tcp --syn -i <interface de rede> -m multiport --dports 80,443 -m state --state NEW -j ACCEPT
```

Fonte: O autor.

Permissão de acesso ao servidor Mysql na porta 3306/TCP somente para os endereços ips: 192.168.0.180 e 192.168.0.181 (somente pacotes de início de conexão):

Figura 15 - Acesso ao servidor Mysql somente para os ips 192.168.0.180 e 192.168.0.181.

```
# iptables -A INPUT -s 192.168.0.180/32 -p tcp --syn --dport 3306 -m state --state NEW -j ACCEPT  
# iptables -A INPUT -s 192.168.0.181/32 -p tcp --syn --dport 3306 -m state --state NEW -j ACCEPT
```

Fonte: O autor.

Deve-se garantir também, que todas as solicitações feitas a partir do próprio servidor LAMP tenham respostas. Exemplo: se o servidor enviar um “*ping*” para um determinado *host*, esta regra garante que o *firewall* aceite a resposta deste *host*:

Figura 16 - Garantindo respostas para as solicitações do próprio firewall.

```
# iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
```

Fonte: O autor.

Por fim, todas estas regras devem ser colocadas em um *script shell* para que sejam carregadas na inicialização do servidor LAMP. A figura 17 mostra o seu código:

Figura 17 - *Script Shell* para *firewall* de *host*.

```
#!/bin/bash

### BEGIN INIT INFO
# Provides:      firewall
# Required-Start: $remote_fs $syslog $named $network $time
# Required-Stop:  $remote_fs $syslog $named $network
# Default-Start:  2 3 4 5
# Default-Stop:   0 1 6
# Short-Description: Host firewall
# Description:    Firewall is a Firewall de Host
### END INIT INFO

PF="/sbin/iptables"
IFACE_LAN="eth1"
REDE_LAN="192.168.0.0/24"

inicio () {
# Definição da política padrao
$PF -P INPUT DROP
$PF -P FORWARD DROP
$PF -P OUTPUT ACCEPT

# Liberação de acesso total para o endereço de localhost
$PF -A INPUT -i lo -s 127.0.0.1 -d 127.0.0.1 -j ACCEPT
```

```

# Para permitir que o firewall responda ao comando ping (pacotes ICMP echo request)
# Limite de 5 pacotes por segundo.
$PF -A INPUT -p icmp --icmp-type 8 -i $IFACE_LAN -m limit --limit 5/sec -j ACCEPT

# Permissão de acesso ao servidor SSH na porta 35777 (somente pacotes de inicio de
conexão)
$PF -A INPUT -p tcp --syn -i $IFACE_LAN --dport 35777 -m state --state NEW -j ACCEPT

# Permissão de acesso ao servidor web Apache nas portas 80 e 443/TCP (somente pacotes
de inicio de conexão):

$PF -A INPUT -p tcp --syn -i $IFACE_LAN -m multiport --dports 80,443 -m state --state NEW -j
ACCEPT

# Acesso ao servidor Mysql somente para os ips 192.168.0.180 e 192.168.0.181 da LAN.
# Somente pacotes de inicio de conexão:
$PF -A INPUT -s 192.168.0.4/32 -p tcp --syn --dport 3306 -m state --state NEW -j ACCEPT
$PF -A INPUT -s 192.168.0.181/32 -p tcp --syn --dport 3306 -m state --state NEW -j ACCEPT

# Regra para garantir respostas de todas as conexões relatadas e estabelecidas pelo
servidor LAMP:
$PF -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT

}

parar () {

echo "Finalizando Firewall"

$PF -Z
$PF -Z -t nat
$PF -Z -t mangle
$PF -F
$PF -F -t nat
$PF -F -t mangle

```

```
$PF -P INPUT ACCEPT
$PF -P FORWARD ACCEPT
$PF -P OUTPUT ACCEPT

}

case $1 in

start)

inicio

;;

stop)

parar

;;

*)

echo "Erro! Use $0 [start|stop]"

exit 1

;;

esac

exit 0
```

Fonte: O autor.

Para que o *script* de *firewall* seja carregado na inicialização do sistema os seguintes passos devem ser executados:

Criação do arquivo ***firewall-host.sh*** no diretório ***/etc/init.d*** com o conteúdo da figura 17. Permissão de execução para o arquivo, com o comando da figura 18:

Figura 18 - Atribuindo permissão de execução para o *script* de *firewall*.

```
# chmod 755 /etc/init.d/firewall-host.sh
```

Fonte: O autor.

Habilitação do serviço de *firewall* no sistema de inicialização padrão da distribuição Linux, conforme figura 19. O Debian Linux 10 faz uso do *Systemd*.

Figura 19 - Habilitando o script para carregar na inicialização do sistema.

```
# cd /etc/init.d  
# systemctl enable firewall-host.sh  
# systemctl daemon-reload
```

Fonte: O autor.

2.6 *Hardening* para o servidor web Apache

O servidor *web* Apache pode ser instalado com suporte a PHP no Debian Linux 10, com os comandos da figura 20:

Figura 20 - Instalação do Apache com suporte a PHP.

```
# apt-get update  
# apt-get install apache2 php7.3 libapache2-mod-php7.3  
# systemctl restart apache2.service
```

Fonte: O autor.

2.6.1 Ocultando Informações sobre o servidor

Após a instalação do Apache, deve-se ocultar as informações sobre a sua versão. Na configuração padrão, as informações sobre a versão do Apache e da distribuição Linux são exibidas juntamente com a mensagem de erro, conforme mostra a figura 21. No exemplo, foi efetuado acesso em um diretório inexistente.

Figura 21 - Informações sobre a versão do Apache e da distribuição Linux.



Fonte: O autor.

A diretiva **ServerTokens** deve ser configurada com o parâmetro “**Prod**” no arquivo de configuração **/etc/apache2/conf-enabled/security.conf**. Depois da mudança, as configurações devem ser recarregadas com o comando da figura 22:

Figura 22 - Ativando as novas configurações do Apache.

```
# systemctl reload apache2.service
```

Fonte: O autor.

Figura 23 - Ocultando informações sobre a versão do Apache e da distribuição Linux.



Fonte: O autor.

Quando a página é acessada novamente, o conteúdo da figura 23 é exibido. Apesar de informações sobre a versão do Apache e do sistema operacional não serem mais apresentadas, ainda é exibido que o Apache é o *software* utilizado como servidor *web*.

Para que nenhuma informação apareça a diretiva *ServerSignature* deve ser desabilitada. Para tanto, o arquivo */etc/apache2/conf-enabled/security.conf* deve ser editado e a diretiva ***ServerSignature*** configurada com o parâmetro “Off”. Com as novas configurações habilitadas, apenas a informação sobre o erro de objeto não encontrado (**Not Found**) é exibida para o usuário.

Figura 24 - Nenhuma informação sobre versões de serviço é exibida.



Fonte: O autor.

2.6.2 Autenticação, autorização e controle de acesso

De acordo com a Apache Docs Auth (2020), autenticação é o processo utilizado para verificar a identidade do usuário, ou seja, se ele é realmente quem afirma ser. O processo de autorização, por sua vez, determina quais recursos e dados um usuário pode acessar. Tem a finalidade de impedir que pessoas não autorizadas acessem dados e outros recursos da rede. Por fim, o controle de acesso estabelece e implanta controles para que um determinado recurso seja acessado.

Como exemplo, para proteger com recursos de autenticação e autorização uma pasta chamada projetos localizada no diretório raiz (*document root*) do Apache deve-se proceder da seguinte maneira:

O arquivo **/etc/apache2/apache2.conf** deve ser editado com o trecho de código da figura 25:

Figura 25 - Controlando o acesso na pasta projetos.

```
<Directory /var/www/html/projetos/>  
    AuthType Basic  
    AuthName "Arquivos confidenciais "  
    AuthUserFile "/usr/local/usuarios.txt"  
    Require valid-user  
</Directory>
```

Fonte: O autor.

Na distribuição Debian Linux 10, a pasta raiz *web* fica em **/var/www/html**.

Os usuários com autorização para acessar a pasta projetos devem ser criados com os comandos da figura 26 e 27:

Figura 26 - Utilização do comando htpasswd para criação do arquivo de usuários.

```
# htpasswd -B -c /usr/local/usuarios.txt Garcia
```

Fonte: O autor.

Onde:

-B: Usa a encriptação bcrypt para a senha.**-c:** Cria o arquivo /usr/local/usuarios.txt para armazenar usuário e senha.**Garcia:** nome do usuário a ser criado.

Para criação do segundo usuário de nome Fonseca não é necessário usar o parâmetro “-c” uma vez que o arquivo /usr/local/usuarios.txt já está criado.

Figura 27 - Criação do segundo usuário.

```
# htpasswd -B /usr/local/usuarios.txt Fonseca
```

Fonte: O autor.

A criação das contas pode ser conferida visualizando-se o conteúdo do arquivo /usr/local/usuarios.txt, conforme figura 28:

Figura 28 - Visualização do conteúdo do arquivo de usuários.

```
# cat /usr/local/usuarios.txt  
Garcia:$apr1$OiRPBQVn$MgYgRXqQ5AKHflliClquL.  
Fonseca:$apr1$mg3Z/v6.$BWZupvih8N5RhPcDdUbgO1
```

Fonte: O autor.

O Apache deve ser reiniciado para que as configurações entrem em vigor.

Figura 29 - Atualizando as configurações do Apache.

```
# systemctl restart apache2.service
```

Fonte: O autor.

Após realização da configuração somente os usuários que efetuarem o processo de autenticação com usuário e senha terão autorização para acessar os recursos do diretório `/var/www/html/projetos`.

Figura 30 - Usuário realizando autenticação para acessar pasta do Apache.



Fonte: O autor.

A fim de aumentar o nível de segurança, além dos processos de autenticação e autorização, um controle de acesso por ip pode ser estabelecido.

O arquivo `/etc/apache2/apache2.conf` deve ser editado de acordo com a figura 31 para que seja acrescentada as seguintes diretivas na seção `<Directory /var/www/html/projetos/>`:

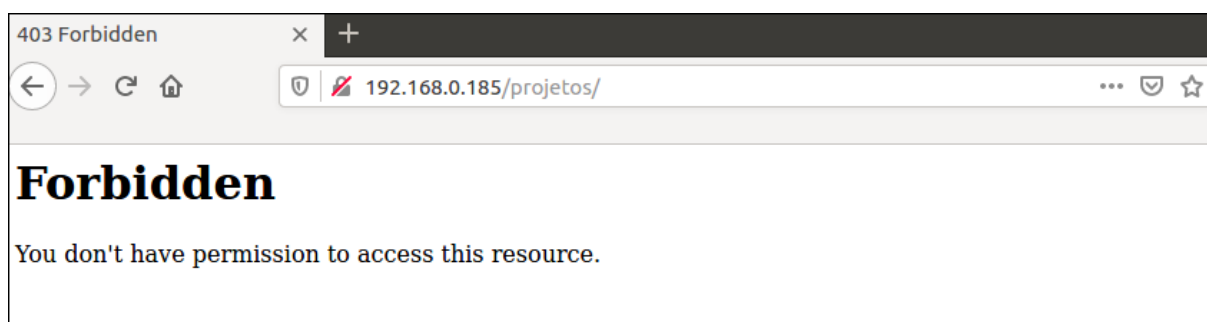
Figura 31 - Permitindo acesso na pasta projetos somente para o ip 192.168.0.26.

```
<Directory /var/www/html/projetos/>
  Require all denied
<RequireAll>
  <RequireAny>
    require ip 192.168.0.26
  </RequireAny>
</RequireAll>
  AuthType Basic
  AuthName "Arquivos confidenciais "
  AuthUserFile "/usr/local/usuarios.txt"
  Require valid-user
</RequireAll>
</RequireAll>
</Directory>
```

Fonte: O autor.

Nesta nova configuração somente o endereço ip 192.168.0.26 poderá acessar os recursos da pasta /var/www/html/projetos. Mesmo assim, somente depois que se autenticar com um usuário válido. Qualquer outro ip terá o acesso recusado imediatamente e sem que a tela de autenticação apareça, conforme mostra a figura 32.

Figura 32 - Acesso proibido para endereço ip não autorizado.



Fonte: O autor.

O arquivo de logs de erro do Apache `/var/log/apache2/error.log` mostra o bloqueio do endereço ip indesejável.

Figura 33 - Arquivo de *logs* do Apache mostra o bloqueio do ip indesejável.

```
[Wed Aug 26 22:30:18.485298 2020] [authz_core:error] [pid 6461:tid 140273462605568] [client 192.168.0.4:57760] AH01630: client denied by server configuration: /var/www/html/projetos/
```

Fonte: O autor.

2.6.3 Acesso seguro com SSL/TLS

Conforme Cloudflare (2020), o SSL (*Secure Sockets Layer*) é um protocolo baseado em criptografia que foi desenvolvido pela Netscape em 1995 com a finalidade de garantir privacidade, autenticação e integridade para os dados que trafegam na Internet.

O SSL protege os dados com autenticação mútua, uso de assinaturas digitais para verificação de integridade e criptografia para garantia de privacidade.

Normalmente é utilizado para proteger a comunicação entre um navegador e um servidor *web*. Utiliza o protocolo HTTPS e por padrão usa a porta 443/TCP. Isto significa que, todos os *sites* que fazem uso do SSL devem ser acessados com o protocolo HTTPS ao invés de HTTP, que não tem suporte a criptografia (APACHE DOCS SSL/TLS, 2020).

O TLS (*Transport Layer Security*) é a versão aprimorada e melhorada do SSL que é utilizada nos dias atuais.

Segundo Apache Docs Ssl/tls (2020), o protocolo foi concebido para suportar uma grande variedade de opções de algoritmos usados para criptografia, autenticação e assinaturas digitais, sendo que, durante o estabelecimento da sessão do protocolo o cliente e o servidor negociam os algoritmos que vão utilizar.

O módulo do Apache que provê suporte para SSL/TLS chama-se *mod_ssl*.

É necessário o uso de um certificado fornecido por uma autoridade certificadora (AC) para que o HTTPS possa ser habilitado em um *site* de Internet. O *site* de Internet, por sua vez, precisa de um domínio de Internet válido como: site.com.br ou www.site.com.br para que a AC consiga atestar a autenticidade do mesmo (LET'S ENCRYPT, 2020).

Também é possível para propósitos de testes e utilização em redes internas gerar e usar um certificado autoassinado, entretanto, apesar de prover criptografia e todas as funcionalidades de segurança, ele não será reconhecido como válido por nenhuma autoridade certificadora. Já para oferecer um certificado SSL/TLS para um site em um domínio de Internet válido a AC gratuita Let's Encrypt pode ser usada.

Geração e configuração de um certificado autoassinado para o Apache

O pacote openssl deve ser instalado:

Figura 34 - Instalação do pacote openssl.

```
# apt-get install openssl
```

Fonte: O autor.

Para habilitar suporte ao SSL, o módulo ***mod_ssl*** e a página padrão com as suas respectivas configurações (default-ssl.conf) devem ser ativados de acordo com a figura 35:

Figura 35 - Ativação do módulo mod_ssl para o Apache.

```
# a2enmod ssl  
# a2ensite default-ssl.conf
```

Fonte: O autor.

Um diretório chamado ssl deve ser criado em /etc/apache2. Deve-se entrar no mesmo para gerar o arquivo de certificado e a chave privada:

Figura 36 - Geração do arquivo de certificado e chave privada.

```
# mkdir /etc/apache2/ssl  
# cd /etc/apache2/ssl  
# openssl req -new -x509 -nodes -out server.crt -keyout server.key
```

Fonte: O autor.

O comando openssl vai gerar o arquivo de certificado **server.crt** e a chave privada **server.key**. Na sequência, entre no diretório /etc/apache2/sites-enabled e edite o arquivo *default-ssl.conf* para indicar o caminho do arquivo de certificado e da chave privada que foram gerados:

Figura 37 - Edição do arquivo default-ssl.conf.

```
# cd /etc/apache2/sites-enabled  
# nano default-ssl.conf
```

Fonte: O autor.

A figura 38 indica as linhas a serem alteradas no arquivo *default-ssl.conf*:

Figura 38 - Indicação do caminho do arquivo de certificado e da chave privada.

```
SSLCertificateFile /etc/apache2/ssl/server.crt  
SSLCertificateKeyFile /etc/apache2/ssl/server.key
```

Fonte: O autor.

Por fim, o Apache deve ser reiniciado para que um teste com o navegador possa ser realizado.

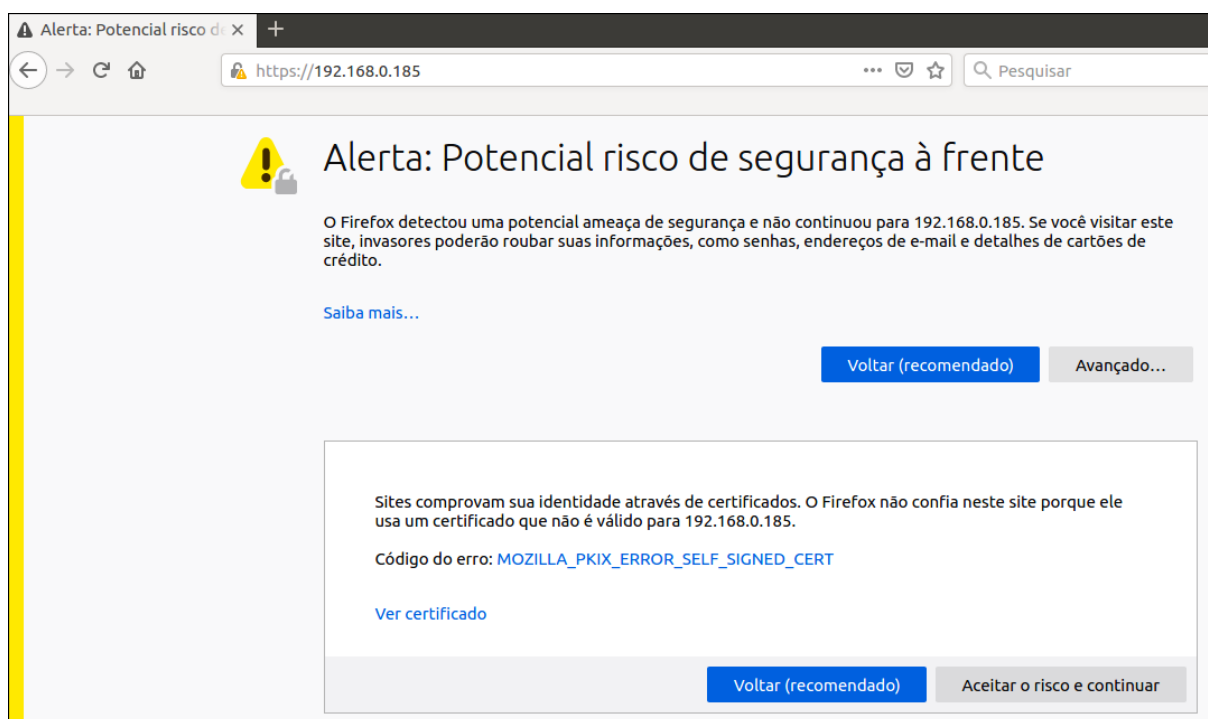
Figura 39 - Reinicialização do Apache.

```
# systemctl restart apache2.service
```

Fonte: O autor.

A figura 40 mostra que o navegador não aceita o certificado autoassinado como válido, apesar dele funcionar com HTTPS e fornecer os recursos de segurança do SSL/TLS.

Figura 40 - Acesso em servidor *web* Apache com certificado autoassinado.



Fonte: O autor.

Let's Encrypt para domínios de Internet

A Let's Encrypt é uma AC que pode fornecer um certificado gratuito para um *site* de um domínio de Internet. A solução usa um *software* que faz uso do protocolo ACME e que é comumente executado no servidor *web* para fazer a verificação de validade de um domínio.

2.7 Hardening para a aplicação PHP

Nesta etapa, algumas das principais vulnerabilidades comumente encontradas em aplicações *web* baseadas em PHP serão explanadas. Também serão abordados os métodos utilizados para que as mesmas sejam corrigidas.

2.7.1 Directory Traversal

Segundo Acunetix Security Directory Traversal & Code Injection (2020), o ataque de *Directory traversal* ou *Path traversal* compromete o sistema de arquivos do servidor *web*. Neste cenário, quaisquer usuários (autenticados ou não) são capazes de visualizar ou executar arquivos que residem fora da raiz *web* do servidor. No Debian Linux o diretório raiz fica normalmente em `/var/www/html`.

Deste modo, o conteúdo de arquivos presentes em outros lugares da estrutura de diretórios como: `/etc/passwd`, `/proc/version`, `/etc/issue.net`, entre outros, poderão ser exibidos. Um código mal escrito e não validado no servidor é o suficiente para que o ataque seja explorado.

Para exemplificar, um arquivo chamado ***insecure_code.php*** é criado em `/var/www/html` (raiz *web* do Apache) com o conteúdo da figura 41.

Figura 41 - Arquivo `insecure_code.php`.

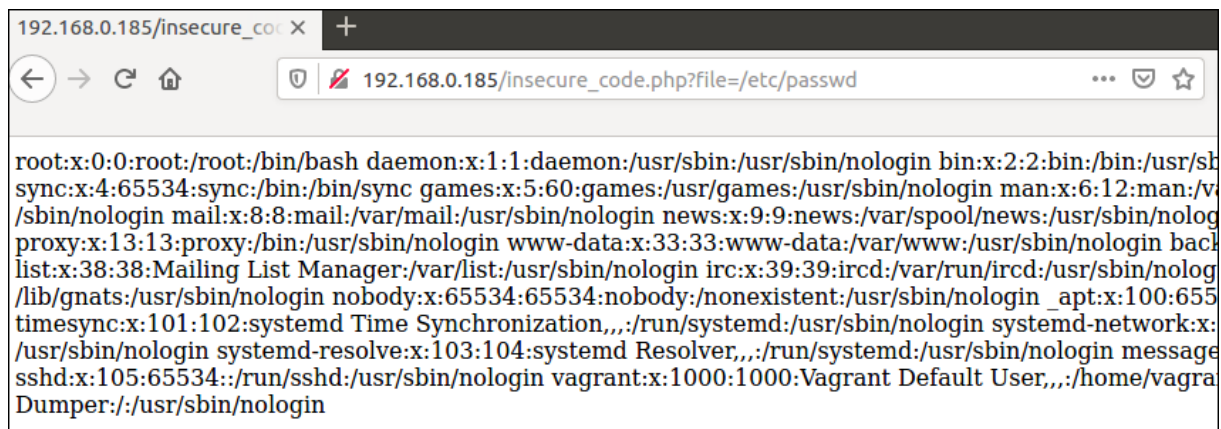
```
<?php
$file = $_GET['file'];
include($file);
?>
```

Fonte: Acunetix Security Directory Traversal & Code Injection (2020).

A exploração da falha pode ser observada ao entrar com a seguinte url no navegador: ***http://192.168.0.185/insecure_code.php?file=/etc/passwd***. Nesta

investida, o invasor visualiza o conteúdo de `/etc/passwd` (arquivo que contém os usuários do sistema) conforme mostra a figura 42.

Figura 42 - Arquivo `/etc/passwd` sendo visualizado de forma arbitrária.



Fonte: O autor.

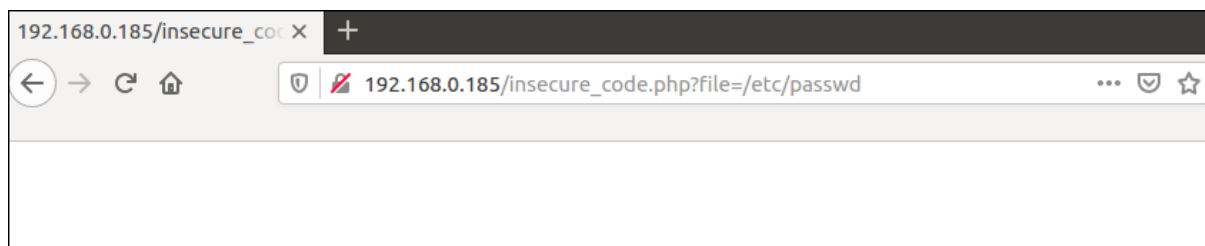
A falha pode ser corrigida usando-se as funções ***basename*** e ***realpath*** do PHP. Deve-se modificar o arquivo `insecure_code.php` de acordo com a figura 43.

Figura 43 - Codificação segura do arquivo `insecure_code.php`.

```
<?php
$file = basename(realpath($_GET['file']));
include($file);
?>
```

Fonte: Acunetix Security Directory Traversal & Code Injection (2020).

A mesma *url* maliciosa agora não exibe mais o conteúdo de arquivos do sistema que estejam fora da raiz *web* do servidor, conforme mostra a figura 44.

Figura 44 - Vulnerabilidade de *Directory Traversal* corrigida.

Fonte: O autor.

2.7.2 Code Injection

Os ataques de *code injection* exploram funções do PHP que fazem chamadas de sistema para executar comandos maliciosos no servidor. Neste contexto, o invasor terá um *shell backdoor* a sua disposição, uma vez que, poderá executar qualquer comando com permissão de usuário do Apache. Em determinadas circunstâncias, esta falha também poderá abrir caminho para a escalção de privilégios, cenário onde o atacante tenta ganhar acesso a um *shell* de root (ACUNETIX SECURITY DIRECTORY TRAVERSAL & CODE INJECTION, 2020).

Um arquivo chamado *monitora.php* usa a função *exec* do PHP para executar o comando ping do *shell* em um *host* que é passado de forma dinâmica via HTTP *GET request*. O arquivo deve ser criado na raiz *web* do servidor e seu conteúdo é exibido na figura 45.

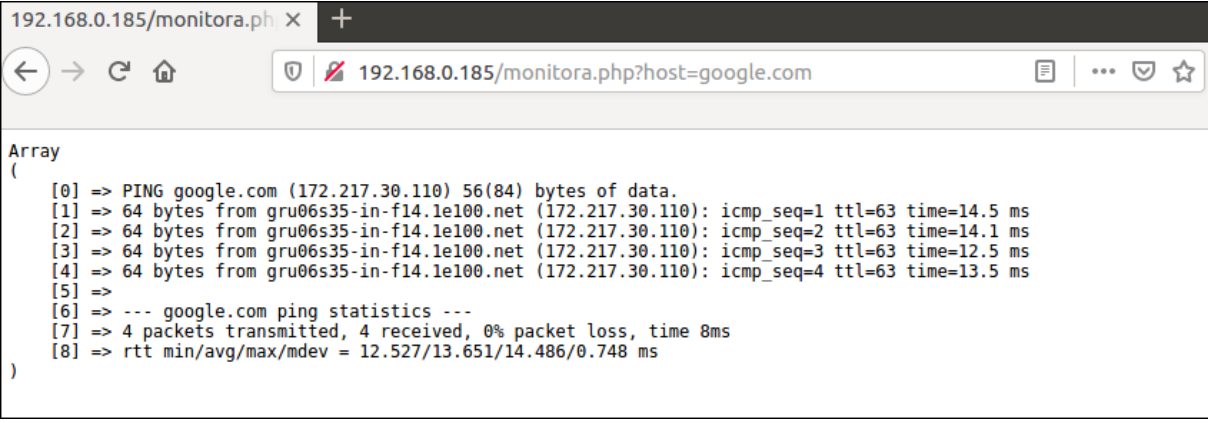
Figura 45 - Arquivo *monitora.php*.

```
<?php
exec("ping -c 4 " . $_GET['host'], $output);
echo "<pre>";
print_r($output);
echo "</pre>";
?>
```

Fonte: Acunetix Security Directory Traversal & Code Injection (2020).

A figura 46 mostra o programa sendo executado da maneira esperada com a url ***http://192.168.0.185/monitora.php?host=google.com*** passando como parâmetro o *host* que deve ser verificado. No exemplo google.com.

Figura 46 - Programa monitora.php sendo executado da maneira esperada.



The screenshot shows a web browser window with the address bar displaying `192.168.0.185/monitora.php?host=google.com`. The page content displays a PHP array containing the output of a ping command to google.com. The array is structured as follows:

```
Array
(
    [0] => PING google.com (172.217.30.110) 56(84) bytes of data.
    [1] => 64 bytes from gru06s35-in-f14.1e100.net (172.217.30.110): icmp_seq=1 ttl=63 time=14.5 ms
    [2] => 64 bytes from gru06s35-in-f14.1e100.net (172.217.30.110): icmp_seq=2 ttl=63 time=14.1 ms
    [3] => 64 bytes from gru06s35-in-f14.1e100.net (172.217.30.110): icmp_seq=3 ttl=63 time=12.5 ms
    [4] => 64 bytes from gru06s35-in-f14.1e100.net (172.217.30.110): icmp_seq=4 ttl=63 time=13.5 ms
    [5] =>
    [6] => --- google.com ping statistics ---
    [7] => 4 packets transmitted, 4 received, 0% packet loss, time 8ms
    [8] => rtt min/avg/max/mdev = 12.527/13.651/14.486/0.748 ms
)
```

Fonte: O autor.

Entretanto, na figura 47, o programa é executado de maneira inusitada. No Linux podemos executar vários comandos em sequência seguidos do caracter “;”.

Assim sendo, o invasor tira proveito deste recurso para executar o código de forma arbitrária e assim inserir os comandos que desejar.

Figura 47 - Programa monitora.php sendo usado para executar comandos do sistema.

```

Array
(
    [0] => PING google.com (172.217.30.78) 56(84) bytes of data.
    [1] => 64 bytes from gru06s34-in-f14.1e100.net (172.217.30.78): icmp_seq=1 ttl=63 time=12.5 ms
    [2] => 64 bytes from gru06s34-in-f14.1e100.net (172.217.30.78): icmp_seq=2 ttl=63 time=20.0 ms
    [3] => 64 bytes from gru06s34-in-f14.1e100.net (172.217.30.78): icmp_seq=3 ttl=63 time=16.4 ms
    [4] => 64 bytes from gru06s34-in-f14.1e100.net (172.217.30.78): icmp_seq=4 ttl=63 time=15.3 ms
    [5] =>
    [6] => --- google.com ping statistics ---
    [7] => 4 packets transmitted, 4 received, 0% packet loss, time 8ms
    [8] => rtt min/avg/max/mdev = 12.460/16.062/20.033/2.710 ms
    [9] => www-data
    [10] => 11:47:46 up 3 days, 13:36, 1 user, load average: 0.00, 0.00, 0.00
)
  
```

Fonte: O autor.

A seguinte *url* maliciosa foi digitada no navegador pelo invasor: ***http://192.168.0.185/monitora.php?host=google.com;whoami;uptime***. O mesmo usou o comando *whoami* para saber que usuário executa os processos do Apache e o comando *uptime* para descobrir o tempo de disponibilidade do servidor, bem como, sua carga de CPU.

As funções ***escapeshellcmd*** e ***escapeshellarg*** podem aumentar a segurança do código quanto a utilização de funções que fazem chamadas de sistema, como: *exec*, *shell_exec*, *passthru* e *system*.

A função *escapeshellcmd* pode ser usada para restringir o uso de múltiplos comandos, enquanto a função *escapeshellarg* coíbe a utilização de múltiplos argumentos. A figura 48 mostra a nova versão do arquivo *monitora.php*.

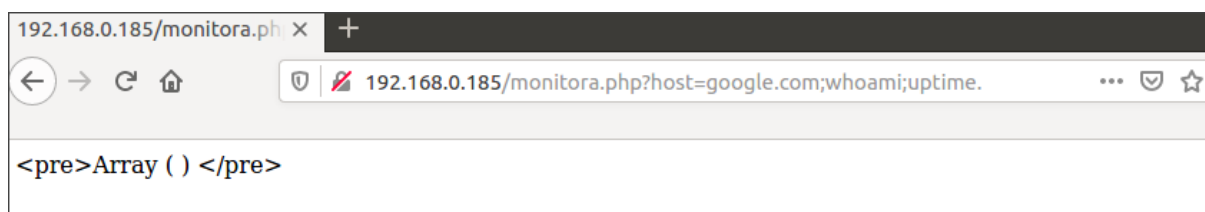
Figura 48 - Codificação segura do arquivo monitora.php.

```
<?php
exec(escapeshellcmd("ping -c 4 " . escapeshellarg($_GET['host'])), $output);
echo "<pre>";
print_r($output);
echo "</pre>";
?>
```

Fonte: Acunetix Security Directory Traversal & Code Injection (2020).

Após as adequações de segurança no código monitora.php, não é mais possível executar comandos do sistema de forma abusiva.

Figura 49 - Programa monitora.php sem executar comandos do sistema.



Fonte: O autor.

É recomendável desabilitar as funções do PHP que fazem chamadas para execução de comandos do sistema como: **exec**, **shell_exec**, **passthru** e **system**.

Estas funções devem ser usadas somente nos casos estritamente necessários.

Para desativar estas funções o arquivo de configuração do PHP (php.ini) deve ser editado. No Debian Linux 10, o arquivo de configuração para a versão 7.3 do PHP é o `/etc/php/7.3/apache2/php.ini`. As funções a serem desabilitadas devem ser adicionadas como valor para o parâmetro `disable_functions` de acordo com a figura 50.

Figura 50 - Desativação de funções PHP que podem executar comandos de sistema.

```
disable_functions = exec,shell_exec,passthru,system
```

Fonte: O autor.

Deve-se reiniciar o Apache para que as configurações entrem em vigor.

Figura 51 - Reinicialização do Apache.

```
# systemctl restart apache2.service
```

Fonte: O autor.

2.7.3 SQL injection

A criação de uma infraestrutura com banco e uma tabela contendo dados é necessária para demonstrar a vulnerabilidade de *Sql injection*, portanto, um sistema gerenciador de banco de dados deve ser instalado. E este estudo tem como foco o servidor Mysql.

Atualmente o Mysql pertence a empresa Oracle e a distribuição Debian Linux 10 utiliza o MariaDB como sistema gerenciador de banco de dados padrão. Entretanto, a Oracle continua disponibilizando o Mysql de forma gratuita e *open source* através do pacote *MySQL Community Edition*.

Para instalação do *MySQL Community Edition* no Debian Linux 10, os seguintes procedimentos devem ser seguidos:

A Oracle disponibiliza um pacote no formato .deb chamado *mysql-apt-config* para atualizar o repositório de programas do Debian, de maneira que, o mesmo fique atualizado para instalação do Mysql e possa ser instalado via gerenciador de pacotes *apt-get*. Um cadastro deve ser feito no *site* da Oracle antes que o *download* do pacote seja realizado.

Figura 52 - Download do Mysql Community.



Fonte: O autor.

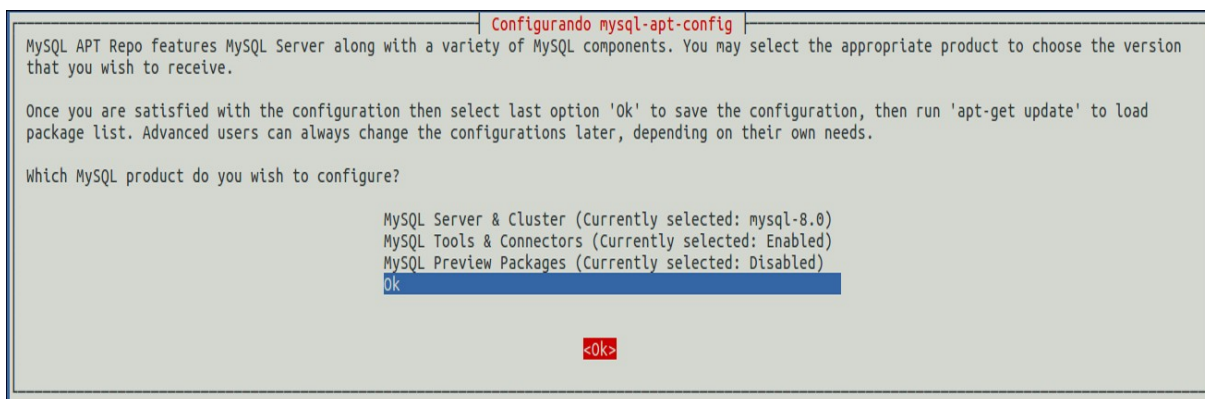
Após o *download*, as dependências e o pacote devem ser instalados conforme comandos da figura 53:

Figura 53 - Instalando os pacotes do Mysql Community

```
# apt-get update  
# apt-get install gnupg  
# dpkg -i mysql-apt-config_<versão_atual>_all.deb
```

Fonte: O autor.

Figura 54 - Seleção dos pacotes do Mysql Community.



Fonte: O autor.

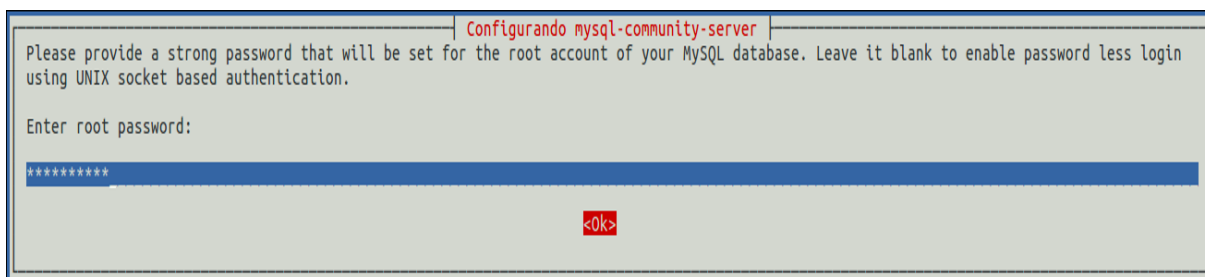
Posteriormente, a base de repositórios deve ser atualizada novamente e os pacotes *mysql-community-server* e *php7.3-mysql* (suporte Mysql para o PHP) instalados:

Figura 55 - Instalação do Mysql e do PHP.

```
# apt-get update
# apt-get install php7.3-mysql
# systemctl restart apache2.service
# apt-get install mysql-community-server
```

Fonte: O autor.

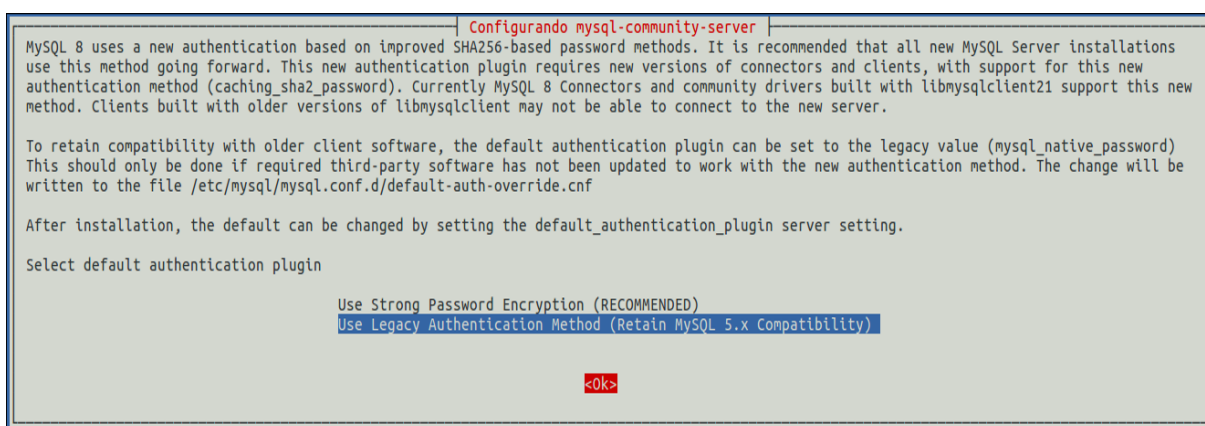
Figura 56 - Escolha da senha de root do Mysql.



Fonte: O autor.

Como mostra a figura 56, nesta etapa uma senha de root deve ser escolhida para o servidor e de acordo com a figura 57, o método de autenticação Mysql escolhido para este estudo foi o *Legacy Authentication Method*.

Figura 57 - Escolha do método de autenticação para o Mysql.



Fonte: O autor.

A verificação do *status* do servidor Mysql pode ser feita com o comando da figura 58:

Figura 58 - Verificação do status do servidor Mysql.

```
# systemctl status mysql.service
```

Fonte: O autor.

Se o processo estiver ativo e em execução a porta do serviço 3306/TCP estará em *listen* (escutando). O que pode ser verificado com o comando da figura 59:

Figura 59 - verificação do estado da porta do servidor Mysql.

```
# ss -t -a -n -p | grep 3306
```

Fonte: O autor.

Uma vez que o Mysql está devidamente instalado, deve-se entrar no banco e criar a infraestrutura necessária para simulação da vulnerabilidade SQL *injection* na aplicação PHP. A figura 60 mostra o comando para acessar o servidor Mysql.

Figura 60 - Acesso ao banco como usuário root.

```
$ mysql -u root -p
```

Fonte: O autor.

A figura 61 apresenta os comandos que devem ser executados dentro do console Mysql.

Figura 61 - Comandos Mysql necessários para simulação de ataque SQL *Injection*.

```
mysql> CREATE USER 'morpheus'@'%' IDENTIFIED BY 'morpheus';
mysql> GRANT ALL PRIVILEGES ON *.* TO 'morpheus'@'%';
mysql> CREATE DATABASE cadastro;
mysql> USE cadastro;
mysql> CREATE TABLE usuarios ( id INT PRIMARY KEY NOT NULL
AUTO_INCREMENT, nome VARCHAR(100) NOT NULL, login VARCHAR(100)
NOT NULL, senha VARCHAR(100) NOT NULL, setor VARCHAR(100) NOT NULL,
cargo VARCHAR(100) NOT NULL );
mysql> INSERT INTO usuarios VALUES (1, "felipe", "fsiqueira", "iomoh4pe", "TI",
"Coordenador");
```

```
mysql> INSERT INTO usuarios VALUES (2, "vanderson", "vandsena", "naisae9a",  
"TI", "Programador Pleno");  
mysql> INSERT INTO usuarios VALUES (3, "vinicius", "vcardoso", "hae4eidu",  
"TI", "Programador Junior");  
mysql> INSERT INTO usuarios VALUES (4, "denise", "dsantos", "aezohVe7",  
"ADM", "Contas a pagar");  
mysql> INSERT INTO usuarios VALUES (5, "carla", "csantana", "eiz7Eica", "RH",  
"Auxiliar junior");  
mysql> INSERT INTO usuarios VALUES (6, "braga", "brsilva", "iew4aPhu",  
"Contabilidade", "Contador Pleno");
```

Fonte: O autor.

Os comandos Mysql digitados servem para realizar as seguintes tarefas:

1. Criar o usuario morpheus com a senha morpheus.
2. Estabelecer a permissão para que o usuário morpheus possa acessar qualquer base de dados no Mysql a partir de qualquer *host*.
3. Criar a base de dados “cadastro”. E na sequência entrar na mesma.
4. Criar a tabela usuários com os campos: *id*, nome, *login*, senha, setor e cargo.
5. Adicionar os usuários Felipe, Vanderson, Vinícius, Denise, Carla e Braga, bem como, suas respectivas informações de cadastro.

No diretório raiz web (/var/www/html), o arquivo *conecta.php* deve ser criado com o conteúdo da figura 62 e o arquivo *pesquisa.php* com os códigos da figura 63.

Figura 62 - Arquivo conecta.php.

```

<?php
$servidor = "127.0.0.1";
$usuario_mysql = "morpheus";
$senha_mysql = "morpheus";
$conexao = mysqli_connect($servidor, $usuario_mysql, $senha_mysql, "cadastro");
?>

```

Fonte: Moreno (2017).

Figura 63 - Arquivo pesquisa.php.

```

<!DOCTYPE html>
<html>
<head>
    <title>Cadastro de colaboradores</title>
    <meta charset="utf-8">
</head>
<body>
    <h3>Pesquisa cadastro de colaborador</h3>
    <form action="" method="GET">
        Nome do Usuário: <input type="text" name="nome">
        <input type="submit" value="Enviar">
    </form>
<?php
include "/var/www/html/conecta.php";

if(isset($_GET["nome"])) {
    $nome = $_GET["nome"];

    $query = "SELECT * from usuarios WHERE nome = '$nome' ";

    $dados = mysqli_query($conexao, $query) or die (mysqli_error($conexao));

    while( $linha = mysqli_fetch_assoc($dados) ){
        $nome = $linha["nome"];
        $setor = $linha["setor"];
        $cargo = $linha["cargo"];
        echo "<pre>Nome: {$nome}<br>Setor: {$setor}<br>Cargo: {$cargo}</pre>";
    }
    mysqli_close($conexao);
}

echo "</body>";
echo "</html>";

?>

```

Fonte: Moreno (2017). Adaptado pelo autor.

Ao entrar com o usuário a ser pesquisado no campo do formulário o programa nos mostra suas informações cadastrais.

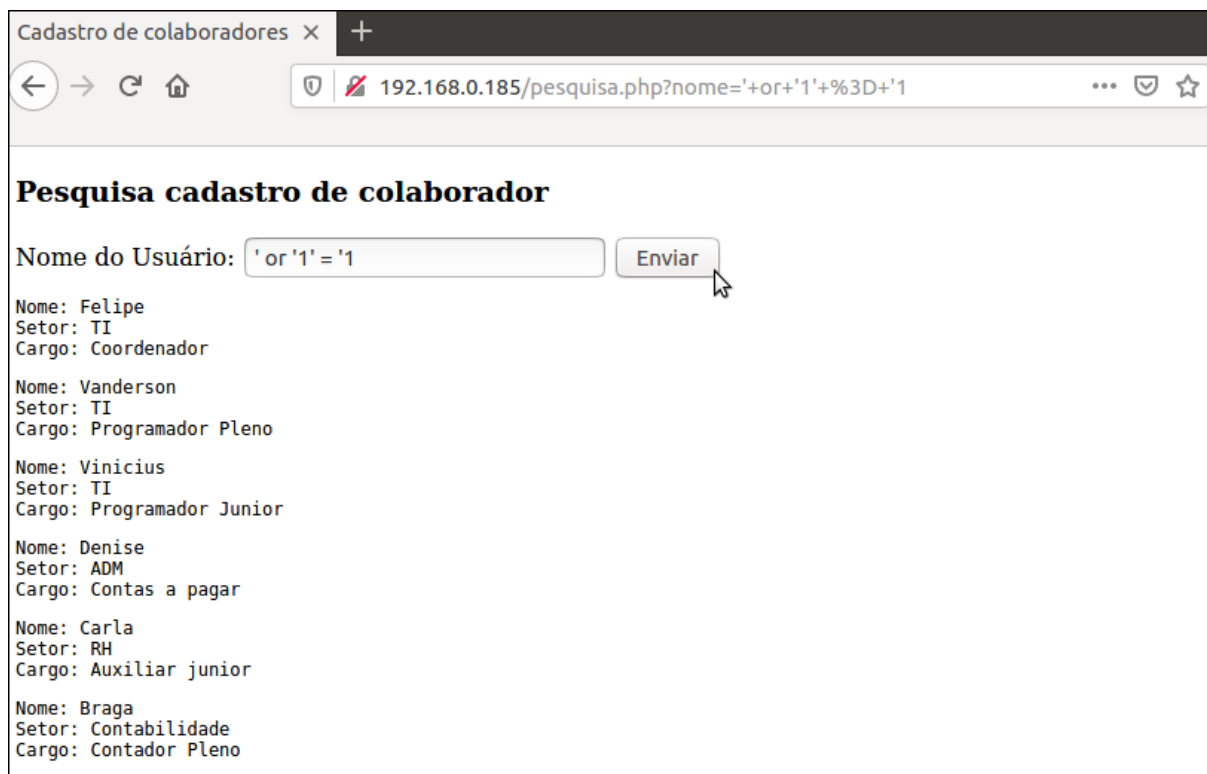
Figura 64 - Programa pesquisa.php sendo executado da maneira esperada.



Fonte: O autor.

Apesar de parecer funcionar corretamente a aplicação *pesquisa.php* está vulnerável a ataques de *SQL injection*. As devidas validações de segurança no campo do formulário (entrada de dados de usuário) e o uso de codificação segura não foi realizado. O que permite que um atacante entre com uma entrada maliciosa e obtenha a saída exibida na figura 65.

Figura 65 - Aplicação pesquisa.php sofrendo um ataque de *SQL injection*.



Fonte: O autor.

A *string* maliciosa `' or '1' = '1'` foi incorporada a consulta Mysql do código da aplicação exibido na figura 66.

Figura 66 - *Query* (consulta) SQL que mostra informações sobre o usuário.

```
$query = "SELECT * from usuarios WHERE nome = '$nome' ";
```

Fonte: O autor.

E transformou-se na *query* apresentada na figura 67:

Figura 67 - *String* maliciosa construída pela invasor.

```
$query = "SELECT * from usuarios WHERE nome = '$nome' or '1' = '1' ";
```

Fonte: O autor.

Isto invalidou o filtro determinado pela condição *WHERE* que estabelecia que fossem exibidas informações somente sobre o nome do usuário especificado no formulário.

Na condição pós *string* maliciosa o *SELECT* mostra todos os usuários, pois a instrução do operador lógico **or '1' = '1'** sempre será verdadeira.

A função PHP ***htmlspecialchars*** pode ser usada para corrigir a falha no código. Basta substituir o trecho de código da figura 68, pelo código da figura 69.

Figura 68 - Trecho de código vulnerável.

```
$nome = $_GET["nome"];
```

Fonte: O autor.

Figura 69 - A função `htmlspecialchars` sendo utilizada para proteger contra o *SQL injection*.

```
$user = $_GET["nome"];  
$nome = htmlspecialchars($user, ENT_QUOTES, 'UTF-8')
```

Fonte: O autor.

2.7.4 Hardening no Mysql

Uma série de configurações de segurança para o servidor Mysql pode ser realizada com o comando *mysql_secure_installation*, que incluem: estabelecimento de uma política de senhas segura para os usuários, remoção de usuários anônimos, desativação de *login* remoto para o usuário root e exclusão da base de dados *test*.

Figura 70 - Execução do comando *mysql_secure_installation*.

```

root@buster:~# mysql_secure_installation

Securing the MySQL server deployment.

Enter password for user root:

VALIDATE PASSWORD COMPONENT can be used to test passwords
and improve security. It checks the strength of password
and allows the users to set only those passwords which are
secure enough. Would you like to setup VALIDATE PASSWORD component?

Press y|Y for Yes, any other key for No: y

There are three levels of password validation policy:

LOW      Length >= 8
MEDIUM  Length >= 8, numeric, mixed case, and special characters
STRONG Length >= 8, numeric, mixed case, special characters and dictionary

Please enter 0 = LOW, 1 = MEDIUM and 2 = STRONG: 1
Using existing password for root.

Estimated strength of the password: 50
Change the password for root ? ((Press y|Y for Yes, any other key for No) : y

New password:

Re-enter new password:

Estimated strength of the password: 100
Do you wish to continue with the password provided?(Press y|Y for Yes, any other key for No) : y
By default, a MySQL installation has an anonymous user,
allowing anyone to log into MySQL without having to have
a user account created for them. This is intended only for
testing, and to make the installation go a bit smoother.
You should remove them before moving into a production
environment.

Remove anonymous users? (Press y|Y for Yes, any other key for No) : y

```

Fonte: O autor.

Figura 71 - mysql_secure_installation desativando o *login* remoto para o usuário root.

```
Normally, root should only be allowed to connect from
'localhost'. This ensures that someone cannot guess at
the root password from the network.

Disallow root login remotely? (Press y|Y for Yes, any other key for No) : y
Success.

By default, MySQL comes with a database named 'test' that
anyone can access. This is also intended only for testing,
and should be removed before moving into a production
environment.

Remove test database and access to it? (Press y|Y for Yes, any other key for No) : y
- Dropping test database...
Success.

- Removing privileges on test database...
Success.

Reloading the privilege tables will ensure that all changes
made so far will take effect immediately.

Reload privilege tables now? (Press y|Y for Yes, any other key for No) : y
Success.

All done!
```

Fonte: O autor.

Princípio do menor privilégio

Segundo Owasp Cheat Series (2020), As permissões atribuídas aos usuários do banco devem seguir o princípio do menor privilégio, ou seja, os usuários devem possuir apenas as permissões necessárias para realizar o trabalho na aplicação. Por exemplo:

Uma empresa possui um banco de dados **estoque** que contém uma tabela chamada **produtos**. Na tabela produtos temos os seguintes campos: id_produto, nome, fabricante e valor.

Na política de permissões de usuários foi definido que o usuário Carlos deve ter acesso total na base de dados estoque e que o usuário Bóris deve possuir apenas acesso para *SELECT* na tabela produtos da base de dados estoque.

Um cenário pode ser construído para demonstrar o funcionamento do exemplo, conforme mostra a figura 72.

Figura 72 - Estrutura para demonstração de esquema de permissões.

```
mysql> CREATE DATABASE estoque;  
  
mysql> use estoque;  
  
mysql> CREATE TABLE produtos ( id_produto INT PRIMARY KEY NOT NULL  
AUTO_INCREMENT, nome VARCHAR(50) NOT NULL, fabricante VARCHAR(50)  
NOT NULL, valor DEC(6,2) NOT NULL );  
  
mysql> INSERT INTO produtos VALUES (1, "Moho de tomate", "Quero", "4.10");  
  
mysql> INSERT INTO produtos VALUES (2, "Maionese", "Kraft", "5.80");  
  
mysql> INSERT INTO produtos VALUES (3, "Atum Sólido", "Coqueiro", "7.75");
```

Fonte: O autor.

Depois da criação da base de dados estoque, da tabela produtos e da inserção de alguns dados na mesma, a criação dos usuários e suas respectivas permissões podem ser atribuídas com os comandos da figura 73. As senhas atribuídas aos usuários devem seguir as diretrizes da política de senhas definida durante a execução do comando *mysql_secure_installation*.

Figura 73 - Criação dos usuários e definição das permissões de acesso com o comando GRANT.

```
mysql> CREATE USER 'carlos'@'localhost' IDENTIFIED BY 'senha';

mysql> CREATE USER 'boris'@'localhost' IDENTIFIED BY 'senha';

mysql> GRANT ALL ON estoque.* TO 'carlos'@'localhost';

mysql> GRANT SELECT ON estoque.produtos TO 'boris'@'localhost';

mysql> FLUSH PRIVILEGES;
```

Fonte: O autor.

Depois da atribuição de permissões o usuário Boris realiza um SELECT na tabela produtos, entretanto, tem o acesso negado para realizar um UPDATE, conforme foi definido na diretiva GRANT da figura 73.

Figura 74 - Usuário Boris com acesso negado para UPDATE na tabela produtos.

```
mysql> SELECT(CURRENT_USER);
+-----+
| (CURRENT_USER) |
+-----+
| boris@localhost |
+-----+
1 row in set (0.00 sec)

mysql> use estoque;
Database changed
mysql> show tables;
+-----+
| Tables_in_estoque |
+-----+
| produtos           |
+-----+
1 row in set (0.00 sec)

mysql> select * from produtos;
+-----+-----+-----+-----+
| id_produto | nome           | fabricante | valor |
+-----+-----+-----+-----+
| 1          | Moho de tomate | Quero     | 4.10  |
| 2          | Maionese       | Kraft     | 5.80  |
| 3          | Atum Sólido    | Coqueiro  | 7.75  |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> UPDATE produtos SET valor = 20.50 WHERE nome = 'Maionese';
ERROR 1142 (42000): UPDATE command denied to user 'boris'@'localhost' for table 'produtos'
mysql>
```

Fonte: O autor.

Armazenamento das senhas dos usuários

As senhas dos usuários nunca devem ser armazenadas no formato “*Clear text*” nas tabelas das bases de dados, pois desta maneira, tanto os administradores quanto pessoas que obterem acesso não autorizado poderão ver as senhas de todas as contas da aplicação. A figura 75 mostra o administrador do banco visualizando a senha de todos os usuários da aplicação.

Figura 75 - Senhas dos usuários armazenadas de forma insegura.

```
mysql> show tables;
+-----+
| Tables_in_cadastro |
+-----+
| usuarios            |
+-----+
1 row in set (0.00 sec)

mysql> select * from usuarios;
+----+-----+-----+-----+-----+-----+
| id | nome   | login  | senha  | setor  | cargo      |
+----+-----+-----+-----+-----+-----+
| 1  | felipe | fsiqueira | iomoh4pe | TI      | Coordenador |
| 2  | vanderson | vandsena | naisae9a | TI      | Programador Pleno |
| 3  | vinicius | vcardoso | hae4eidu | TI      | Programador Junior |
| 4  | denise  | dsantos  | aezohVe7 | ADM     | Contas a pagar |
| 5  | carla   | csantana | eiz7Eica | RH      | Auxiliar junior |
| 6  | braga   | brsilva  | iew4aPhu | Contabilidade | Contador Pleno |
+----+-----+-----+-----+-----+-----+
6 rows in set (0.00 sec)

mysql> █
```

Fonte: O autor.

A fim de aumentar a segurança deve-se fazer um *Hash* da senha dos usuários antes que a mesma seja armazenada no banco de dados.

Segundo Miletto e Bertagnolli (2014), o *Hash* consiste de um resumo matemático da senha. Os caracteres de entrada são transformados em uma *string* de tamanho fixo que não pode ser revertida. *MD5*, *SHA1*, *Bcrypt* e *Argon2* são exemplos de algoritmos de *hashing*.

Uma determinada *string* sempre irá gerar o mesmo *Hash*, portanto, uma *string* aleatória chamada *Salt* deve ser acrescentada a *string* original para que sejam

gerados diferentes *hashes* quando a mesma sequência de caracteres de entrada for utilizada.

No PHP a função *password_hash* pode ser usada para gerar um *Hash* das senhas dos usuários. Por padrão, a função usa o algoritmo *Bcrypt*. A Owasp (2017), recomenda o uso de funções de hashing como *Argon2*, *scrypt*, *bcrypt* ou *PBKDF2* com recursos de *Salt*.

Figura 76 - Senhas armazenadas de forma segura com uso do algoritmo de *Hash Bcrypt*.

```
mysql> select nome,senha from usuarios;
+-----+-----+
| nome | senha |
+-----+-----+
| felipe | $2y$10$yigUd1l2dn.VQNCGOaH8m.hQp5oar7cWy8jC0hGFN6BREuItgHeRW |
| vanderson | $2y$10$.QXdaXF5DJ6C0VgNuxQeF.Fb0J5TXufuPZreiYWKA4tbyP7PaVoEG |
| vinicius | $2y$10$vGcvHN5JzSjr0snZL/wvM.L./eHjjGG28DBdAWG6mj8ZZ7tcbqy6e |
| denise | $2y$10$j/1NJxmTxbfpm50Jn7TgK0FhhACpuH4MEfaHKI0QptwXeFhhnxLe |
| carla | $2y$10$s0bcfrs/WfoeJcZI0fH/R.ZmIIJsobWx9mZPs3pm5xEDUVbYXoy56 |
| braga | $2y$10$8K/AcPCrwdLhBSp2yKBpLOA0fDOTKSMUFRPiNiES0z3dJxMKEYDPm |
+-----+-----+
6 rows in set (0.00 sec)
```

Fonte: O autor.

Disponibilidade do banco

Backups regulares devem ser programados para garantir a disponibilidade das bases de dados do servidor. Também é recomendável que sejam atribuídas permissões de acesso somente para os usuários que forem manipular estes *backups* e que o mesmo seja armazenado de forma criptografada (OWASP CHEAT SERIES, 2020).

O *backup* de uma base de dados pode ser realizado com o comando *mysqldump*, e para restauração pode-se utilizar o comando *mysql*.

Figura 77 - *Backup* de uma base de dados com *mysqldump*.

```
mysql> mysqldump -v -u root -p --databases nome_da_base > arquivo-de-backup.sql
```

Fonte: O autor.

Figura 78 - Restauração de uma base de dados com o comando *mysql*.

```
mysql> mysql -v -u root -p -D nome_da_base < arquivo-de-backup.sql
```

Fonte: O autor.

2.7.5 Cross-site scripting (XSS)

Qualquer código *JavaScript* presente em um *site web* pode ser executado a partir de um navegador. Assim sendo, um ataque *cross-site scripting* pode ocorrer quando um invasor encontra uma maneira de introduzir código *JavaScript* malicioso no navegador de um usuário, enquanto o mesmo visualiza um determinado *site*.

O *JavaScript* pode ler e modificar qualquer parte de um *site*. Informações sensíveis como credenciais de *login* e números de cartão de crédito podem ser capturadas enquanto o usuário as digita. Se o invasor for capaz de obter informações da sessão HTTP, poderá sequestrar a sessão de um usuário e utilizar a sua conta para fazer *login* remotamente (MCDONALD, 2020).

Shiflett (2005), descreve que o *cross-site scripting* é um dos tipos mais conhecidos de ataques, afeta as aplicações *web* baseadas em PHP e em todas as outras linguagens.

A maioria dos aplicações trabalham com entradas de dados como: formulários para pesquisa, autenticação de usuários, fóruns e *blogs*. Quando as mesmas, não realizam a filtragem e a tratativa dos dados de entrada de usuário, ficam vulneráveis a este tipo de ataque.

Os códigos HTML e PHP das figuras 79 e 80 demonstram como os ataques *cross-site scripting* (também conhecidos como XSS) podem ser executados em entradas de usuário sem o tratamento de segurança adequado. A figura 79 apresenta um formulário HTML capturando nome de usuário e senha através do método *HTTP GET*.

Figura 79 - Código HTML com formulário usando o método HTTP GET.

```
<!DOCTYPE html>
<html>
<head>
  <title>Estudo XSS</title>
  <meta charset="utf-8">
</head>
<body>
  <form action="teste_risco.php" method="GET" />
  <p>Usuario: <input type="text" name="usuario" /> <br /></p>
  <p>Senha: <input type="password" name="senha" /> <br /></p>
  <p><input type="submit" /></p>
</form>
</body>
</html>
```

Fonte: Mileto e Bertagnolli (2014). Adaptado pelo autor.

Ao ser acionado, o botão *submit* envia estes dados para serem processados pelo arquivo *teste_risco.php*, que tem seu código exibido na figura 80.

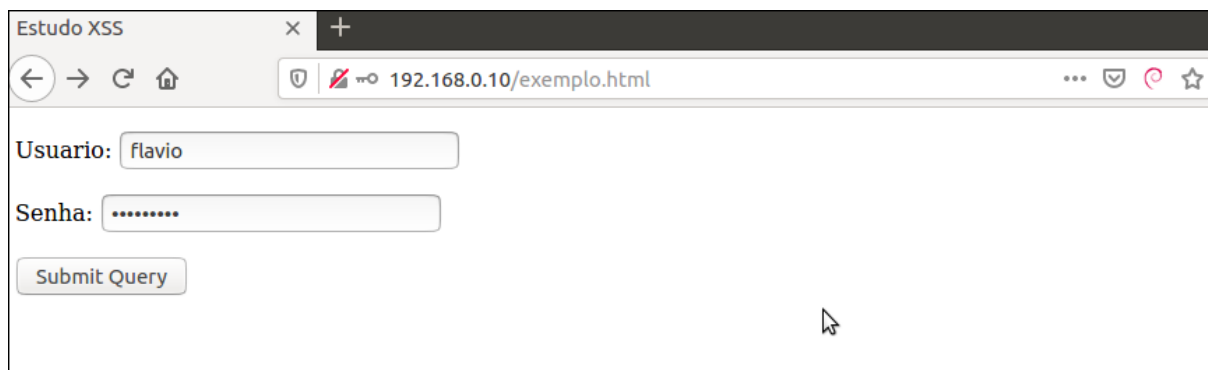
Figura 80 - Arquivo teste_risco.php.

```
<?php
  $usuario=$_GET['usuario'];
  $senha=$_GET['senha'];
  echo "<p> usuario: $usuario</p>";
  echo "<p> senha: $senha</p>";
?>
```

Mileto e Bertagnolli (2014). Adaptado pelo autor.

O arquivo *teste_risco.php*, por sua vez, captura os dados de usuário e senha inseridos no formulário e os exibe na tela, conforme mostram as figuras 81 e 82.

Figura 81 - Entrada de dados no formulário.



A screenshot of a web browser window. The title bar shows 'Estudo XSS'. The address bar contains '192.168.0.10/exemplo.html'. The page content includes a form with two input fields: 'Usuario:' with the text 'flavio' and 'Senha:' with masked characters '.....'. Below the fields is a button labeled 'Submit Query'.

Fonte: O autor.

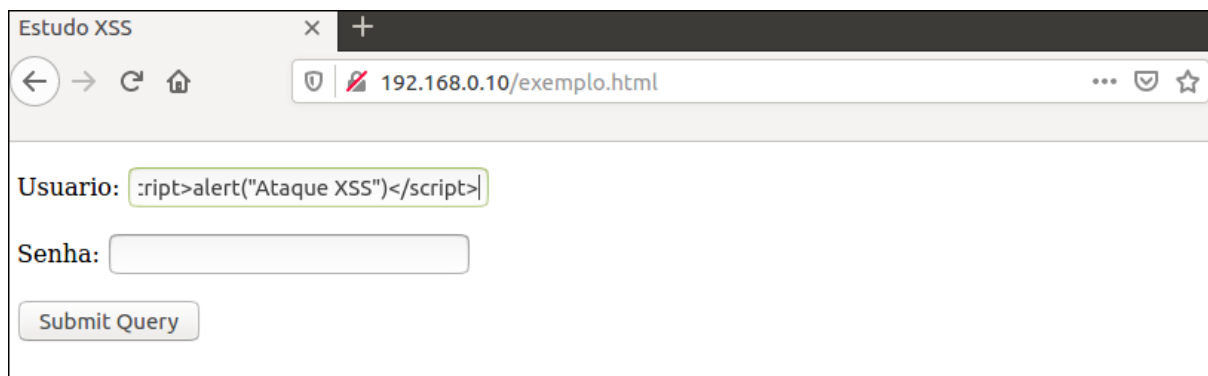
Figura 82 - Dados processados e exibidos pelo arquivo teste_risco.php.



A screenshot of a web browser window. The title bar shows '192.168.0.10/teste_risco.php'. The address bar contains '192.168.0.10/teste_risco.php?usuario=flavio&senha=Falc09%23GT'. The page content displays the processed data: 'usuario: flavio' and 'senha: Falc09#GT'.

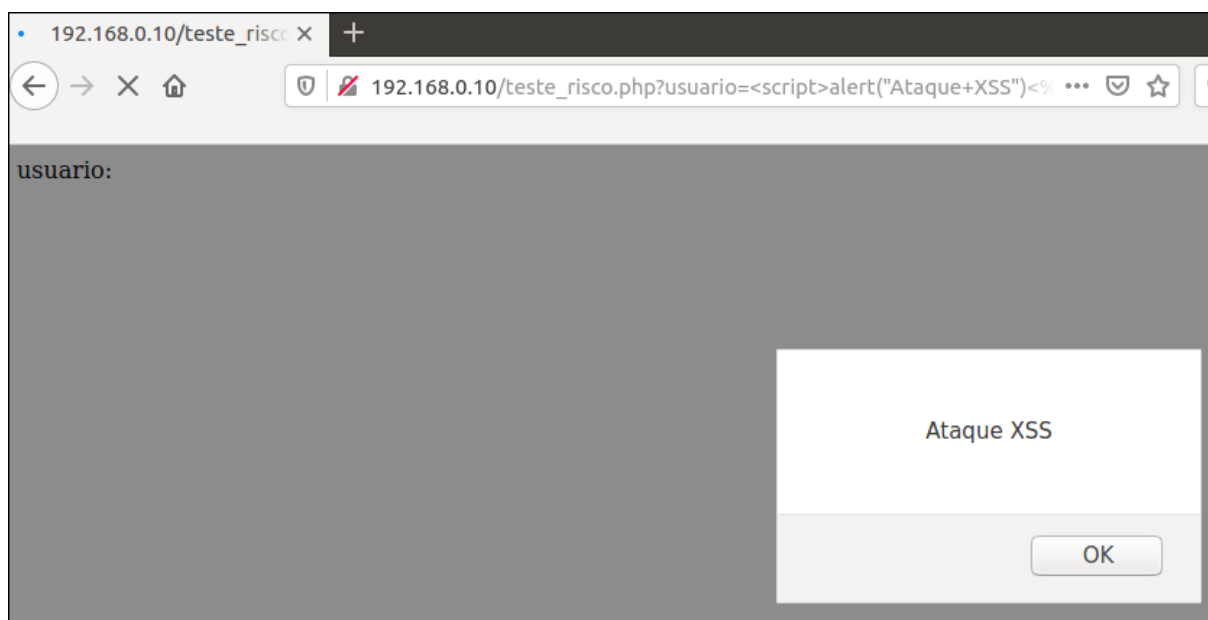
Fonte: O autor.

Contudo, um código JavaScript malicioso também será interpretado e executado pelo navegador. Ao inserir o código **<script>alert("Ataque XSS")</script>** no campo Usuário do formulário, temos a saída exibida na figura 84.

Figura 83 - Injeção de código *JavaScript* malicioso no formulário.

Fonte: O autor.

Figura 84 - Sucesso na execução do ataque XSS.



Fonte: O autor.

A função ***htmlspecialchars*** do PHP pode ser usada para corrigir a vulnerabilidade de XSS neste código. A figura 85 apresenta a aplicação fazendo uso da função.

Figura 85 - Arquivo teste_risco.php com função htmlentities.

```
<?php
    $usr=$_GET['usuario'];
    $usuario=htmlentities($usr);
    $pass=$_GET['senha'];
    $senha=htmlentities($pass);
    echo "<p> usuario: $usuario</p>";
    echo "<p> senha: $senha</p>";
?>
```

Fonte: Mileto e Bertagnolli (2014). Adaptado pelo autor.

3 METODOLOGIA

Neste capítulo será apresentada a metodologia utilizada para desenvolvimento deste estudo. Podemos dividi-la em quatro partes: método, tipo de pesquisa, coleta e tratamento de dados.

3.1 Método

O método utilizado foi a pesquisa bibliográfica. Segundo Martins e Lintz (2010, p.15), “A pesquisa bibliográfica procura explicar e discutir um tema ou um problema com base em referências teóricas publicadas em livros, revistas, periódicos, etc”. Seu objetivo é buscar e analisar as contribuições de cunho científico sobre um determinado tema.

Como uma de suas principais vantagens a pesquisa bibliográfica pode proporcionar ao investigador a cobertura de uma gama muito maior de informações do que ele poderia obter pesquisando diretamente (GIL, 2010).

3.2 Tipo de pesquisa

Como abordagem foi adotada o tipo qualitativa, pois diferentemente das pesquisas quantitativas que utilizam a linguagem matemática para identificar e mensurar dados a abordagem qualitativa tem um foco maior na subjetividade.

Os aspectos essenciais da pesquisa qualitativa consistem na escolha adequada de métodos e teorias convenientes; no reconhecimento e na análise de diferentes perspectivas; nas reflexões dos pesquisadores a respeito de suas pesquisas como parte do processo de produção do conhecimento e na variedade de abordagens e métodos. (FLICK, 2009, p. 23).

3.3 Coleta de dados

Como técnica para coleta de dados foi utilizada a análise bibliográfica.

Desta maneira, referências teóricas publicadas por especialistas nas áreas de segurança de aplicações *web*, Linux, segurança de redes, TCP/IP, servidores *web*, PHP e bancos de dados Mysql foram utilizadas. A pesquisa bibliográfica é construída com base em material já publicado, sendo que, o mesmo pode ser impresso como livros, jornais, revistas, etc e também pode ser obtido no formato digital através de discos, cds, fitas magnéticas, bases de dados e sistemas de busca disponibilizados na Internet (GIL, 2010).

3.4 Tratamento de dados

Para tratamento dos dados foi utilizada a análise de conteúdo. A análise de conteúdo é uma metodologia empregada para descrição e interpretação de conteúdos de todas as classes de documentos e textos. Aliada a descrições sistemáticas, quantitativas ou qualitativas esta análise pode contribuir para que se atinja uma melhor compreensão das mensagens e dos seus significados (MORAIS, 1999).

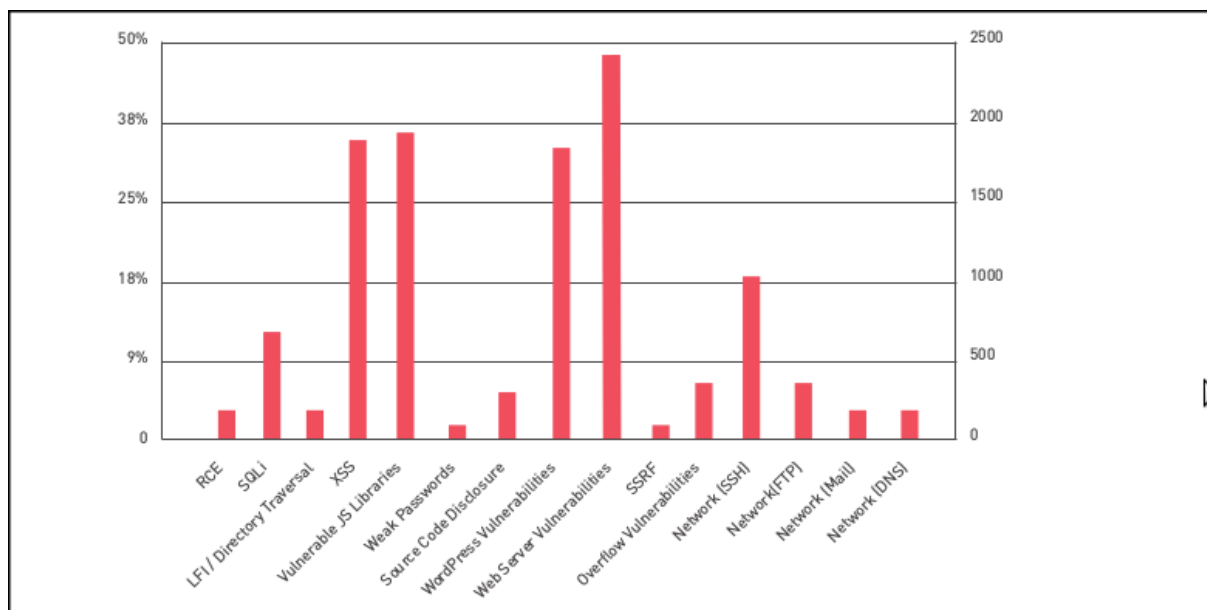
4 ANÁLISE DOS DADOS

4.1 Tema – *Cross-site Scripting (XSS)*

De acordo com a Owasp (2017), que enumera os riscos mais críticos para as aplicações *web* em geral, as falhas relacionadas a XSS atingem dois terços destas aplicações e representam o segundo maior risco dentro da classificação deste documento (OWASP, 2017).

A empresa Acunetix, uma das referências globais em segurança na *web*, divulga anualmente um relatório sobre a segurança das aplicações *web*. A amostragem do ano de 2019 foi realizada durante um período de 12 meses em 10.000 objetos de verificação. Nesta análise observou-se que cerca de 30% dos objetos verificados possuíam vulnerabilidades relacionadas a XSS, bibliotecas *JavaScript* e no software CMS (*content management system*) Wordpress (ACUNETIX, 2020). A figura 86, exibe as vulnerabilidades que foram categorizadas como sendo de alta gravidade pelo relatório Acunetix de 2019, onde as falhas de XSS alcançam a terceira posição com o maior número de ocorrências.

Figura 86 - Vulnerabilidades *web* mais graves de acordo com relatório Acunetix 2019.



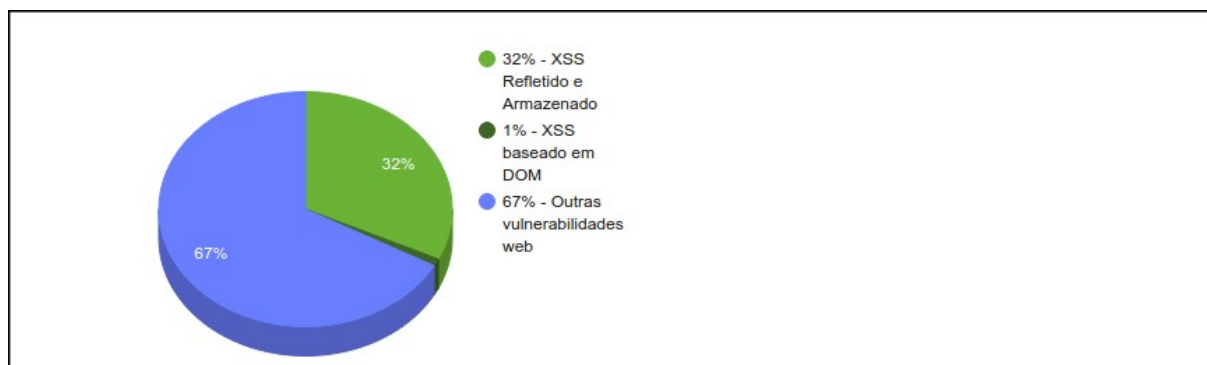
Fonte: Acunetix (2020).

Ainda segundo o relatório Acunetix 2019, 33% das aplicações verificadas possuíam falhas de segurança com relação a XSS, como mostra a figura 87.

Portanto, o conhecimento de métodos para mitigar este tipo de ameaça é fundamental para todas as pessoas que estão envolvidas com a gerência e administração de aplicações *web*.

A análise de dados deste estudo apresenta contramedidas de três grandes referências em segurança *web* para diminuir o risco e aumentar a segurança das aplicações *web* baseadas em LAMP contra ataques de XSS.

Figura 87 - Falhas de segurança XSS em aplicações *web* de acordo com relatório Acunetix 2019.



Fonte: Acunetix (2020). Adaptado pelo autor.

As aplicações LAMP utilizam a linguagem PHP, logo, grande parte das correções técnicas propostas pelos autores consideram o uso de funções e outros recursos desta linguagem. A tabela 9 mostra as contramedidas recomendadas.

Tabela 9 - Autores e correções propostas para ataques XSS.

Autores	Contramedidas
Moreno (2017)	Uso da função <i>htmlentities</i> do PHP para tratamento e filtragem das entradas de usuário.
Miletto e Bertagnolli (2014)	Fazer uso da função <i>htmlspecialchars</i> do PHP.
Mcdonald (2020)	<p>1 - Fazer a substituição dos caracteres de controle HTML pela codificação da entidade correspondente. Isto é necessário para que o navegador do usuário não interprete caracteres como < e >, por exemplo.</p> <p>2 – Utilização de <i>Content Security Policies</i>. Os navegadores mais recentes permitem que os sites definam políticas de segurança de conteúdo, que podem ser usadas para bloquear a execução de código <i>JavaScript</i>.</p> <p>3 - Evitar qualquer coisa retirada de um fragmento de URI (<i>Uniform Resource Identifier</i>) antes de interpolar este conteúdo em HTML com o código do lado do navegador.</p>

Fonte: Própria.

4.2 Considerações sobre a análise de dados

Moreno (2017) e Miletto e Bertagnolli (2014) recomendam a utilização de funções do PHP para tratamento das entradas maliciosas de usuários. Já McDonald (2020) é mais abrangente e oferece mais técnicas que podem ser usadas para proteção contra os ataques XSS. Apesar das funções PHP se mostrarem eficientes para tratamento dos dados maliciosos o uso de múltiplas estratégias para correção de uma vulnerabilidade proporcionam camadas de segurança adicionais, portanto, são mais eficazes, além de estarem alinhadas com o conceito de *defesa em profundidade*, que recomenda o uso de várias camadas de segurança.

5 CONSIDERAÇÕES FINAIS

Toda aplicação *web* possui falhas e portanto, deixá-las mais seguras compreende um amplo conjunto de processos que devem ser executados, entre eles: gerenciamento adequado da infraestrutura, práticas de codificação segura, controle de acesso de usuários, atualizações de segurança, desativação de componentes vulneráveis e desnecessários, análise de *logs*, monitorações da disponibilidade dos serviços, instalação de sistemas de detecção de intrusão, controles criptográficos, detecção de *malwares* e implementação de *firewalls*.

Descuidar da segurança de apenas um componente que faz parte da pilha LAMP pode expor a aplicação e os respectivos serviços de rede à diversas vulnerabilidades que poderão ser exploradas em eventuais ataques.

Todo conjunto atual de tecnologias e metodologias recentes que compreendem: *Cloud Computing*, *Internet of things*, inteligência artificial, indústria 4.0, *Mobile* e *Devops*, que por sua vez, propõe a integração das equipes de desenvolvimento e operações com práticas de infraestrutura como código constituem o novo território onde a segurança das aplicações *web* baseadas em LAMP terão que se manter.

Novos trabalhos envolvendo a segurança das aplicações LAMP poderão ser desenvolvidos dentro do contexto *Devops*, com a utilização de ferramentas para virtualização de ambientes, *containers* e serviços sendo executados em ambientes *Cloud Computing*.

6 REFERÊNCIAS

ACUNETIX (org.). **Acunetix Web Application Vulnerability Report 2019**.

Disponível em: <<https://www.acunetix.com/acunetix-web-application-vulnerability-report-2019>>. Acesso em: 18 maio 2020.

ACUNETIX SECURITY DIRECTORY TRAVERSAL & CODE INJECTION. **PHP Security 2: Directory Traversal & Code Injection**. Disponível em:

<<https://www.acunetix.com/websitesecurity/php-security-2/>>. Acesso em: 16 ago. 2020.

APACHE DOCS AUTH. **Authentication and Authorization**. Disponível em:

<<http://httpd.apache.org/docs/2.4/howto/auth.html>>. Acesso em: 06 maio 2020.

APACHE DOCS SSL/TLS. **SSL/TLS Strong Encryption**. Disponível em:

<https://httpd.apache.org/docs/2.4/ssl/ssl_intro.html>. Acesso em: 25 jul. 2020.

APACHE SOFTWARE FOUNDATION. **Apache HTTP Server Project**. 2019.

Disponível em: <<http://httpd.apache.org/>>. Acesso em: 04 jun. 2019.

ARORA, Namita et al. Linux Hardening. **International Journal On Recent And Innovation Trends In Computing And Communication**. Índia, p. 1019-1022. maio

2014. Disponível em: <<https://ijritcc.org/index.>>. Acesso em: 20 out. 2019.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR ISO/IEC 27002:2013**: Tecnologia da informação - Técnicas de segurança - código de prática para controles de segurança da informação. Rio de Janeiro: Abnt, 2013. 99 p.

BASSO, Tania. **Uma abordagem para avaliação da eficácia de scanners de vulnerabilidades em aplicações web**. 2010. 120 f. Dissertação (Mestrado) - Curso

de Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, Sp, 2010.

CLOUDFLARE. **What is SSL?** Disponível em: <<https://www.cloudflare.com/learning/ssl/what-is-ssl/>>. Acesso em: 27 jul. 2020.

FERNÁNDEZ SANGUINO PEÑA, Javier ; REELSEN, Alexander . **Securing Debian Manual**. 2013. Disponível em: <<https://www.debian.org/doc/>>. Acesso em: 10 out. 2019.

FLICK, Uwe. **Introdução à pesquisa qualitativa**. 3. ed. Porto Alegre: Artmed Bookman, 2009.

GHEORGHE, Lucian. **Designing and Implementing Linux Firewalls and QoS using netfilter, iproute2, NAT, and L7-filter**: Learn how to secure your system and implement QoS using real-world scenarios for network of all sizes. Olton Birmingham: Packt Publishing, 2006.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa**. 5. ed. São Paulo: Atlas, 2010.

KISSEL, Joe. **Aprendendo a proteger suas senhas**. São Paulo: Novatec, 2017.

KUROSE, James F; ROSS, Keith W. **Redes de Computadores e a Internet**: Uma Nova Abordagem. São Paulo: Addison Wesley, 2003.

LET'S ENCRYPT. **Começando a usar**. Disponível em: <<https://letsencrypt.org/pt-br/getting-started/>>. Acesso em: 26 jul. 2020.

MARTINS, Gilberto de Andrade; LINTZ, Alexandre. **Guia para elaboração de monografias e trabalhos de conclusão de curso**. 2. ed. São Paulo: Editora Atlas, 2010.

MCDONALD, Malcolm. **Web security for developers**. [s. L.]: no Start Press, 2020. 172 p.

MILANI, André. **Construindo Aplicações Web com PHP e Mysql**. São Paulo: Novatec, 2010.

MILETTO, Evandro Manara; BERTAGNOLLI, Silvia de Castro. **Desenvolvimento de software II**: Introdução ao desenvolvimento web com html, css, javascript e php. Porto Alegre: Bookman, 2014.

MORAIS, Roque. Análise de conteúdo. **Revista Educação**, Porto Alegre, v. 22, n. 37, p.7-32, 1999.

MORENO, Daniel. **Pentest em aplicações web**. São Paulo: Novatec, 2017. 480 p.

MYSQL, Homepage. **MySQL 8.0 Reference Manual**. Disponível em: <<https://dev.mysql.com/doc/>>. Acesso em: 23 out. 2019.

NEMETH, Evi; SNYDER, Garth; HEIN, Trent R.. **Manual Completo do LINUX**: Guia do administrador. São Paulo: Pearson Makron Books, 2004.

NETCRAFT (Org.). **April 2019 Web Server Survey**. 2019. Disponível em: <<https://news.netcraft.com/>>. Acesso em: 02 jun. 2019.

NIST - NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **800-41**: Guidelines on Firewalls and Firewall Policy. Gaithersburg: Nist, 2009. 48 p.

OWASP CHEAT SERIES. **OWASP Cheat Sheet Series**: database security cheat sheet. Database Security Cheat Sheet. 2020. Disponível em: <https://cheatsheetseries.owasp.org/cheatsheets/Database_Security_Cheat_Sheet.html#permissions>. Acesso em: 15 ago. 2020.

OWASP, Comunidade . **OWASP Top 10 – 2017**. 2017. Disponível em: <<https://www.owasp.org/images/>>. Acesso em: 02 out. 2019.

PAULI, Josh. **Introdução ao web hacking**: Ferramentas e técnicas para invasão de aplicações web. São Paulo: Novatec, 2014.

PHP, Comunidade. **O que o PHP pode fazer?** Disponível em: <<https://www.php.net/manual/>>. Acesso em: 20 out. 2019.

SHIFLETT, Chris. **Essential php security**. Sebastopol: O'reilly, 2005.

STEVENS, W Richard; FENNER, Bill; RUDOFF, Andrew M. **Programação de rede Unix**: API para soquetes de rede. 3. ed. Porto Alegre: Bookman, 2005.

TANENBAUM, Andrew s. **Sistemas Operacionais Modernos**. 3. ed. São Paulo: Pearson, 2010.

THE NETFILTER.ORG PROJECT. **What is netfilter.org?** Disponível em: <<https://www.netfilter.org/index.html>>. Acesso em: 26 jun. 2020.