UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

FELIPE GUIMARÃES PACHECO

ANÁLISE DAS TÉCNICAS DE SEGURANÇA DO FRAMEWORK LARAVEL CONTRA ATAQUES AS APLICAÇÕES WEB

GARANHUNS

2019

Felipe Guimarães Pacheco

ANÁLISE DAS TÉCNICAS DE SEGURANÇA DO FRAMEWORK LARAVEL CONTRA ATAQUES AS APLICAÇÕES WEB

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco.

Orientador:

Prof. Dr. Igor Medeiros Vanderlei

GARANHUNS

2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas da UFRPE
Biblioteca Ariano Suassuna, Garanhuns-PE, Brasil

P116a Pacheco, Felipe Guimarães
Análise das técnicas de segurança do *framework laravel* contra ataques as aplicações web / Felipe Guimarães Pacheco. - 2019.
94 f. ; il.

Orientador: Igor Medeiros Vanderlei.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação)-Universidade Federal Rural de Pernambuco, Departamento de Ciência da Computação, Garanhuns, BR-PE, 2019.
Inclui referências.

1. Segurança da informação 2. Framework (Arquivo de Computador) 3. Web Semântica 4. Computação I. Vanderlei, Igor Medeiros, orient. II. Título

CDD 004

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação da Unidade Acadêmica de Garanhuns da Universidade Federal Rural de Pernambuco, aprovada pela comissão examinadora que abaixo assina.

Prof. Dr. Igor Medeiros Vanderlei - Orientador
UAG
UFRPE

Prof. Dr. Jean Carlos Teixeira de Araujo - Examinador
UAG
UFRPE

Prof. MSc. Sérgio Francisco Tavares de Oliveira Mendonça - Examinador
UAG
UFRPE

GARANHUNS

2019

Resumo

As aplicações *web* estão cada vez mais presentes em nosso cotidiano, nos auxiliando em diversas tarefas, como por exemplo, transações bancárias, sites de compras, sistemas de empresas, redes sociais, entre outros. Boa parte dessas aplicações armazenam e processam dados fornecidos pelo cliente, onde os mesmos são, muitas vezes, sigilosos e necessitam de proteção. Para agilizar o processo de entrega do desenvolvimento das aplicações, muitos desenvolvedores optam por fazer o uso de *frameworks* na criação das funcionalidades dos seus projetos. Com isso, a segurança da informação acaba não sendo priorizada, podendo causar problemas futuros, como ataques às aplicações *web*, por exemplo. Os ataques as aplicações *web* é algo corriqueiro entre *crackers*, tendo em vista que estas aplicações não estão limitadas apenas a ataques internos de uma organização, mas a qualquer usuário que acesse a internet. Diante disso foi desenvolvido este trabalho que aborda uma pesquisa de carater exploratório e que visa apresentar as técnicas de segurança integradas ao *framework* Laravel, analisando cada vulnerabilidade através da realização de testes contra cada uma utilizando ferramentas de penetração e verificação dos recursos presentes no *framework*. Foi observado como ataques alvo, as dez vulnerabilidades listadas pelo OWASP Top Ten, um dos projetos para segurança de dados desenvolvida pela OWASP (*Open Web Application Security Project*) e tem como finalidade apresentar as dez vulnerabilidades das aplicações *web* que podem causar mais danos a sistemas na internet. Por meio dos experimentos realizados é possivel ilustrar para quais vulnerabilidaes o Laravel tem mecanismos de defesa e como os desenvolvedores podem utilizá-los para proteção dos dados do usuário e contra quais vulnerabilidades o *framework* não apresenta recursos contra ataques externos.

Palavras-chave: Segurança da informação. Laravel. OWASP

Abstract

Web applications are increasingly present in our daily lives, helping us in various tasks, such as banking transactions, shopping sites, business systems, social networks, among others. A good part of these applications store and process data provided by the customer, where they are often confidential and need protection. To streamline the application development delivery process, many developers choose to make use of frameworks in designing the features of their projects. As a result, information security is not prioritized, and may cause future problems, such as attacks on web applications. Attacks on web applications are commonplace among crackers, given that these applications are not limited to an organization's internal attacks alone, but to any user who access the internet. In the light of this, this work was developed that addresses an exploratory research aiming to present the integrated security techniques to the Laravel framework, analyzing each vulnerability by performing con-tra tests each using penetration tools and checking the resources present in the framework. The ten vulnerabilities listed by OWASP Top Ten, one of the data security projects developed by OWASP (Open Web Application Security Project) and aims to present the ten vulnerabilities of Internet applications that can cause more damage to systems on the Internet. Throughout the experiments, it is possible to illustrate to which vulnerabilities Laravel has defense mechanisms and how developers can use them to protect user data and against which vulnerabilities the organization does not have resources against external attacks.

Keywords: Information Security, Laravel, OWASP

Sumário

Lista de Figuras

Lista de Quadros

Lista de Códigos

Capítulo 1

Introdução

Sistemas de informação são desenvolvidos constantemente com diversos propósitos. Tais sistemas têm como principal foco atender ao cliente da melhor maneira possível, bem como ser entregue no prazo estipulado entre as partes interessadas, tendo em vista que este é um dos fatores mais críticos para o sucesso de um projeto de software, a entrega do produto no prazo. Sendo a entrega do software um requisito primordial para o sucesso do projeto, muitos desenvolvedores optam fazer uso de *frameworks* tornando o desenvolvimento mais sistemático e reutilizável.

Frameworks são basicamente *templates*, uma estrutura de desenvolvimento, com diversas funções que podem ser usadas pelo desenvolvedor, auxiliando na reutilização de componentes já desenvolvidos e testados. Com isto o uso de *frameworks* pelos desenvolvedores tendem a ser frequente e sempre com preocupação na entrega das funcionalidades, ou seja, dos requisitos funcionais (JOY & SINGH, 2015).

Com a maior preocupação por parte dos desenvolvedores nos requisitos funcionais do projeto, os aspectos relacionados à segurança acabam sendo deixados de lado. Vale ressaltar que as aplicações fazem uso dos dados pessoais do usuário, sejam eles sigilosos ou não, o que nos leva a reforçar o cuidado com a segurança de tais informações.

Aplicações *web* tem sido um grande alvo para os atacantes que exploram estes sistemas, devido a sua grande utilização pelos usuários, determinadas aplicações tornam-se vulneráveis se não forem tomadas medidas necessárias para proteção das informações. Algumas das prevenções a serem tomadas está por parte dos desenvolvedores, onde os mesmos devem realizar testes de intrusão como também revisão de código para averiguar se há um erro na segurança da aplicação.

No cenário de segurança em aplicações *web* podemos citar a OWASP (*Open Web Application Security Project*), uma organização sem fins lucrativos com sede nos Estados Unidos cuja missão é promover iniciativas para que pessoas e organizações possam manter-se informados sobre os potenciais riscos de segurança de software. A OWASP gera um relatório que contém as principais vulnerabilidades encontradas em *websites* e que necessitam ser corrigidas pelos

desenvolvedores, este relatório é chamado de Top Ten (OWASP, 2017).

Neste projeto será estudado o *framework* Laravel para verificar as técnicas que o mesmo oferece para segurança dos dados da aplicação tendo como base as vulnerabilidades elencadas pela OWASP. Nisto serão analisados o Top Ten 2017: Injeção, Quebra de Autenticação, Exposição de Dados Sensíveis, Entidades Externas de XML (XXE), Quebra de Controle de Acessos, Configurações de Segurança Incorretas, *Cross-Site Scripiting* (XSS), Desserialização Insegura, Utilização de Componentes Vulneráveis, Registo e Monitorização Insuficiente.

1.1 Problema

Como o *framework* Laravel aborda técnicas de segurança para proteger as aplicações contra as vulnerabilidades da *web*?

1.2 Hipótese

O *framework* Laravel apresenta técnicas para tratamento de vulnerabilidades web sendo necessário intervenção do desenvolvedor utilizando métodos e serviços do próprio software.

1.3 Objetivo

1.3.1 Objetivo Geral

Analisar as ameaças mais críticas nas aplicações *web* segundo a OWASP e examinar o tratamento destas vulnerabilidades no *framework* Laravel, verificando que medidas o desenvolvedor deve adotar quando utilizar o *framework* estudado.

1.3.2 Objetivos específicos

- Analisar as técnicas de tratamento de segurança no *framework* Laravel para proteção de aplicações *web* contra ameaças e ataques externos;
- Apresentar medidas de prevenção presentes no *framework* para utilização dos desenvolvedores contra as principais ameaças *web*.

1.4 Justificativa

A prevenção de vulnerabilidades *web* é um tópico de estudo que está em constante evolução. São inúmeras as vulnerabilidades que são exploradas nas mais diversas plataformas e

ferramentas. É devido a esta constante evolução que podemos analisar as ameaças que afetam as aplicações web e como essas aplicações podem ser protegidas.

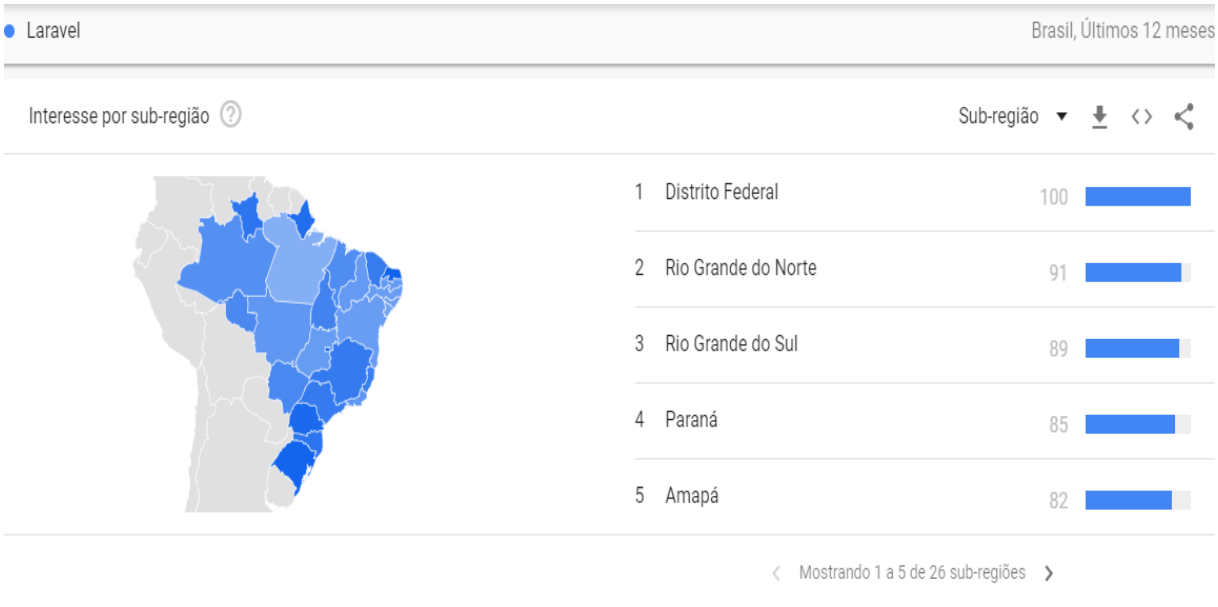
Segundo Singh e Dua (2017), em 2017 62% dos dados estão envolvidos com hackers para realizar ataques a vulnerabilidades. Por volta de 30% dos ataques são feitos em aplicações web. 81% são de aberturas correlacionadas com *bugs* que desviou senhas. 77% dos ataques são realizados por *botnets* e não por indivíduos. E, 32% oprimiram erros de *SQL injection*.

Em 2015 a OWASP realizou uma pesquisa e listou os principais ataques a aplicações *web*. Os ataques foram de *SQL Injection* e XSS, onde as aplicações eram mais propensas a esses ataques (SINGH & DUA, 2017).

Neste projeto utilizaremos o *framework* Laravel para demonstrar seu uso focado em segurança das aplicações *web* tendo como base as dez vulnerabilidades mais críticas definidas pelo OWASP Top Ten 2017.

O *framework* Laravel foi escolhido devido a sua grande aceitação pelos desenvolvedores, tendo uma boa documentação, utilização da arquitetura MVC, fortes pacotes de criptografia, e segundo o Google Trends, o *framework* PHP Laravel é o mais pesquisado em todo o território nacional nos últimos 12 meses, conforme a figura 1.1.

Figura 1.1: Análise das pesquisas sobre Laravel no Google Trends



Fonte: (TRENDS, 2019)

Diante do desafio de evitar que sistemas web sejam invadidos, este trabalho propõe estudar medidas de prevenção que podem ser adotadas pelos desenvolvedores para impedir a exploração das vulnerabilidades críticas já conhecidas por atingirem estas aplicações, garantindo desta maneira a segurança da informação.

1.5 Metodologia

O presente projeto tem como base a pesquisa de carater exploratório e de cunho qualitativo, que visa proporcionar o estudo de caso sobre vulnerabilidades *web* descritas na OWASP Top Ten.

Uma pesquisa exploratória tem por objetivo aprimorar hipóteses e explorar melhor o campo de estudo para melhor proporcionar compreenção do problema. Ela é muito utilizada em pesquisas cujo tema foi pouco explorado, podendo ser aplicada em estudos iniciais de determinados fatos (GIL, 2002).

Para a realização da pesquisa, foram desenvolvidos exemplos de sites para serem utilizados como teste. Na verificação de vulnerabilidades foi utilizado técnicas de invasão para analisar as defesas que o *framework* Laravel possui.

Destaca-se a importância da utilização da pesquisa exploratória, pela necessidade de conhecer e analisar instrumentos adequados aos sujeitos que foram realizados as investigações. Por tal motivo, o estudo nos trará uma melhor compreensão e precisão dos objetivos alcançados na pesquisa.

Este tipo de pesquisa nos traz dados qualitativos, os resultados poderão aprofundar nossa compreensão do tema proposto. Para Minayo (2012), a pesquisa qualitativa trabalha com o ambiente de significados, aspirações, valores e atitudes, o que corresponde a um espaço mais profundo das relações dos processos e dos fenômenos. Sendo assim, o estudo das vulnerabilidades ampliará nossos conhecimentos, bem como o estudo de caso sobre o tema abordado, podendo também contribuir na transferência de mais informações para o campo de desenvolvimento de aplicações web e segurança da informação.

1.6 Organização do trabalho

A organização deste trabalho segue da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica dos conceitos utilizados na pesquisa; o Capítulo 3 apresenta os experimentos e resultados da pesquisa, demostrando as técnias de segurança que o *framework* oferece para proteção das vulnerabilidades do Top Ten; por fim, o Capítulo 4 mostra a conclusão do trabalho e as considereações finais

Capítulo 2

Fundamentação teórica

Nesta seção são apresentados conceitos utilizados no desenvolvimento deste projeto e suas respectivas justificativas para o uso dos mesmos.

2.1 Aplicações web

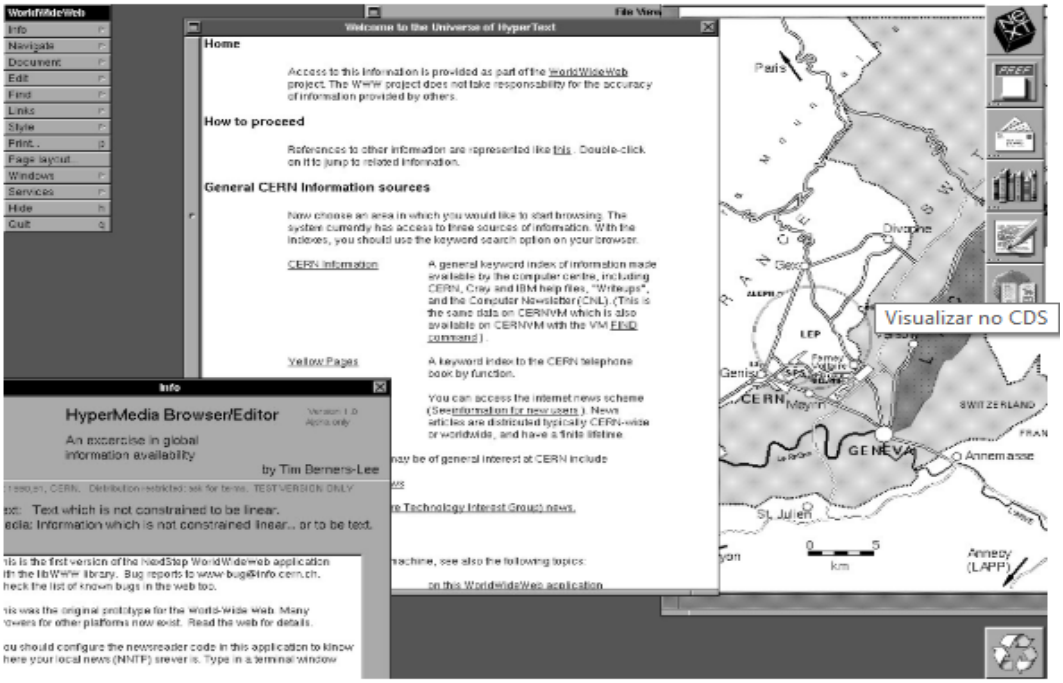
2.1.1 Origem da web

Alguns anos depois do sugimento da internet, à qual foi criada com o objetivo inicial de comunicação entre os membros da Secretaria de Defesa dos Estados Unidos para troca de informações, a CERN (Centro Europeu de Pesquisa Nuclear) estava com dificuldades de comunicação com seus pesquisadores pois os mesmos estavam em diversos projetos. O ciêntista Tim Berners-Lee criou em março de 1989 a *World Wide Web* originalmente concebida para atender a demanda do compartilhamento de informações (CERN, 2019).

A WWW tinha como objetivo combinar as tecnologias em evolução dos computadores, redes de dados e hipertexto em um sistema de informações global de fácil de usabilidade. O design da WWW permitiu o acesso fácil as informações presentes em uma página inicial da *web* vinculada a informações úteis para os cientistas do CERN, como por exemplo, guias para uso dos computadores centrais do CERN. Para a realização de pesquisas era necessário o uso de palavras-chaves (CERN, 2019).

O navegador da *Web* original de Berners-Lee executando nos computadores da NeXT incluía a capacidade de modificar páginas diretamente do navegador, o primeiro recurso de edição da *Web*. Onde também apresentava uma aparencia muito proxima dos navegadores atuais.

Figura 2.1: Navegador mundial da NeXT criado por Tim Berners-Lee em 1993



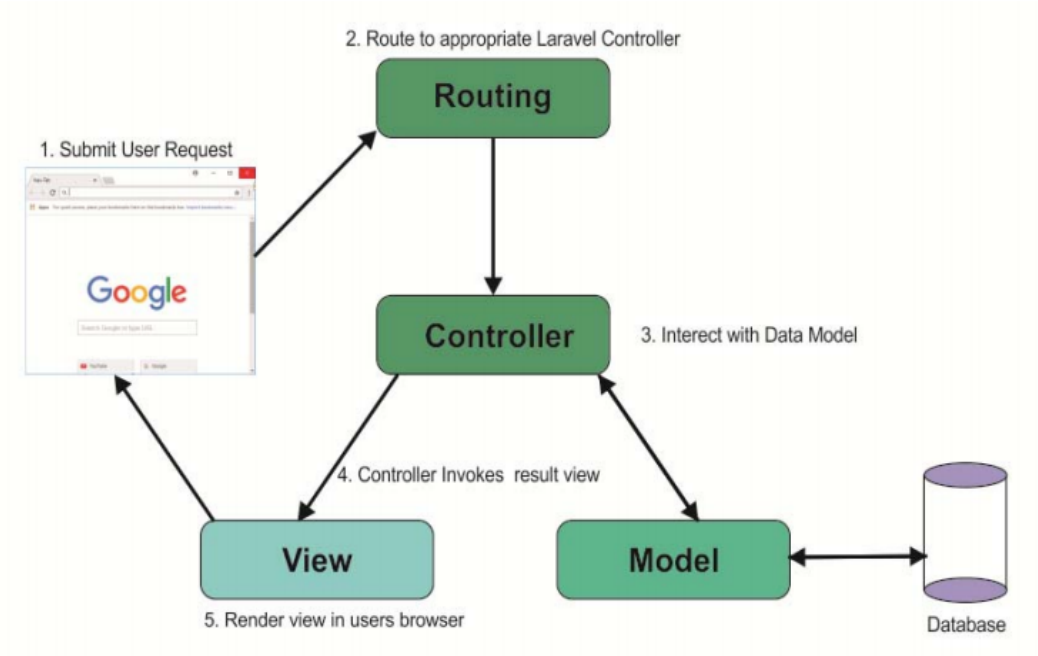
Fonte: CERN (2019)

Segundo Muniz e Lakhani (2013), uma aplicação *web* é todo aplicativo executado através de um navegador. Elas operam seguindo uma arquitetura Cliente-Servidor, onde há um cliente que executa comandos e envia requisições para um servidor *web*, o qual é responsável por disponibilizar as páginas do site através do protocolo de rede HTTP. Sendo assim, há a necessidade de que o desenvolvedor da aplicação tome medidas de segurança para que as requisições vindas dos usuários sejam legitimas e que o servidor forneça acesso apenas a páginas correspondente ao nível de privilégio do usuário que realizou a requisição

2.2 Laravel

O Laravel é um *framework* PHP criado por Taylor Otwell em junho 2011 com o objetivo de facilitar a criação de projetos de forma rápida deixando o código mais limpo e com uma boa aprendizagem, facilitando assim manutenções. Este *framework* possui arquitetura MVC (*Model-View-Controller*) e tem, como aspecto principal, o de auxiliar no desenvolvimento de aplicações de forma ágil e simplificada incentivando a boas praticas de programação e utilização de padrões.(STAUFFER, 2017).

Figura 2.2: Modelo de arquitetura MVC do Laravel



Fonte: (ANIF et al., 2017)

Para o gerencimento de dependências o Laravel dispõe do *composer*, uma ferramenta criado por Nils Adermann e Jordi Boggiano para gerenciamento de dependências em PHP. O *composer* permite que o desenvolvedor declare as bibliotecas que necessita para o seu projeto e as instalará e atualizará. O *composer* foi inspirado no NPM e Bundler, porém, ele gerencia cada pacote ou biblioteca em uma base por projeto, instalando-os em um diretório chamado *vendor* dentro do projeto (ADERMANN & BOGGIANO, 2019).

O Laravel também contém uma ferramenta de linha de comando chamada de *Artisan*, um conjunto de ações internas com a oportunidade de inclusão de outras. Com o *artisan*, o desenvolvedor é capaz de acionar ações, como testes de unidade, tarefas agendadas e execução de migrações do banco de dados (MCCOOL, 2012).

O Laravel tem uma estrutura de arquivos e diretórios projetados para exibição das páginas do site. Em um arquivo de rotas (*/routes/web.php*), os URLs do site são especificados e mapeados em *closures* ou para métodos definidos nos controladores. Nos arquivos de rotas também é possivel definir os *Middleware*, elementos que o Laravel utiliza para autenticar usuários e acesso a funções que eles tenham permissão (STAUFFER, 2017).

Os controladores são *contêiners* para métodos que contém a lógica da aplicação. As *closures* ou os métodos implementados nos controladores são responsáveis por atender as requisições HTTP recebidas pelo servidor *Web*. O código dos controladores é escrito na linguagem PHP e além de controlar o fluxo da aplicação, fornecerá dados do banco de dados para a visualização nas *views* (MCCOOL, 2012).

Para exibição do conteúdo, o Laravel utiliza *Views*, onde o conteúdo de uma página é des-

crito utilizando um motor de *templates* chamado *Blade*. Este *template engine* personalizado oferecido pelo Laravel foi inspirado no *engine Razor* da plataforma .NET e contém uma sintaxe limpa e concisa, além de conter um modelo de herança intuitivo e de fácil extensão. Os arquivos das páginas ficam armazenados no diretório *resources/views/* e devem ter a extensão *.blade.php*, estes arquivos são compilados em PHP e armazenados em cache, tornando-os assim mais rápidos e permite também inserir códigos PHP nativos nos arquivos *Blade* se desejar (STAUFFER, 2017, p. 40-79).

Para o *framework* Laravel a definição da estrutura do banco de dados é facilitada através do arquivo de migrações. Este arquivo pode ser criado através de um comando da CLI (*Command-Line Interface*) e nele podem ser definidos cada nova tabela, coluna, indice, tipo do dado, isso em pouco tempo. O Laravel utiliza três formas de manipulação de Banco de Dados, duas delas utilizando a *Facade DB*: *Raw SQL*(SQL Bruto), e a *Fluent Query Builder* para uso através de métodos especificos para cada transação requerida no BD. A outra forma de manipulação do banco de dados é através do *Eloquent*, um dos mais populares recursos do Laravel. O Eloquente é um ORM (*Object-Relational Mapping*) *ActiveRecord* o que significa que o mesmo é uma camada de abstração do banco de dados e fornece uma interface única para a interação com o BD. Isto significa que através de uma classe do *Eloquent* é possível fornecer a interação com a tabela inteira ou até representar uma linha individual da tabela. Ele se baseia no modelo *convetion over configuration* para permitir a elaboração de modelos com código mínimo e com grande poder nas funcionalidade. Nisto o ORM oferece ênfase a simplicidade (STAUFFER, 2017, p. 158-190).

2.3 Segurança da informação

A informação é um ativo muito importante para as organizações e que necessita ser protegida, além disto ela pode ser criada e mantida em diferentes formas: papel, armazenada em um meio eletrônico, falada em uma conversa ou transmitida por qualquer dispositivo (ISO/IEC27002, 2005).

Para compreender o que é informação, é necessário compreender o que são dados.

De acordo com Chiavenato (2014), dados são registros, indices, que, em si mesmos, não possuem significado. Os dados necessitam ser classificados, relacionados, obter um processamento para que os mesmos possam ganhar siginificado e obter informação para assim realizar a tomada de decisão.

Gordon e Gordon (2006), define dados como valores, medidas e fatos, onde não estão organizados, nem contextualizados. Onde por exemplo os dados: 1015, 1500, 42, não teem siginificados. A informação, por sua vez, são os dados processados, dados que foram filtrados, formatados, analisados e resumidos. Os dados 1015, 1500, 42 após serem analisados podem oferecer as seguintes informações: Número de espera 1015 ocorreu às 15:00 com o atendente

de cadastro 42.

Outrora as informações mais importantes para as organizações poderiam ser guardadas em gavetas e trancadas. As informações sempre foram um ativo importante para as organizações, mas atualmente, independente do nível de tecnologia da organização, a proteção deste ativo deve ser uma precaução para os proprietários e profissinais de tecnologia (PONTES, 2008).

Como vivemos em um mundo interconectado a proteção dessas informações contra as ameaças e vulnerabilidades é de intrínseca importância, uma vez que a prevenção é o melhor meio para que empresas, em geral, tenham menores incidentes em danos em sistemas devido a essas vulnerabilidades ou fragilidades dos mesmos.

Segurança da informação também conhecida como segurança do sistema, de computadores e também proteção da infra-estrutura, é definida por Pfleeger et al. (2015) como a proteção dos itens que o usuário valoriza, chamados de ativos de um computador. Existem vários tipos de ativos, dentre eles hardware, software, processos, documentos, fotos. Quem define o que é importante e o que proteger é o usuário, onde deve-se realizar a identificação do que é de valor para o mesmo.

Segundo a ISO/IEC 27002 (2005), segurança da informação é a proteção das informações que possam ser afetadas por ameaças, diminuindo desta maneira os riscos e também aumentando o retorno sobre os investimentos das organizações.

A segurança da informação pode ser obtida através de um conjunto de controles, dentre os quais pode-se citar os processos, politicas, estruturas organizacionais e funções de hardware e software. Estes mesmos controles precisam ser determinados, analisados e monitorados para garantir a segunrança da organização e as metas do negócio.

A segurança da Informação garante a continuidade do negócio, fornecendo proteção contra as ameaças e ataques às informações. Ela é obtida através de gerenciamento dos riscos, tais gerenciamentos devem ser feitos de forma adequada e utilizando implementação de políticas, procedimentos e métodos de hardware e software (ISO/IEC27002, 2005).

De acordo com o PMI (2013) um risco é uma condição incerta que, se vier a ocorrer, poderá provocar um efeito positivo ou negativo nos ativos da organização. Uma ou mais causas podem dar origem aos riscos, e os mesmos, podem ocasionar vários impactos.

2.3.1 Ameaças e vulnerabilidades

Segundo Amoroso (1994), Uma ameaça pode ser descrita como uma ocorrência potencial, maliciosa o não, que possa ter um efeito indesejável nos recursos ou ativos associados a um sistema de computador.

Amoroso (1994) também define Vulnerabilidade como uma característica que possibilita a ocorrência de uma ameaça. Em outras palavras, a presença de vulnerabilidades permite que danos aconteçam em um sistema de computador. Como tal, as ameaças de um sistema de computador podem ser mitigadas e removidas ao serem identificandas as vulnerabilidades. E

ataque é alguma ação tomada por um intruso mal-intencionado que envolve a exploração de certas vulnerabilidades para causar uma ameaça existente.

Para Pfleeger et al. (2015) uma ameaça a um sistema de computação é um conjunto de circunstâncias que tem o potencial de causar perda ou dano. Os desastres naturais também são ameaças, eles podem trazer um sistema para baixo quando a sala de informática é inundada ou ocorre um terremoto, por exemplo. Já uma vulnerabilidade é uma fraqueza no sistema, por exemplo, em procedimentos, design ou implementação, que pode ser explorada para causar perda ou dano. Por exemplo, um determinado sistema pode ser vulnerável a manipulação de dados não autorizados, porque o sistema não verifica a identidade de um usuário antes de permitir o acesso.

Para lidar com estes problemas é necessário usarmos um controle como proteção. Em outras palavras, um controle é uma contramedida, uma ação, dispositivo ou procedimento que retira ou reduz uma vulnerabilidade. Desta maneira, os controles impossibilitam que as ameaças atuem nas vulnerabilidades.

2.3.2 Confidencialidade, integridade e disponibilidade

Os aspectos confidencialidade, integridade e disponibilidade são três das características fundamentais para a segurança da informação e também objetos de ameaças. Estas propriedades são conhecidas como os pilares de sustentação da segurança ou também tríade CID da segurança (PFLEEGER et al., 2015).

Um dos pilares é a confidencialidade, que segundo Stewart et al.(2005), é o princípio de que os objetos (Itens de dados) não são divulgados a sujeitos não autorizados. Algumas áreas requerem descrição e sigilo de divulgação de informações, pode-se citar como exemplo: transações financeiras, notas de alunos, desclarações fiscais e também registros médicos, onde um prontuário médico de determinado paciente, apenas o médico e funcionários da saúde, com permissão, poderão visualizar o mesmo.

Algumas vezes, no entanto, não é tão obvio que determinada informação é sigilosa. Por exemplo: compras de alimentos, acesso a livros, mudanças de horário em uma localização, mas podem revelar algo que alguém deseja manter em descrição.

Pfleeger et al.(2015), define a confidencialidade como a propriedade em que apenas pessoas ou sistemas autorizados pode acessar dados protegidos. Confidencialidade relaciona-se mais aos dados, embora possamos pensar confidencialidade de uma peça de hardware (uma invenção nova) ou de uma pessoa (o paradeiro de um criminoso procurado). Uma palavra que capta a maioria dos aspectos de confidencialidade é a visão, mas uma falha de confidencialidade não significa necessariamente que alguém vê um objeto. Em segurança da informação, confidencialidade geralmente significa obter determinado item de dado.

Algumas propriedades que podem significar uma falha de dados de confidencialidade de acordo com Pfleeger et al.(2015):

- Uma pessoa não autorizada acessa um item de dado;
- Um processo ou programa não autorizado acessa um item de dado;
- Uma pessoa não autorizada acessa um valor aproximado de dados, por exemplo: não saber o salário exato de alguém, mas sabendo que o salário cai em um determinado intervalo ou excede um determinado valor;
- Uma pessoa não autorizada aprende a existência de um dado, por exemplo: sabendo que uma empresa está desenvolvendo um determinado produto novo ou que as negociações são em curso sobre a fusão de duas empresas.

A integridade é o pilar que tem capacidade de garantir que um ativo (objeto) mantenha sua veracidade e seja modificado apenas por sujeitos autorizados. Esta propriedade tem por objetivo garantir se um determinado recurso ou informação é confiável ou não (STEWART et al., 2005).

De acordo com Pfleeger et al.(2015), a integridade de um item é definido por ser preciso, não modificado, modificado apenas de formas aceitáveis, modificado apenas por pessoas autorizadas, modificado apenas por processos autorizados, consistente, significativo e utilizável.

Integridade pode ser dividida em, integridade dos dados (o conteúdo da informação) e integridade da origem (a fonte dos dados, denominada de autenticação). Por exemplo, um jornal obtem informações devido a um vazamento no Banco Central e realiza a impressão das mesmas, porém atribui estas informações a fonte errada. A informação é impressa como recebida, pois preserva a integridade dos dados, no entato a integridade da origem está corrompida devido sua fonte está incorreta (BISHOP, 2004).

Em Bishop (2004) são apresentados dois mecanismos de integridade. O primeiro mecanismo é a prevenção que procura manter a integridade dos dados bloqueando quaisquer tentativas não autorizadas de alterar os dados ou usuários autorizados realizem alterações indevidas. Outro mecanismo é a detecção, onde não é impedido violações de integridade e sim relatado que a integridade dos dados não é mais confiável. Podendo usar desta forma recursos do sistema para gerar alertas com informações da causa da corrupção dos dados.

O princípio de que os indivíduos autorizados recebem acesso oportuno a objetos com largura de banda suficiente para realizar a interação desejada, é definida como disponibilidade (STEWART et al., 2005).

De acordo com Bishop (2004), a disponibilidade refere-se à capacidade de usar as informações ou recursos conforme desejados. Um sistema indisponível é tão ruim como nenhum sistema, logo a disponibilidade é um aspecto importante da confiabilidade. Para a segurança da informação um aspecto com relação a disponibilidade é que algum sujeito ou processo pode intencionalmente negar o acesso aos dados ou serviços, tornando-os indisponíveis. Um ataque conhecido que torna sistemas lentos e impossíveis de se utilizar ou até mesmo derrubando o serviço, é conhecido como *Distributed Denial of Service*(DDOS).

De acordo com Pfleeger et al.(2015), a disponibilidade se aplica tanto a dados quanto a serviços, ou seja, as informações e processamento de informações. Algumas características da disponibilidade:

- Há uma resposta oportuna ao nosso pedido;
- Os recursos são alocados de forma justa para que alguns solicitantes não sejam favorecidos a outros;
- Concorrência é controlada, isto é, acesso simultâneo, gerenciamento de *deadlock*, e acesso exclusivo são suportados conforme necessário;
- O serviço ou sistema pode ser utilisado facilmente, característica de usabilidade, porém se um sistema for inutilizável ocasionará falha de disponibilidade.

A segurança da informação visa impedir a visualização não autorizada, confidencialidade, ou modificação dos dados, integridade, preservando o acesso, disponibilidade.

2.4 OWASP

O OWASP (*Open Web Application Security Project*) é uma organização sem fins lucrativos com sede nos Estados Unidos criada em dezembro de 2001 e tem como objetivo disseminar iniciativas para que organizações e pessoas possam obter informações sobre os riscos de segurança que os softwares podem oferecer. Entre as atividades do OWASP para espalhar as informações sobre segurança em aplicações estão a organização de conferências, realização de palestras, publicação de livros e projetos diversos, entre eles, o OWASP Top Ten (OWASP, 2017).

O OWASP Top Ten é um projeto que visa disseminar as vulnerabilidades das aplicações web elencadas mais prejudiciais para a segurança da informação. É um documento que contém as principais vulnerabilidades encontradas em sites e que precisam ser corrigidas por desenvolvedores, pois não dependem de medidas de segurança implantadas nos servidores. Seu objetivo principal é educar desenvolvedores, gestores, design, arquitetos e organizações sobre as consequências das mais importantes vulnerabilidades de segurança de aplicações *web*. Além de apresentar as vulnerabilidades o Top Ten apresenta algumas técnicas para se proteger de determinados ameaças.

A lista das principais vulnerabilidades em aplicações *web* é realizada através de consultas com especialistas em segurança da informação e desenvolvedores *web* de diversos segmentos. A primeira versão do Top Ten foi em 2003, que com algumas atualizações gerou a edição de 2004, em seguida as edições que vieram foram as de 2007, 2010, 2013 e 2017. A edição mais recente do OWASP Top Ten é do ano de 2017, lançada no respectivo ano, a edição contém 10 vulnerabilidades que foram destacadas como as que necessitam de maior atenção do desenvolvedor para implementação da sua segurança na aplicação. Dentre elas estão:

Quadro 2.1: OWASP Top Ten 2017

Vulnerabilidades	Descrição
A1:Injeção	Falhas de Injeção ocorrem quando dados não confiáveis são inseridos nos campos de entrada como parte de um comando válido
A2:Quebra de Autenticação	Atacantes podem assumir identidades de outros usuários devido o comprometimento dos passwords ou tokens de sessão
A3: Exposição de Dados Sensíveis	Dados sensíveis como número de cartão de crédito e dados pessoais necessitam de segurança tanto no armazenamento como em trânsito
A4:Entidades Externas de XML (XXE)	Vulnerabilidade ocasionada devido a processadores XML antigos ou mal configurados que avaliam entidades externas em documentos XML
A5:Quebra de Controle de Acessos	Atacantes podem ter acesso a dados ou funcionalidades não autorizadas devido à falta de implementação correta das restrições de que cada usuário pode fazer no sistema
A6:Configurações de Segurança Incorretas	Tem como objetivo as configurações definidas na aplicação e como elas podem trazer vulnerabilidades a aplicação se mal definidas
A7:Cross-Site Scripting (XSS)	O atacante pode inserir código malicioso no navegador da vítima por falta de validação.
A8:Desserialização Insegura	Invasores podem realizar ataques de injeção, por repetição ou alteração de privilégios
A9:Utilização de Componentes Vulneráveis	Componentes externos como bibliotecas e módulos estão vulneráveis a ataques externos e são implementados a aplicação
A10:Registo e Monitorização Insuficiente	Atacantes continuam a realizar ataques as aplicações devido a resposta a incidentes serem insuficientes ou até inexistente

Fonte: O AUTOR

2.4.1 Injeção

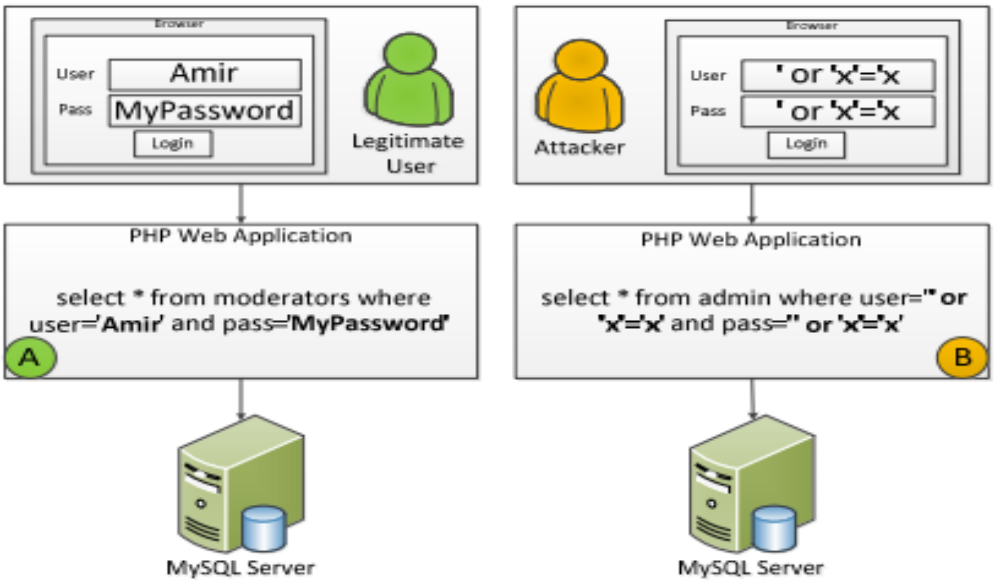
Injeção é uma falha que continua com a primeira posição do ranking da OWASP Top Ten sendo então a que mais os desenvolvedores devem tomar cuidado para a implementação da segurança nesse aspecto. As falhas de Injeção podem ser referentes a SQL, LDAP, PHP e ocorrem quando dados hostis são inseridos numa consulta ou parte de um comando, onde atacantes

tentam injetar códigos maliciosos seja por meio de um vetor de injeção, parâmetros e serviços (OWASP, 2017).

As consequências deste ataque pode ocasionar na perda de dados, divulgação de informações para partes não autorizadas, negação de acesso e pode levar ao controle completo do sistema, prejudicando assim a reputação da empresa.

Um exemplo de ataque de injeção pode ser visto na figura 2.3, onde o atacante terá acesso ao sistema através do login do usuário:

Figura 2.3: Exemplo de ataque de *SQL Injection*



Fonte: (SADEGHIAN et al., 2014)

Para proteção contra esta vulnerabilidade, Sadeghian et al.(2014) cita uma abordagem para proteger as aplicações *web* contra injeção, pricipalmente de SQL. A solução chamada de SQL-rand, seria que durante o desenvolvimento da aplicação fosse adicionado um número inteiro a cada palavra-chave do SQL. Logo, o atacante ao realizar a injeção não terá êxito pois utilizará comando SQL padrão, o qual não terá efeito sem a chave aleatória predefinida.

A OWASP (2017) apresenta algumas recomendações para proteção contra injeção. Dentre elas estão a utilização de *whitelists* para validação dos dados de entrada por parte do servidor. Utilização de uma API ou ferramenta ORM (*Object Relational Mapping*), ou também para toda consulta utilizar um processo de verificação os caracteres especiais e utilizando um interpretador específico.

2.4.2 Quebra de autenticação

Autenticação quebrada ocorre quando invasores têm acesso a chaves, senhas ou até tokens de sessão podendo assumir identidades de outros usuários de forma temporária ou permanente.

Esses ataques podem ser de força bruta automatizada, ferramentas de ataques de dicionário e também com relação a *tokens* de sessão não expirados. A aplicação é considerada vulnerável se os identifacadores de sessão e os *tokens* de autenticação não são invalidados após o *logout*, se permite *passwords* padrão, tais como admin/admin, ou também não renova os identificadores de sessão quando o processo de autenticação é concluído com sucesso (OWASP, 2017).

Dependendo do domínio do aplicativo *web* a quebra de autenticação pode ocasionar fraude de segurança social, lavagem de dinheiro ou divulgação de informações confidenciais.

Um exemplo desta vulnerabilidade são os ataques que ocorrem de autenticação, devido se usarem apenas as palavras-passe como um único fator. Outro exemplo é o tempo de expiração da sessão não ser definido.

A OWASP (2017) cita algumas dicas de prevenção contra a quebra de autenticação. Dentre elas estão a utilização de autenticação multifator para evitar ataques automatizados de *credential stuffing*, limitar o número máximo de autenticações falhadas, e utilizar um gestor de sessões seguro no servidor o qual gere de forma que os identificadores de sessão não constém na URL e que sejam invalidados após o *logout* ou após um período de tempo predefinido.

2.4.3 Exposição de dados sensíveis

A terceira falha abordada pelo Top Ten é a Exposição de Dados Sensíveis onde APIs e aplicativos da *web* não protegem apropriadamente dados confidenciais como financeiro, assistência média, podendo os invasores roubar e modificar esses dados que estão fracamente protegidos. Normalmente esses dados possuem informações confidenciais como registro de saúde, cartões de credito, credenciais, que devem ser protegidos com tecnologia extra, como criptografia, e terem cuidados adicionais ao estarem em trânsito (OWASP, 2017).

Dentre as consequências para este ataque está o roubo de identidade, fraudes com cartões de credito, credenciais. Dados pessoais que ficaram a disposição do atacante.

Um exemplo para esta vulnerabilidade seria uma aplicação que armazena as informações na base de dados de forma criptografada, porém em uma consulta a base de dados os dados são decifrados permitindo um ataque de injeção de SQL para obtenção das informações.

Para a prevenção de ataques a esta vulnerabilidade, pode-se citar a encriptação dos dados em trânsito, utilizando TLS juntamente com cifras PFS. Desativar o cache para resposta com dados sensíveis. Encriptar dados em repouso (OWASP, 2017).

2.4.4 Entidades externas de XML (XXE)

Entidades Externas de XML é uma vulnerabilidade ocasionada devido a processadores XML antigos ou mal configurados que avaliam entidades externas em documentos XML. Isto ocorre utilizando o manipulador de URI de arquivo, incluindo conteúdo hostil, seja de forma remota ou local, em documentos XML e assim explorando as dependências, integrações e códigos

vulneráveis (JAN et al., 2015).

Essas falhas podem ser usadas para execução remota de código, compartilhamento de arquivos internos e ataques de negação de serviço. E se também o analisador XML não bloquear o acesso da entidade externa, e o atacante obter o referido dado, poderá ter acesso a dados de outros usuários, violando a confidencialidade das informações (OWASP, 2017).

Como exemplo de ataque XXE, o código 2.1 apresenta um documento XML em que o atacante substituirá a entidade externa '&' por *'/etc/passwd'* que contém informações confidenciais do servidor.

Código 2.1: Código de exemplo da vulnerabilidade XXE

```
1 <?xml version="1.0" ?>
2 <!DOCTYPE myFile [
3     <!ELEMENT myFile ANY >
4    <!ENTITY xe SYSTEM "file:///etc/passwd">
5 ]>
6 <myFile>&xe</myFile>
```

Fonte: (JAN et al., 2015)

Como orientação para prevenir os ataques XXE, a OWASP apresenta algumas sugestões para serem seguidas: Optar por um formato mais simples como o JSON; Atualizar todos os processadores e bibliotecas de XML; Implementar filtragem e validação para os valores de entrada no documento XML, prevenindo assim dados hostis.

2.4.5 Quebra de controle de acessos

Quebra de controle de acesso é uma vulnerabilidade onde os invasores podem ter acesso a dados ou funcionalidades não autorizadas devido à falta de implementação correta das restrições de que cada usuário pode fazer no sistema. Os pontos fracos do controle de acesso são comuns devido à falta de detecção automatizada e à falta de testes pelos desenvolvedores de aplicativos, pois a implementação de um controle de acesso não é uma tarefa trivial, pricipalmente levando em consideração os sitemas *web* com o uso do protocolo HTTP que não armazena estado sendo necessário a verificação de toda requisição que acesse informação sigilosa. (GAUTHIER & MERLO, 2012; OWASP, 2017)).

Como impacto desta vulnerabilidade os invasores agem como administradores ou usuários que possuem privilégios, podendo criar, visualizar, alterar e excluir registros, além de execução de funções fora do limite de seu usuário.

Um exemplo do ataque de quebra de controle de acesso, pode ser visto na figura abaixo. O atacante força o acesso através da URL, onde obtendo acesso terá privilégio de administrador.

Figura 2.4: Exemplo de quebra de controle de acesso

Fonte: (OWASP, 2017)

Para prevenção deste tipo de ataque a OWASP apresenta algumas abordagens que evitam esta vulnerabilidade: Modelar o controle de acesso para que venha garantir a propriedade dos registros; Registrar falhas de controle de acesso e avisar os administradores do sistema o ocorrido; Incluir testes unitários e de integração para as funcionalidades de controle de acessos (OWASP, 2017).

2.4.6 Configurações incorretas de segurança

A sexta vulnerabilidade segundo o OWASP Top Ten é a Configuração Incorreta de Segurança. Esta vulnerabilidade apresenta muitas vezes dados confidenciais para os invasores, abrindo portas para o acesso dos mesmos. Isso é comumente ocorrido quando ocorre configurações incompletas, armazenamento em nuvem aberta, mensagem de erro contendo informações confidenciais, a não instalação de atualizações e cabeçalhos HTTP configurados incorretamente. Esta vulnerabilidade é baseada em como as configurações da aplicação *web* é tratada e não em código, logo ela deve ser impedida com esforços de desenvolvedores e administradores (OWASP, 2017).

Uma falha nesta vulnerabilidade pode ocasionar acesso não autorizado aos dados do sistema e podendo comprometer o sistema de uma forma completa.

Como exemplo da vulnerabilidade de Configurações Incorretas de Segurança, pode-se citar a listagem de diretórios do servidor que não está desativada. O atacante pode realizar o download de classes da aplicação, se compiladas, pode reverté-las e visualizar outras vulnerabildades.

Para prevenção desta vulnerabilide é incluído a verificação de componentes externos, o que é inclusive uma das ameaças presentes no Top Ten. Uso da plataforma de forma mínima e necessária, sem uso de funcionalidades desnecessárias, como também a remoção de tais funcionalidades. Criação de processos automatizados para verificação das configurações em todos os ambientes (OWASP, 2017).

2.4.7 Cross-Site scripting (XSS)

Cross-Site Scripting (XSS) é uma vulnerabilidade em que o atacante insere código no navegador da vítima, devido à falta de validação ou escape. O ataque sendo bem-sucedido, o comportamento do sistema ou do site podem ser alterados por completo. O *script* utilizado nesta manipulação do atacante pode ser executado toda vez que a página for carregada ou quando

ocorrer a execução de um evento associado. Há três tipos de XSS: refletido, persistente ou baseados no DOM (SHRIVASTAVA et al., 2016).

O XSS Persistente é quando o servidor salva a entrada maliciosa que foi adicionada através de um formulário em seu banco de dados, e a saída para a página *web* não é validada, sendo apresentada da mesma forma que foi armazenada. Exemplo seria um formulário de revisão, mensagens de *webmail*. O XSS Refletido ocorre quando o navegador exibe dados passados para a aplicação, como mensagem de erro, parâmetros de busca, sem a devida validação, ou o atacante cria uma URL com script mal-intencionado e incita a vitíma a acreditar que a URL é confiável, enviando o link por email ou adicionado a outra página web em outro servidor. Tendo recebido o link o usuário irá confiar na URL que aparece na barra de endereços, por se tratar de um endereço legítimo, no entanto, pode ser uma referência a um arquivo executável contendo um vírus ou um cavalo de tróia. O ataque de XSS baseado no DOM permite a modificação de propriedades do modelo de objetos dos documentos HTML diretamente no navegador do usuário alvo, e permite que scripts dinâmicos, incluindo JavaScript, referencie componentes no documento, como por exemplo, uma sessão, um *cookie* ou um campo de formulário (OWASP, 2017).

De acordo com OWASP (2017), Os tipos de XSS refletido e baseado em DOM tem o impacto considerado moderado, porém o XSS *stored* ou persistente pode ocasionar prejuizo ao usuário e a corporação, prejudicando sua imagem. Dentre os impactos causados por esta vulnerabilidade estão o roubo de sessão, redirecionar o usuário a sites maliciosos, ataques contra o *browser* do usuário podendo ocasionar o download de malwares, entre outros.

Como exemplo de código XSS pode-se citar uma aplicação onde não há validação ou técnicas de *escaping*:

Código 2.2: Código de exemplo XSS 01

```
1 (String) page += "<input␣name='creditcard'␣type='TEXT'
2 value='"+ request.getParameter("CC") + "'>";
```

Fonte: (OWASP, 2017)

Após a substituição do parâmentro "CC"pelo código:

Código 2.3: Código de exemplo XSS 02

```
1 '><script>document.location=
2 'http://www.attacker.com/cgi-bin/cookie.cgi?
3 foo='+document.cookie</script>'
```

Fonte: (OWASP, 2017)

Isto fará com que o atacante tenha acesso a sessão atual do utilizador.

Para evitar esta vulnerabilidade, a OWASP (2017) elenca algumas prevenções: O uso de Firewall que manipula as solicitações entre cliente e servidor; o uso de métodos de escapes para HTML, JavaScript, JSON, para evitar que caracteres especiais venham manipular a aplicação; Não permitir a inserção direta de dados em comentário HTML, nome de *tag*, CSS. A OWASP também indica para proteção contra XSS o uso de *frameworks*, tratamento adequado, como *escaping*, para informações no pedido HTTP.

2.4.8 Desserialização insegura

A serialização é um processo de gravação dos dados como um binário bruto para a transferência através de uma rede. Para recriar o dado que foi serializado, é utilizado a desserialização, que é o processo de recuperar os dados binários de um arquivo ou até mesmo de um soquete de rede para reconstruir o objeto. Dados que não estiverem protegidos por alguma função de criptografia podem ser modificados no trânsito da mensagem pelo atacante. Logo, os dados não confiáveis não podem ser desserializados sem a verificação e validação que não sofreram alteração (DEHALWAR et al., 2017).

A vulnerabildiade de Desserilização Insegura foi incluída no Top Ten da OWASP na sua última versão lançada, a de 2017. No entanto os ataques a esta vulnerabilidade pode causar percas financeiras a indivíduos ou emprezas, execução remota de código e quebra de controle de acesso podendo obter privilégios administrativos da aplicação.

Como exemplo da desserialização insegura, um fórum utiliza a serialização para enviar um *cookie* com informações de ID do utilizador, *hash* da sua password, e outros estados do usuário. Um atacante pode interceptar o objeto serializado e alterar caracteristicas que lhe garantam privilégios de administrado.

Código 2.4: Objeto serializado para ser enviado pela rede

```
1 a:4:{i:0;i:132;i:1;s:7:"Mallory";i:2;s:4:"user";
2 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Fonte: (OWASP, 2017)

Código 2.5: Intervenção de um atacante na serialização do objeto

```
1 a:4:{i:0;i:1;i:1;s:5:"Alice";i:2;s:5:"admin";
2 i:3;s:32:"b6a8b3bea87fe0e05022f8f3c88bc960";}
```

Fonte: (OWASP, 2017)

A OWASP descreve que se os dados a serem desserializados não vierem de fontes seguras, não haverá uma forma segura para o processo. Mas caso não seja possível verificar a origem do objeto enviado, a organização apresenta dicas de prevenção contra esta vulnerabilidade, algumas delas são: realizar verificação de integridade dos dados implementando um certificado digital nos objetos serializados, registrar falhas e exceções na disserialização, monitorar a disserialização criando alertas caso a operação apresente anomália (OWASP, 2017).

2.4.9 Utilização de componentes vulneráveis

Utilizar Componentes com Vulnerabilidades Conhecidas é uma falha onde determinado componente vulnerável é explorado, esse ataque pode facilitar a perda de dados ou a tomada do servidor, isto ocorre porque bibliotecas, estruturas e outros módulos de software são componentes que são executados com o mesmo privilégio da aplicação. Quando estes aplicativos e APIs estão usando componentes com vulnerabilidades conhecidas as defesas da aplicação podem ser desfeitas ocasionando ataques e impactos. Dependendo dos ativos que se esteja protegendo, pode ser um risco alto para os negócios (OWASP, 2017).

Para que a aplicação se torne vulnerável não foi atualizado ou corrigido as bases e dependeências dos componentes, o não conhecimento das versões dos componentes que estão tanto no servidor como no cliente. Os programadores não testaram a compatibilidade das novas versões das bilbiotecas lançadas.

Há ferramentas que auxiliam os atacantes a encontrar vulnerabilidades em componentes de sistemas como também a má configuração dos mesmos. Um exemplo de ataque a esta vulnerabilidade são as redes de *Internet of Things* onde deixar os equipamentos atualizados pode ser uma tarefa complexa, mas de esforço necessário.

Algumas dicas para prevenção contra esta vulnerabilidade é encontrada na Owasp Top Ten, entre elas estão a remoção de dependências não utilizadas, obter componentes apenas de fontes oficiais, uma realização de inventário das versões dos componentes tanto do cliente como do servidor asim também de suas dependências (OWASP, 2017).

2.4.10 Registo e monitorização insuficiente

Registro e Monitoramento Insuficientes permitem que os atacantes continuem atacando sistemas, mantenham a persistência e violem os dados. Isto devido ao registro e o monitoramento serem insuficientes, associados à integração ineficaz ou ausente. Estudos de violações mostram que o tempo para detectar uma violação é por volta de 190 dias ou mais, e quando é detectado, é por terceiros, e não por monitoramento ou processos internos. Embora não possa levar a uma invasão direta, o risco de não detectar a invasão de maneira oportuna, é uma falha que pode custar muito dinheiro as empresas (OWASP, 2017).

Um exemplo desta vulnerabilidade seria um projeto onde o código mantido por uma equipe

foi comprometido utilizando uma vulnerabilidade do próprio software. Tendo acesso a próxima versão da aplicação como também dos conteúdos. Não houve monitoramento, nem registro de *logging*.

Outro exemplo pode-se citar um usuário que possuía uma ferramenta para análise de recebimento de *malware*. Foi detectado várias vezes as tentativas de ataque e a ferramenta alertou o ocorrido, mas não houve monitoramento e o ataque foi efetivado.

A OWASP apresenta algumas formas de prevenção a esta vulnerabilidade. Dentre as prevenções estão registro das falhas de autenticação, controle de acesso, validação dos dados; Definição de processos de monitoramento e alerta contra atividade suspeita; Garantia que dados mais criticos terão registros para realização de auditoria com controle de integridade para prevenir adulteração dos dados (OWASP, 2017).

2.5 Teste de invasão

Na verificação das aplicações *web* é realizado alguns testes de invasão, denominados também de testes de penetração (*Pentest*). Este tipo de teste é semelhante a invadir um aplicativo por um invasor, é um dos métodos de busca por problemas de segurança em uma aplicação *web*.

De acordo com Chu e Lisitsa (2018), a metodologia utlizada para avaliar a segurança de sistemas e redes de computadores é denominada de teste de penetração. Todo o ciclo de teste é parcialmente longo e é realizado de forma manual, sendo necessário ferramentas para melhorar a eficiência de tais testes, os tornando automatizados.

Pode ser utilizado algumas abordages para a realização de testes de penetração em aplicações *web* em busca de vulnerabilidades. Dentre estes tipos de testes pode-se citar teste de caixa preta, teste de caixa branca, ou uma mistura de ambos os testes denominada de teste de caixa cinza.

2.5.1 Caixa preta

No teste de caixa preta assume que o atacante não tem conhecimento da aplicação, dos processos de negócio, dos alvos de rede. Os atancantes para concluir o ataque utilizando o teste de caixa preta passam um bom tempo estudando o ambiente alvo antes de efetivar o ataque (MUNIZ & LAKHANI, 2013).

2.5.2 Caixa branca

Testes de caixa branca consiste no conhecimento prévio do alvo, onde o atacante tem detalhes sobre processos de negócio, informações sobre a rede, conhecimento do código fonte do sistema. Este tipo de teste é focado em um objetivo específico na origanização, tendo em vista o já conhecimento de várias áreas que possam ser atacadas (MUNIZ & LAKHANI, 2013).

2.5.3 Caixa cinza

É o tipo de teste que é considerado de caixa preta e caixa branca. Neste teste o atacante possui alguns detalhes com relação ao alvo, outras informações importantes são ainda mantidas em sigilo, sendo necessário o efetivo ataque para desberta das informações. (MUNIZ & LAKHANI 2013).

2.6 Ferramentas de ataque

Para realização de ataques a aplicações *web* é necessário utilização de alguns softwares. Será listado algumas ferramentas utilizadas por atacantes para obter dados das aplicações *web*.

2.6.1 Kali Linux

Kali Linux é uma distribuição Linux baseada no Debian e possui vários softwares para auditoria de segurança de computadores, além de ser considerado o sucessor do BackTrack, Sistema Operacional Linux baseado no Ubuntu e que também é focado em testes de segurança e penetração (KALI, 2019).

Esse sistema operacional, apresenta ferramentas para análise de vulnerabilidades, scan de portas de servidores, detecção de vulnerabilidades em aplicações *web*, ataques às senhas como força bruta, ferramentas de *sniffer* para monitorar e analisar o tráfego de rede, injeção de código malicioso, entre outras, contando com mais de 600 ferramentas de teste de intrusão.

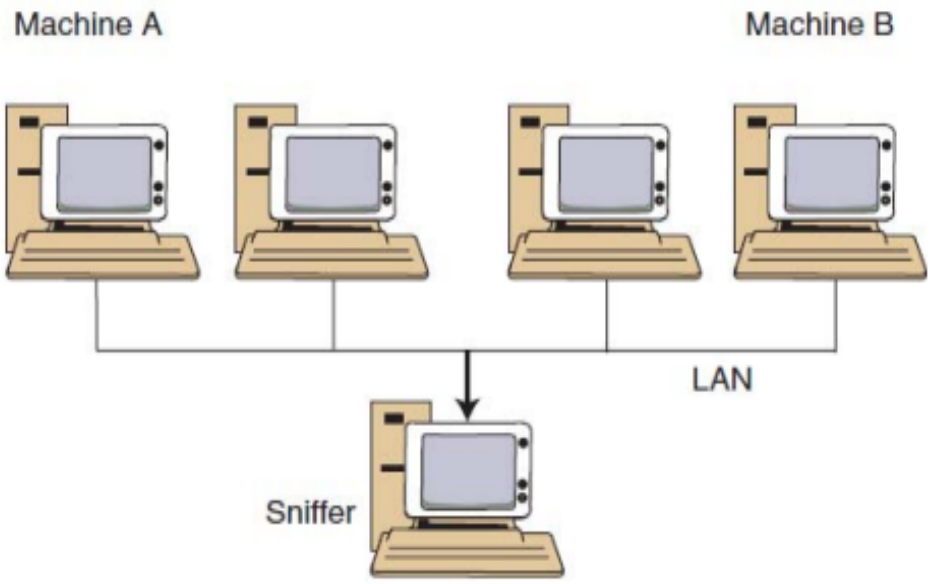
2.6.2 SQLMAP

Uma ferramenta para verificação da vulnerabilidade de *SQL Injection* é o SQLMAP que automatiza o processo de busca e detecção de falhas de injeção. O SQLMAP vem com mecanismos de testes de penetração contra sistema de arquivos e execução de comandos no Sistema Operacional e também invasão de banco de dados de impressões digitais. O comando básico do SQLMAP é: *sqlmap -u URL -função* (MUNIZ & LAKHANI, 2013).

2.6.3 Sniffing

A técnica de utilizar uma ferramenta para interceptação e captura dos dados que trafegam na rede de computadores podendo realizar a decodificação e mudança do conteúdo transmitido entre computadores, é o *Sniffing*. Uma vez que os dados são capturados, os crackers podem obter os dados que desejarem, como configurações de roteador, sessões de bate-papo, tragefos da *web*, entre outros (ANU & VIMALA, 2017).

Figura 2.5: Um *Sniffer* em uma rede local



Fonte: (ANU & VIMALA, 2017)

O Wireshark é um *sniffer* pré-instalado no Kali Linux e é um analisador de rede open source bastante utilizado para solução de problemas de rede e monitoração de trafego para capturar informações, como por exemplo, *tokens* de sessão. Outro *sniffer* bastante útil é o TCPReplay que pode classificar o trafégo como cliente ou servidor, e enviar pacotes para destinos específicos na rede. (MUNIZ & LAKHANI, 2013).

2.6.4 EditThisCookie

O EditThisCookie é uma extensão capaz de gerenciar os cookies no navegador, permitindo adicionar, remover, alterar e procurar estes pacotes de dados. Uma vez instalado o EditThisCookie permite gerenciar os dados obtidos durante a navegação na internet fornecendo um painel para interação, podendo ser possível trocar de uma navegador da *web* para outro utilizando a adição de um novo *cookie* no *Plug-in* (CAPANO, 2014).

2.6.5 Postman

O Postman é um gerenciador de API que é possivel projetar, simular, documentar, testar e monitorar APIs. Realizar ataques a APIs já criadas com valores predefinidos é uma das caracteristicas do Postman. O Postman era utilizado em 2012 no Google Chrome como uma extensão, após ser popularizado ganhou uma versão executavel e atualmente é usado por 7 milhões de desenvolvedores, além de 300 mil empresas (ASTHANA et al., 2019).

Capítulo 3

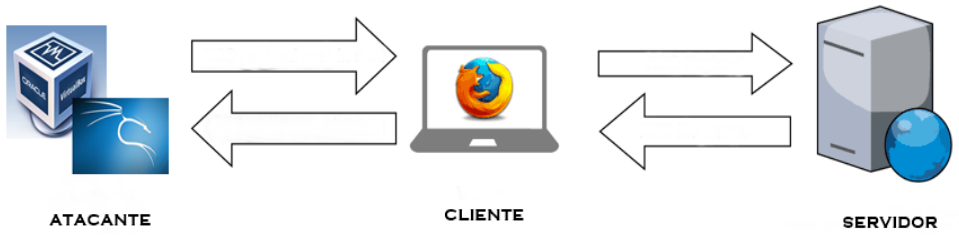
Experimentos e resultados

Para demonstrar as vulnerabilidades apresentadas pelo OWASP Top Ten e as técnicas de prevenção disponibilizadas pelo *framework* Laravel, foi desenvolvido alguns exemplos de sites em que determinadas vulnerabilidades podem acontecer.

Para o desenvolvimento das aplicações e verificações de testes foram utilizados os seguintes softwares: Sistema Operacional Ubuntu 18.04.2 LTS como cliente, a IDE Atom 1.35.1, a linguagem de programação PHP 7.2, o servidor *web* Apache 2.4, o banco de dados MySQL 14.1, o gerenciador de dependências Composer 1.8, *Framewok* Laravel 5.8, a biblioteca HTTP/PHP Guzzle 6.0, navegador Chromium 75.0, navegador Mozilla Firefox 66.0

Na execução dos ataques as aplicações *web*, foram utilizados o software de virtualização VirtualBox 6.0 para a criação de uma maquina virtual e o Kali Linux 2019.1 como sistema operacional e as ferramentas para ataques as aplicações desenvolvidas: Plugin EditThisCookie, *Sniffing* Wireshark 2.6, software de *pentest* SQLMAP 1.3 e gerenciador de API Postman 7.2.

Figura 3.1: Ataque a uma aplicação *web* através do Kali Linux



Fonte: O AUTOR

A figura 3.1 é uma representação do processo de ataque as aplicações *web* desenvolvidas neste projeto, sendo utlizado um servidor web (Apache), cliente (Navegador Mozilla Firefox/-

Chromium) e um atacante (Maquina virtual com S.O. Kali Linux). Nas seções a seguir será demonstrado cada vulnerabilidade de acordo com sua classificação de risco segundo a OWASP.

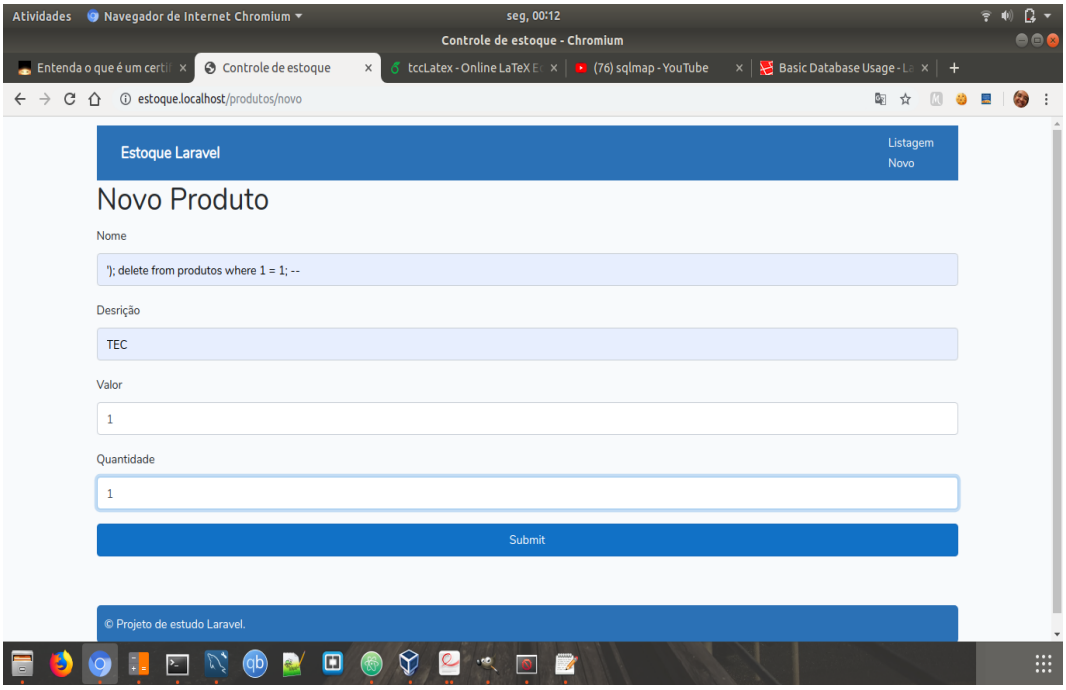
3.1 A1:Injeção

3.1.1 Cenário

Foi desenvolvido um sistema de estoque para verificação da vulnerabilidade de injeção utilizando a linguagem SQL e efetuado um ataque ao banco de dados através da ferramenta SQLMAP para explorar a SQLi.

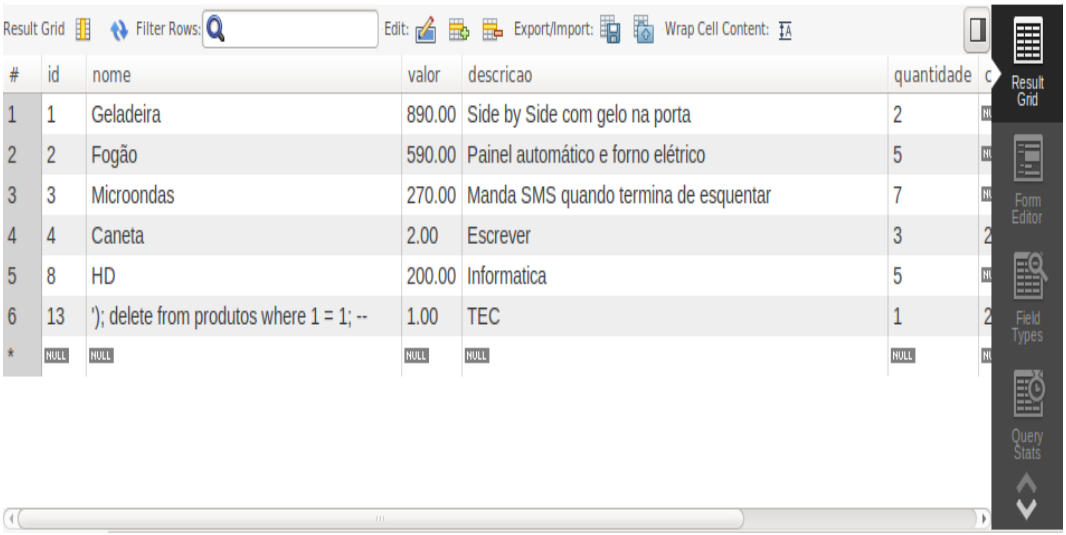
No sistema de estoque foi adicionado um produto com código malicioso em um dos campos de entrada (Figura 3.2). Após adicionar o produto é verificado que o comando SQL não afetou o Banco de Dados conforme apresentado na figura 3.3.

Figura 3.2: Criação de um produto passando código malicioso



Fonte: O AUTOR

Figura 3.3: Visualização do BD após inserção de código malicioso



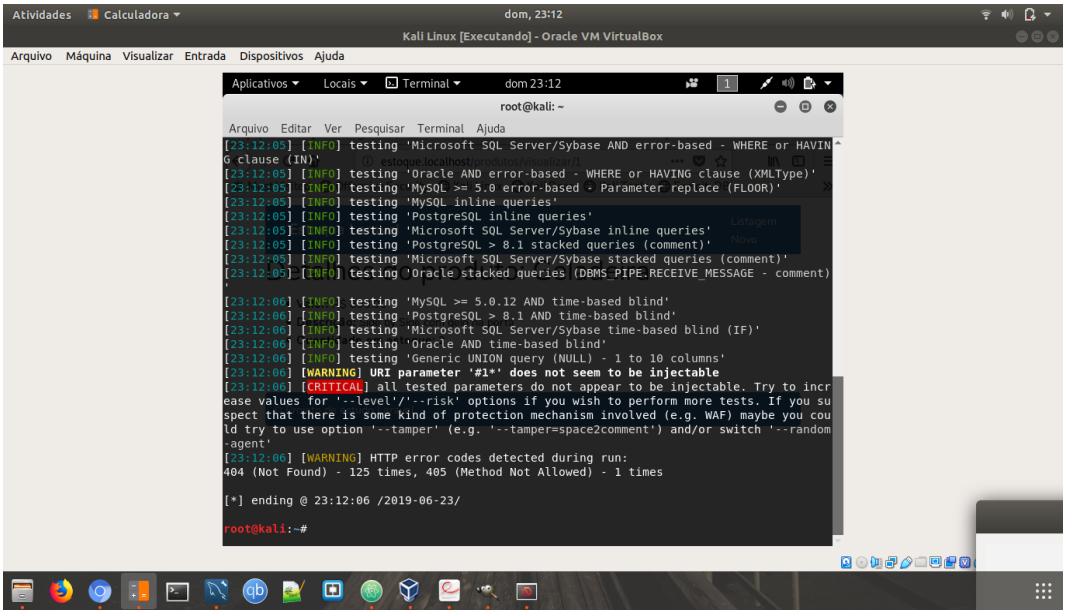
#	id	nome	valor	descricao	quantidade
1	1	Geladeira	890.00	Side by Side com gelo na porta	2
2	2	Fogão	590.00	Painel automático e forno elétrico	5
3	3	Microondas	270.00	Manda SMS quando termina de esquentar	7
4	4	Caneta	2.00	Escrever	3
5	8	HD	200.00	Informatica	5
6	13	'); delete from produtos where 1 = 1; --	1.00	TEC	1
*	NULL	NULL	NULL	NULL	NULL

Fonte: O AUTOR

Foi realizado um ataque ao banco de dados do sistema de estoque utilizando o Sistema Operacional Kali Linux e a ferramenta SQLMP para a vulnetabilidade de SQLi.

Ao realizar o ataque utilizando o SQLMAP e passando o site de estoque, que está na máquina local, como parâmetro (*sqlmap -u http://estoque.localhost/produtos/visualizar/1 –dbs*), foi observado que o ataque não é concluído devido, além do Laravel ter mecânismos de proteção contra este ataque que é o caso do uso do *Eloquent ORM*, o *framework* em questão utiliza URLs cujo parâmentros fazem parte da rota (*path param*) *estoque/produtos/visualizar/1* e não parâmetros de busca: *estoque/produtos/visualizar?id=1*.

Figura 3.4: Erro no ataque ao sistema de estoque - SQLMAP



Fonte: O AUTOR

3.1.2 Resultado

O *Eloquent ORM* implementa o *ActveRecord*, o que significa que uma única classe (Model) é responsável por interagir com a tabela de forma completa, podendo inserir, atualizar e exluir novos registros (STAUFFER, 2017).

Com o uso do *Eloquent ORM* o Laravel usa uma ligação de parâmetro PDO para evitar *SQL Injection*, garantindo que usuários mal-intencionados não venham modificar a consulta e assim afetar o BD.

Código 3.1: Código de criação de um Produto utilizando o *Eloquent ORM*

```
1 $produto = new Produto();
2 $produto->nome = Request::input('nome');
3 $produto->valor = Request::input('valor');
4 $produto->descricao = Request::input('descricao');
5 $produto->quantidade = Request::input('quantidade');
6
7 $produto->save();
```

Fonte: O AUTOR

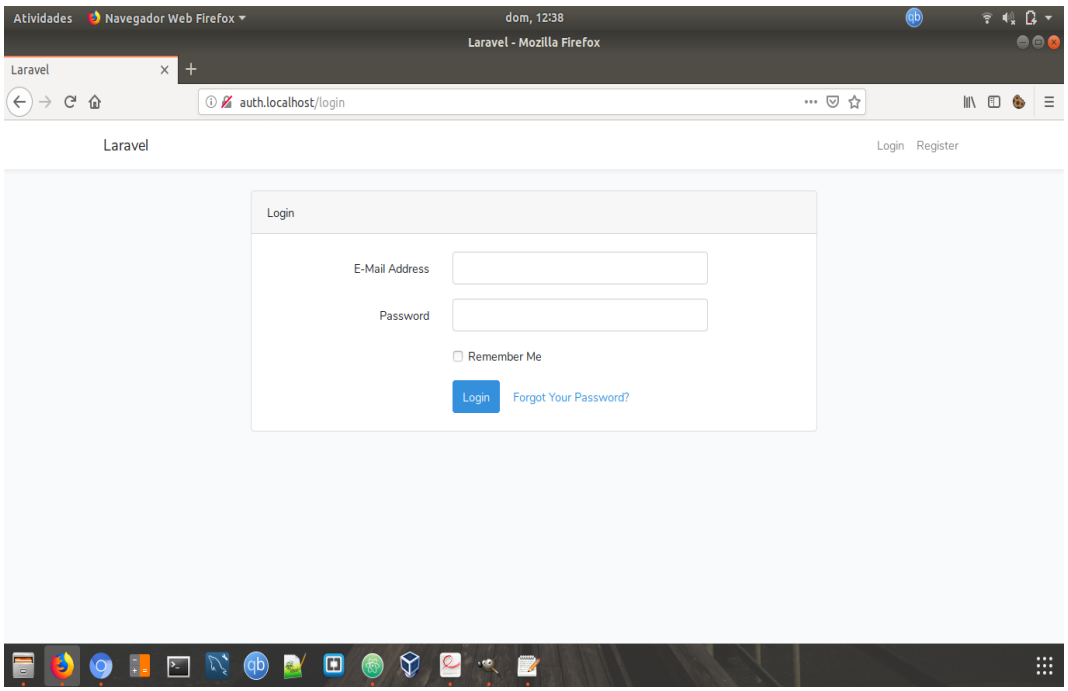
Pra tentativas de penetração de SQL utilizando ferramentas como o SQLMAP, o ataque não é concluído, devido, além do uso do *Eloquent ORM*, o Laravel faz uso de parâmetros de rota e não parâmetros de busca em suas rotas de solicitações e respostas, bloqueando assim acesso a parâmetros injetáveis.

3.2 A2:Quebra de autenticação

3.2.1 Cenário

Utilizando um site onde o usuário realiza o cadastro de suas anotações será verificado a vulnerabilidade de quebra de autenticação. Neste exemplo o usuário terá acesso a uma página de login:

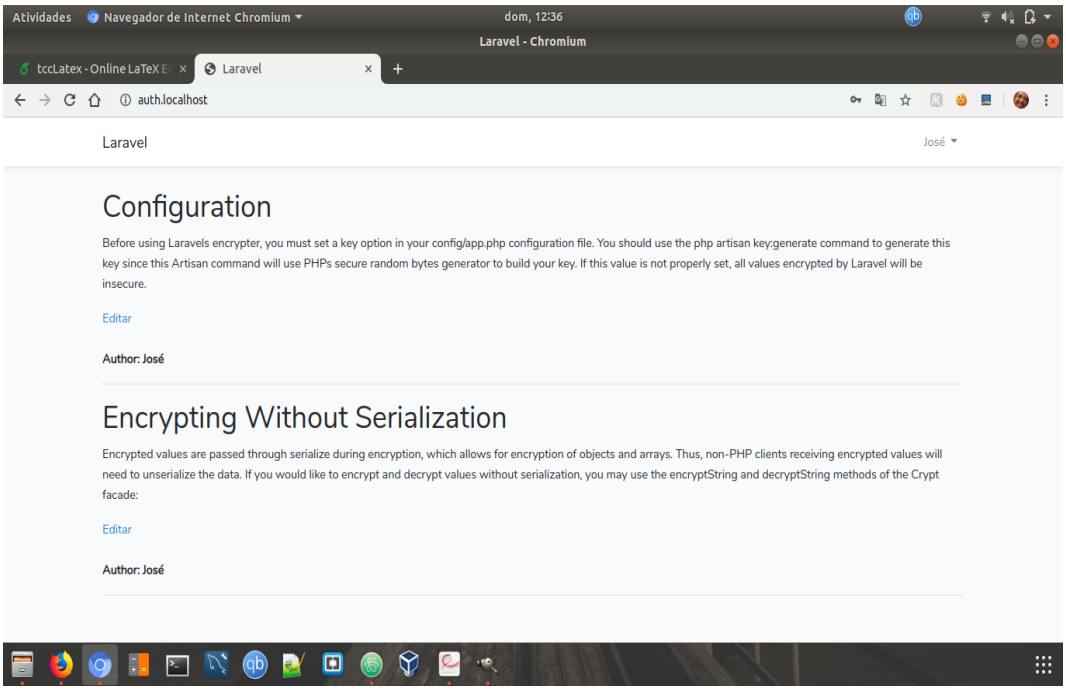
Figura 3.5: Página de login



Fonte: O AUTOR

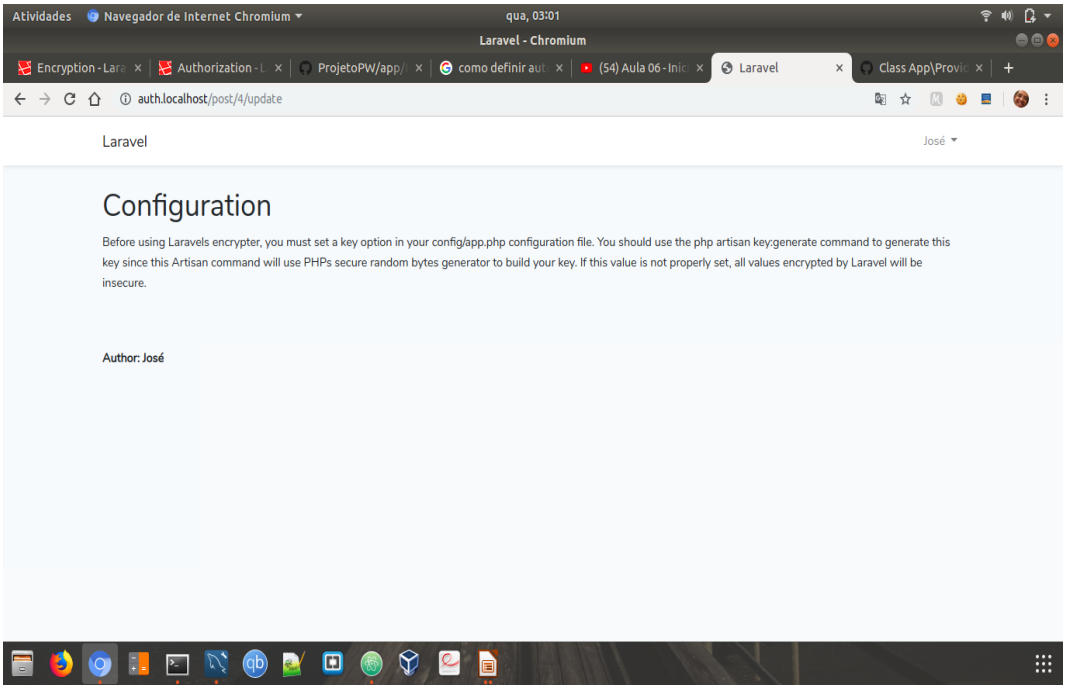
Após a realização do login, cada usuário terá sua página de acesso, conforme a figura 3.6. E também a opção de alterar o texto, caso assim deseje. A edição do texto é possível através do link "Editar"no fim da anotação e direcionará o usuário para outra página de visualização e edição, figura 3.7.

Figura 3.6: Usuário logado em sua página de anotações - Mozilla Firefox



Fonte: O AUTOR

Figura 3.7: Tela de Edição da primeira anotação do usuário José

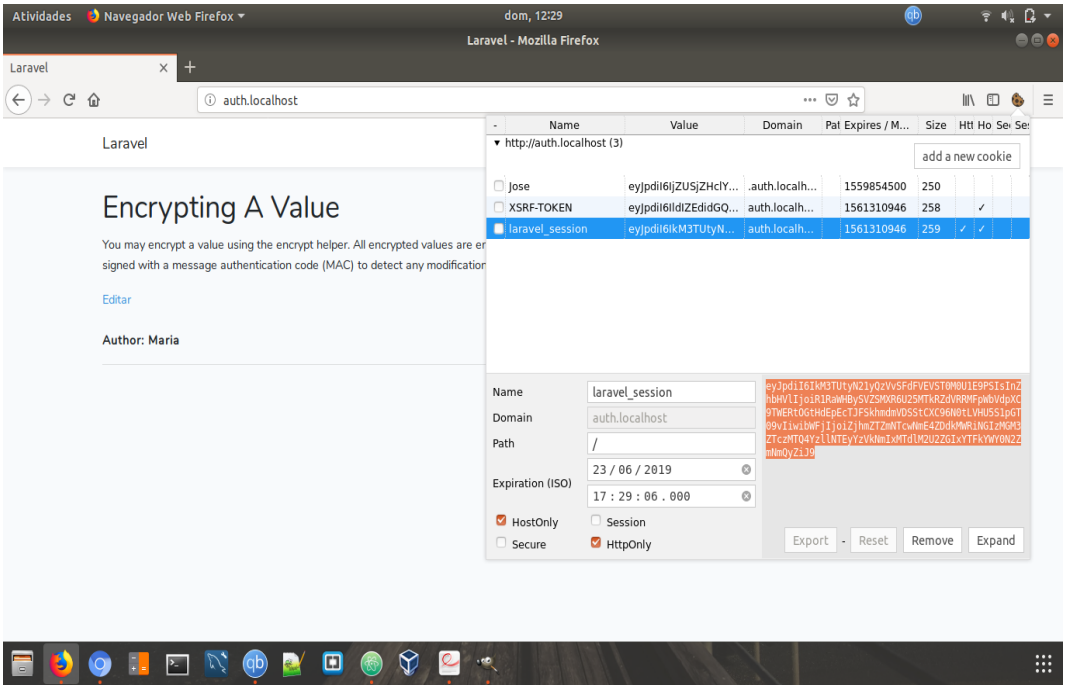


Fonte: O AUTOR

O plugin *Edit This Cookie* é um gerenciador de *cookies* e será utilizado na verificação desta vulnerabilidade para realizar a edição de um *cookie* de sessão de uma página web onde há um usuário logado.

Para realizar o teste, foram criados dois usuários para o site de anotações: Maria e José. A usuária Maria realiza logon no site através do *browser* Mozilla Firefox. O usuário José realiza logon no mesmo site através do *browser* Chromium. Após os usuários estarem logados é possivel realizar a visualização e também copiar o *cookie* de sessão do usuário logado, utilizado o *plugin* EditThisCookie, conforme a figura 3.8.

Figura 3.8: Captura do *cookie* de sessão da no *browser* Mozilla Firefox

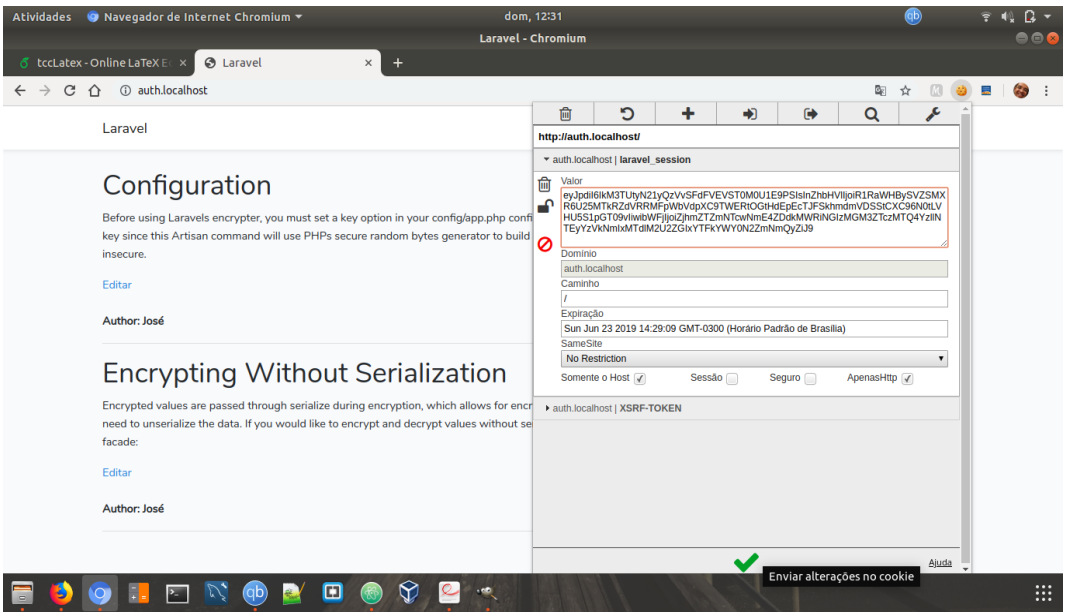


Fonte: O AUTOR

Na página de logon do usuário José foi utilizado o *plugin* EditThisCookie, disponível também para o navegador Chromium, para alterar o *cookie* de sessão do mesmo pelo *cookie* de sessão do logon da usuária Maria, conforme a figura 3.9.

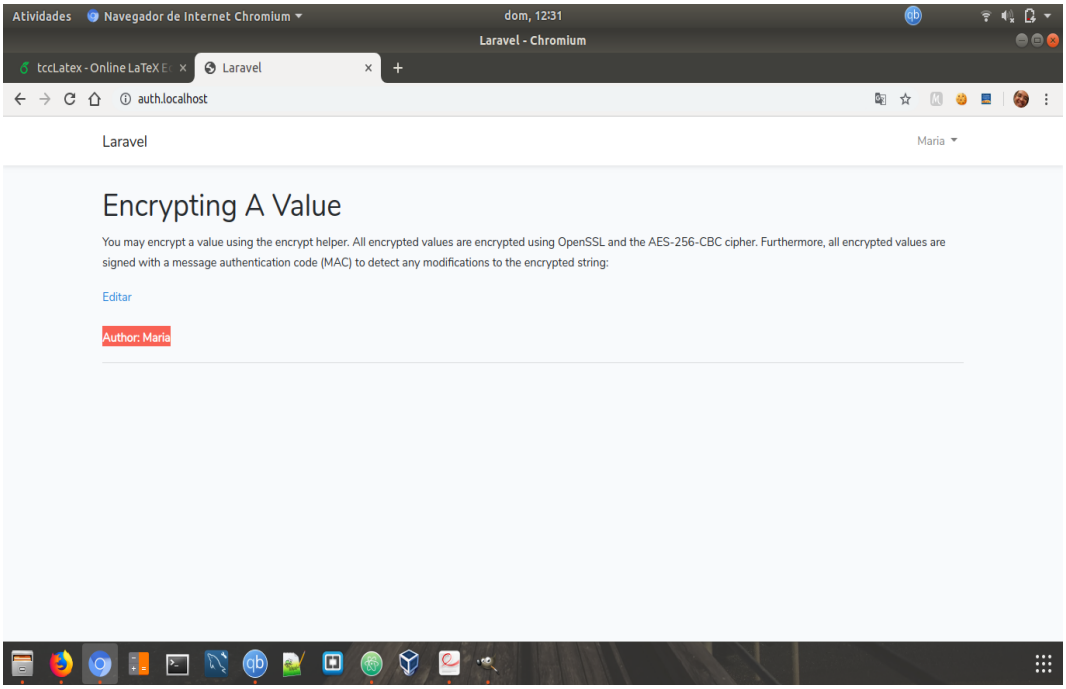
Após a alteração do *cookie* de sessão e atualização da página web é possível visualizar o resultado na figura 3.10

Figura 3.9: Edição do *cookie* de sessão no *browser* Chromium



Fonte: O AUTOR

Figura 3.10: Roubo de Sessão após alteração do *cookie*



Fonte: O AUTOR

3.2.2 Resultado

Após a realização dos testes descritos no cenário acima, é possível verificar que através do *plugin* EditThisCookie ocorre o roubo de sessão no *framework* Laravel.

A versão do Laravel até a realização deste experimento não abrange mecanismos de defesa contra quebra de autenticação, permitindo desta forma que atacantes adquiram identidades de outros utilizadores através de ataques de roubo de sessão para realizar as alterações que desejarem.

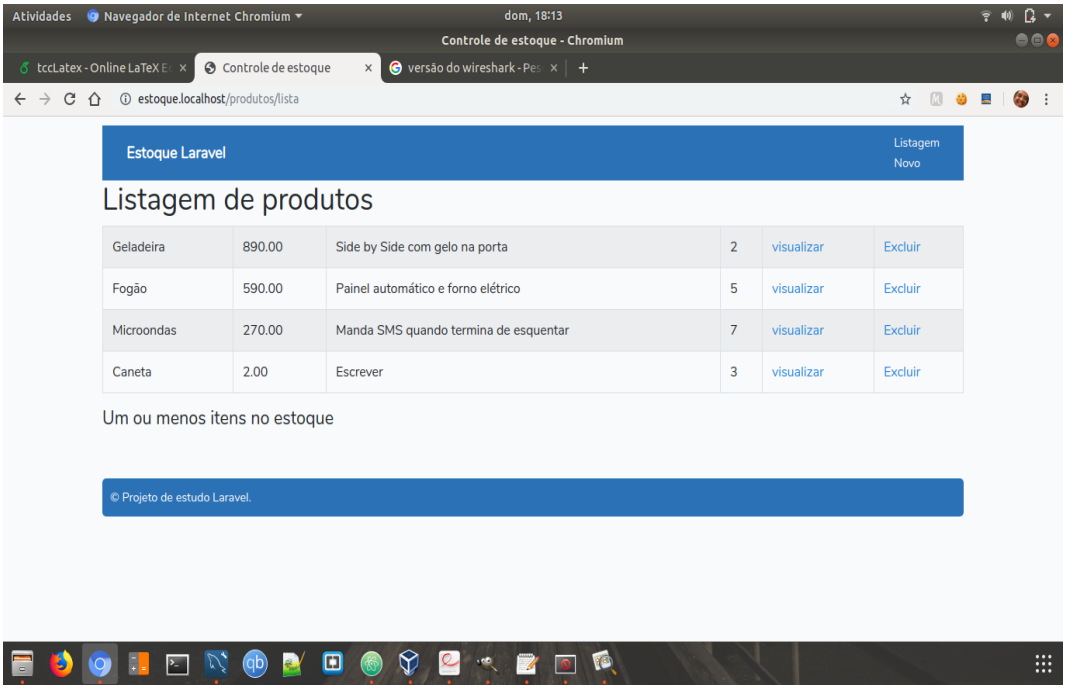
3.3 A3:Exposição de dados sensíveis

3.3.1 Cenário

Para a vulnerabilidade A3, Exposição a Dados Sensíveis, foi utilizado um site de estoque de produtos para verificação do armazenamento dos dados sensíveis e também se os dados estão sendo transmitidos em claro.

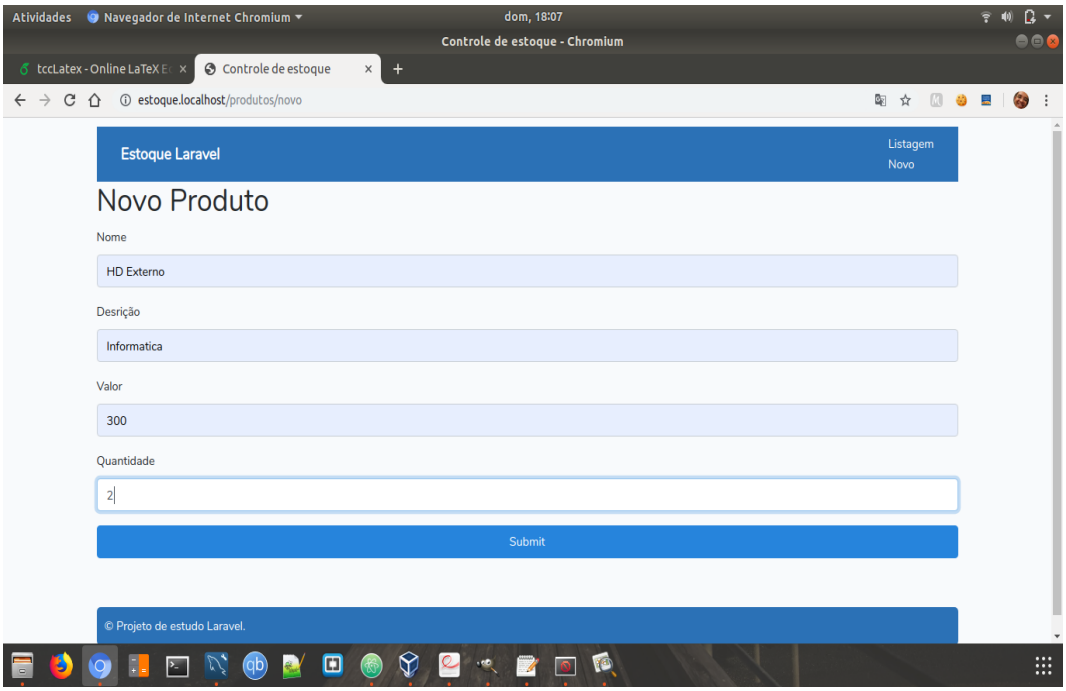
Na figura 3.11 é possível verificar os produtos já adicionados no estoque. Na mesma página há um link no lado superior direito para a inserção de um novo produto que ao clicar é direcionado para uma página onde é necessário preecher as informações e em seguida salvar para que o produto sejá adicionado ao estoque. Figura 3.12.

Figura 3.11: Produtos adicionados ao estoque



Fonte: O AUTOR

Figura 3.12: Tela de cadastro de um produto



Fonte: O AUTOR

Foi definido que o atributo "descrição"do produto seria armazenado de forma criptografada para verificação se o *framework* Laravel armazena as informações de forma segura.

Código 3.2: Código fonte da criação de um produto encriptado no projeto Estoque

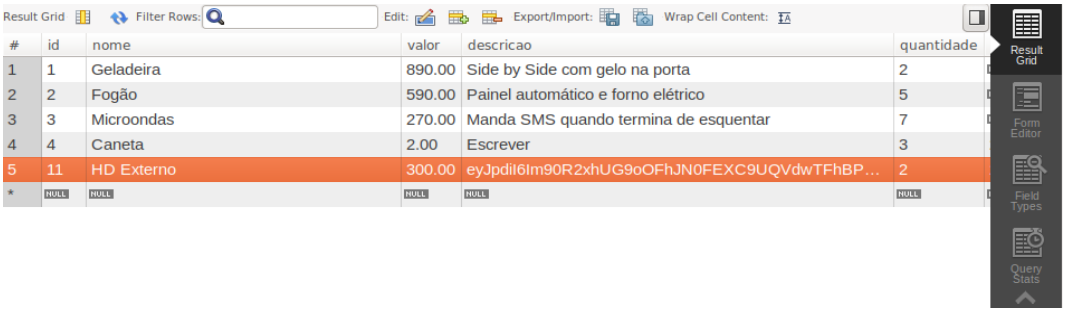
```php
$produto = new Produto();
$produto->nome = Request::input('nome');
$produto->valor = Request::input('valor');
$produto->descricao = encrypt(Request::input('descricao'));
$produto->quantidade = Request::input('quantidade');

$produto->save();
```

Fonte: O AUTOR

Após a inserção do produto, pode-se verificar no banco de dados, figura 3.13, que o atributo "descrição"foi adicionado de forma segura, onde o dado foi encriptado para seu armazenamento.

Figura 3.13: Visualização do produto no Banco de Dados

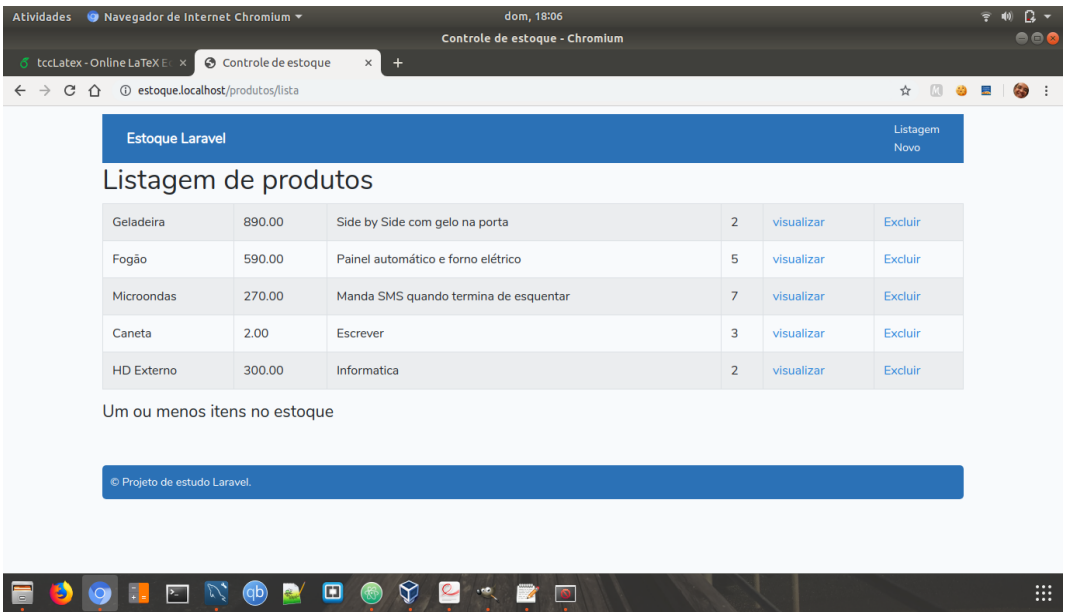


#	id	nome	valor	descricao	quantidade
1	1	Geladeira	890.00	Side by Side com gelo na porta	2
2	2	Fogão	590.00	Painel automático e forno elétrico	5
3	3	Microondas	270.00	Manda SMS quando termina de esquentar	7
4	4	Caneta	2.00	Escrever	3
5	11	HD Externo	300.00	eyJpdiI6Im90R2xhUG9oOFhJN0FEXC9UQVdwTFhBP...	2
*	NULL	NULL	NULL	NULL	NULL

Fonte: O AUTOR

Para visualizar os produtos na listagem deve-se decriptar o dado, antes criptografado, para que todos os valores dos atributos do produto possam ser visualizados como foram inseridos.

Figura 3.14: Visualização de todos os produtos após a decriptação

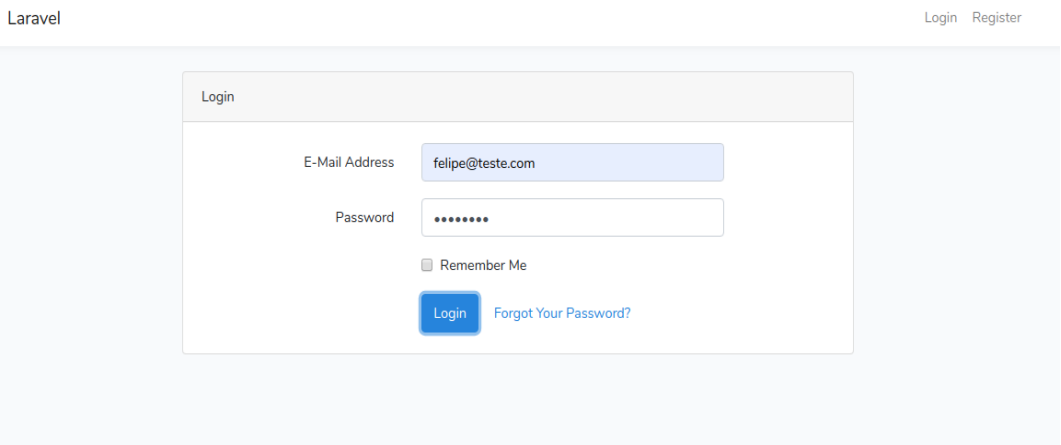


Fonte: O AUTOR

Para verificação se os dados estão sendo transmitidos de forma segura no Laravel, foi utilizado o software VirtualBox, o Sistema Operacional Kali Linux e o *Sniffing* Wireshark para analisar os pacotes enviados.

Para verificação da exposição de dados sensíveis em trânsito, foi realizado a captura de pacotes do usuário apartir do momento que é efetuado login na aplicação, figura 3.15. Em seguida foram filtrados os pacotes de rede enviados entre o servidor da aplicação e o navegador do cliente através do filtro "http"do Wireshark.

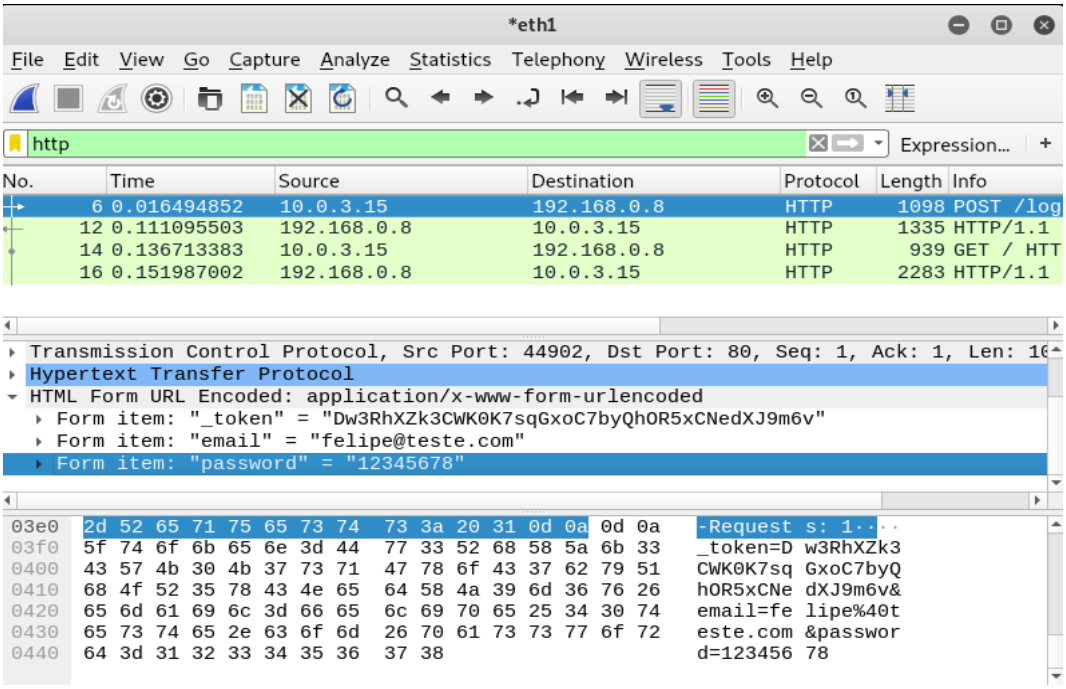
Figura 3.15: Login da aplicação



Fonte: O AUTOR

Conforme visto na figura 3.16, foi possível capturar os dados enviados do navegador para o servidor, e como observado os valores dos atributos login e senha do usuário logado estão visiveis.

Figura 3.16: Login da aplicação



Fonte: O AUTOR

3.3.2 Resultado

O Laravel tem a opção de encriptar e decriptar os dados através das funções *encrypt($var)* e *decrypt($var)*, respectivamente. Para tal é necessário realizar a definição da chave no arquivo de configurações do projeto Laravel, utilizando o comando *php artisan key:generate*. Se este valor não for definido, todos os valores criptografados do Laravel estarão inseguros.

A encriptação dos dados no Laravel é feita através do OpenSSL para fornecer criptografia AES-256 e AES-128. A documentação no Laravel recomenda a utilização dos recursos de criptografia integrados do Laravel, isto é devido aos valores criptografados serem assinados utilizando um código de autenticação MAC (*Message Authentication Code*) para detectar qualquer alteração na *string* criptografada. Se o MAC for identificado como inválido, não será possível descriptografar o dado.(OTWELL, 2019)

Os dados que estão em trânsito são capturados devido o envio das informações que está entre o navegador do usuário e o servidor serem enviados pelo protocolo de comunitação HTTP, o qual é *stateless*, ou seja, um protocolo que não guarda estado, que declara cada requisição como independente.

O envio destas informações independe da utilização do *framework* Laravel e sim do uso do protocolo HTTPS pelo desenvolvedor para que as informações tansmitidas entre cliente e servidor sejam enviadas de forma segura.

3.4 A4:Entidades externas de XML (XXE)

O Laravel não utiliza o XML de forma nativa. O padrão JSON é utilizado para as solicitações, serializações e *queries*. Na própria documentação do *framework* não há explicações sobre a aplicação do XML nos projetos e sim do recurso JSON.(OTWELL, 2019)

De acordo com a OWASP (2017), utilizar o padrão de JSON em lugar do XML é uma boa prática de segurança no desenvolvimento de aplicações *web*, evitando assim ataques XXE.

No entanto, é possível utilizar o padrão XML no Laravel através de pacotes de terceiros. Efetuado desta forma ataque a vulnerabilidade A9 - Utilização de Componentes Vulneráveis.

3.5 A5:Quebra de controle de acessos

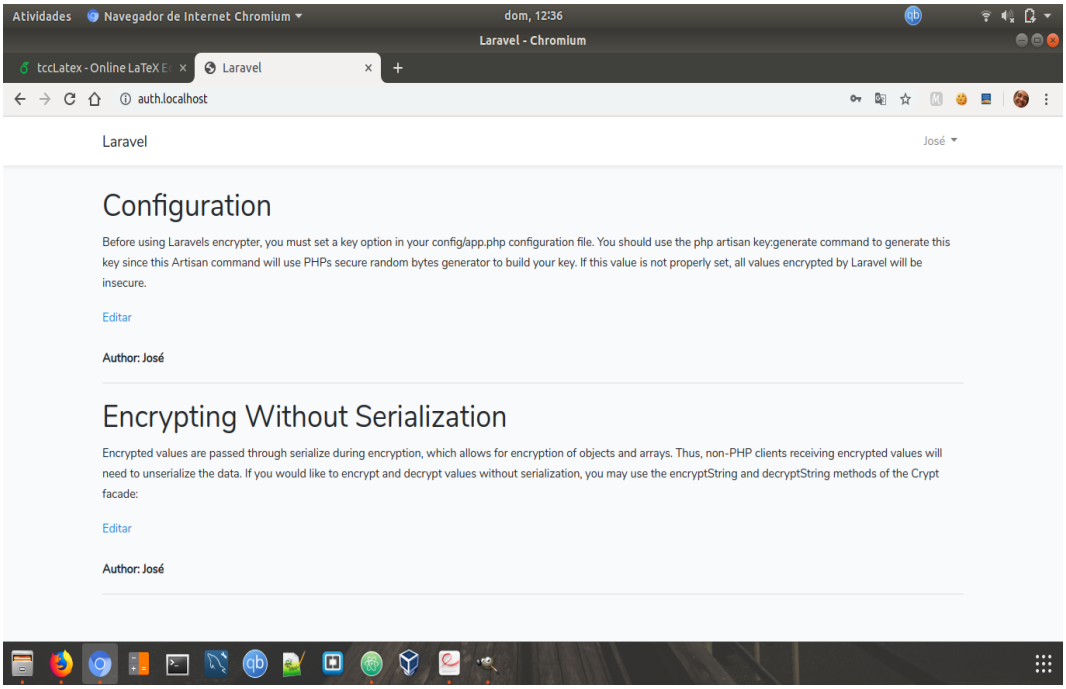
3.5.1 Cenário

Para verificação da vulnerabilidade de quebra de controle de acesso, foi utilizado o sistema anotações usado como exemplo na vulnerabilidade A2.

Neste sistema será demonstrado o que os utilizadores autenticados estão autorizados a fazer no sistema e como o Laravel garante a confidencialidade destas informações através dos seus recursos nativos.

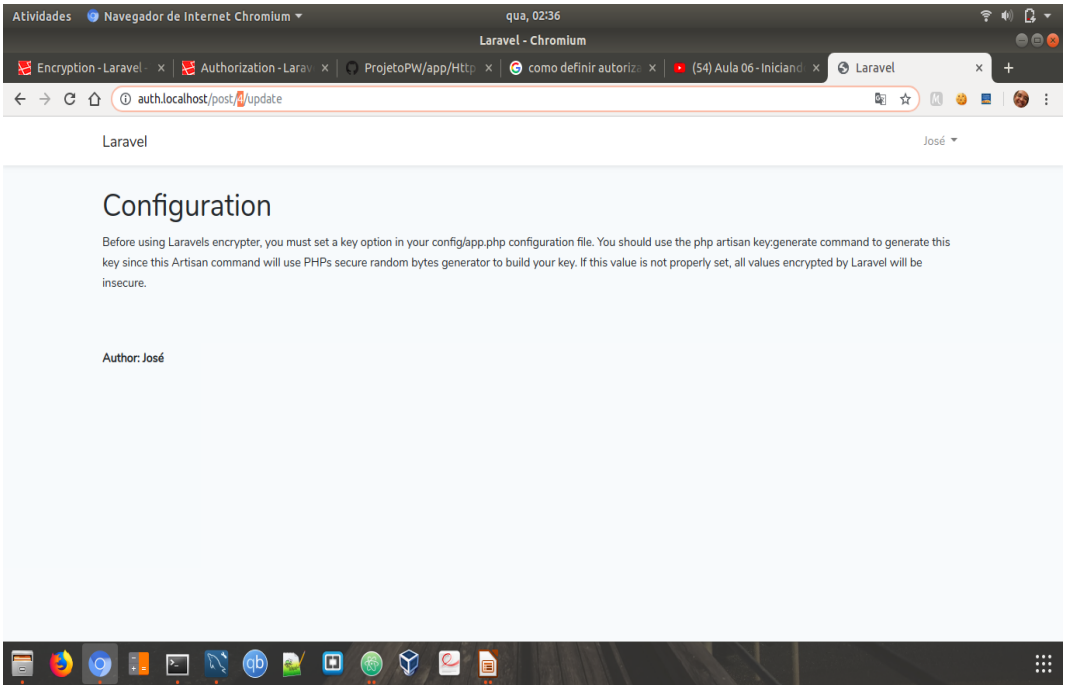
Após a realização do login na página inicial o usuário é direcionado para uma página que contém todas as anotações que ele criou (Figura 3.17) e a opção de realizar uma edição na anotação. Após clicar no link "Editar", a página aberta mosta na URL o índice daquele registro no BD. Sem o uso das regras do Laravel é possivel realizar a alteração do registro na URL e ter acesso a outro post de outro usuário. Figura 3.18.

Figura 3.17: Tela Inical do site de anotações após o login do usuário



Fonte: O AUTOR

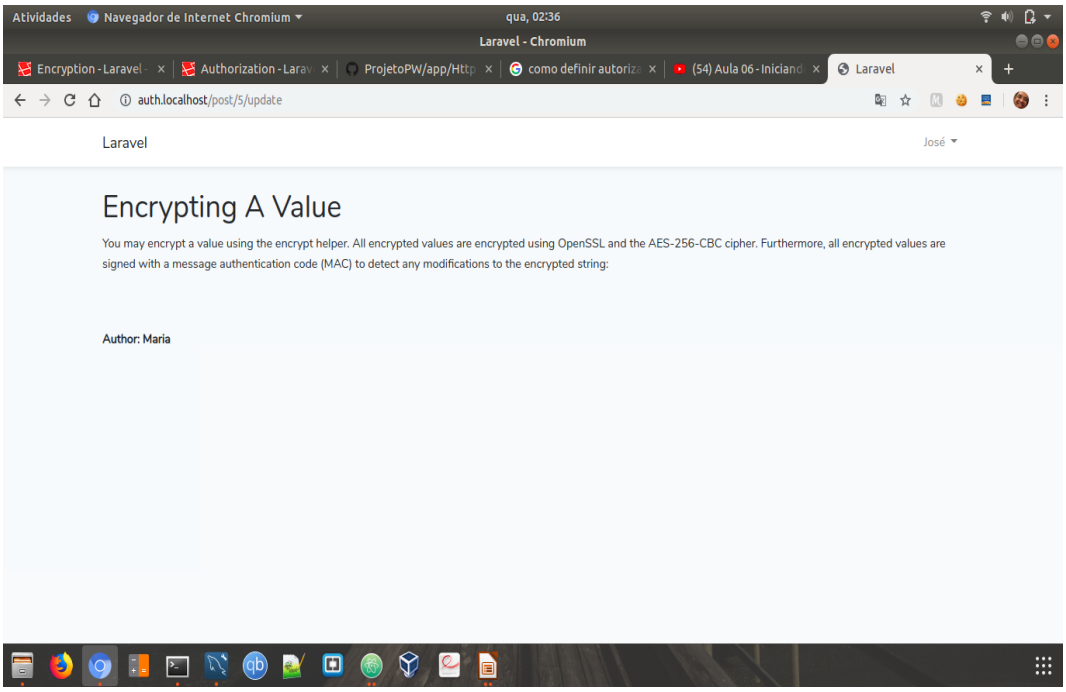
Figura 3.18: Tela de visualização de anotação para edição do usuário José



Fonte: O AUTOR

Após realizar a alteração do registro através da URL, o post de um usuário não logado é visualizado:

Figura 3.19: Tela de visualização de anotação de outro usuário após alteração do registro na URL



Fonte: O AUTOR

A partir da versão 5.2, o Laravel aborda um sistema de autorização, ACL - *Access control list* que após a definição de um *ability* (Regra de Autorização) permite acesso a determinado conteúdo apenas usuários com permissões.

Código 3.3: Código-fonte de exemplo da definição de um *ability*

```
1 public function boot(GateContract $gate)
2 {
3     $this->registerPolicies($gate);
4
5     $gate->define('update-post', function(User $user, Post $post)
6     {
7         return $user->id == $post->user_id;
8     });
9 }
```

Fonte: O AUTOR

O código 3.3 é referente a definição de um *ability* desenvolvido no método *boot* da classe *AuthServiceProvider*. Após a definição é necessário vincular a regra criada na edição da anotação no *controller* para que quando o usuário não autorizado tente acessar outro registro na URL em que não seja autorizado como visto na figura 3.18, venha ocorrer o *Error* 403, demonstrando

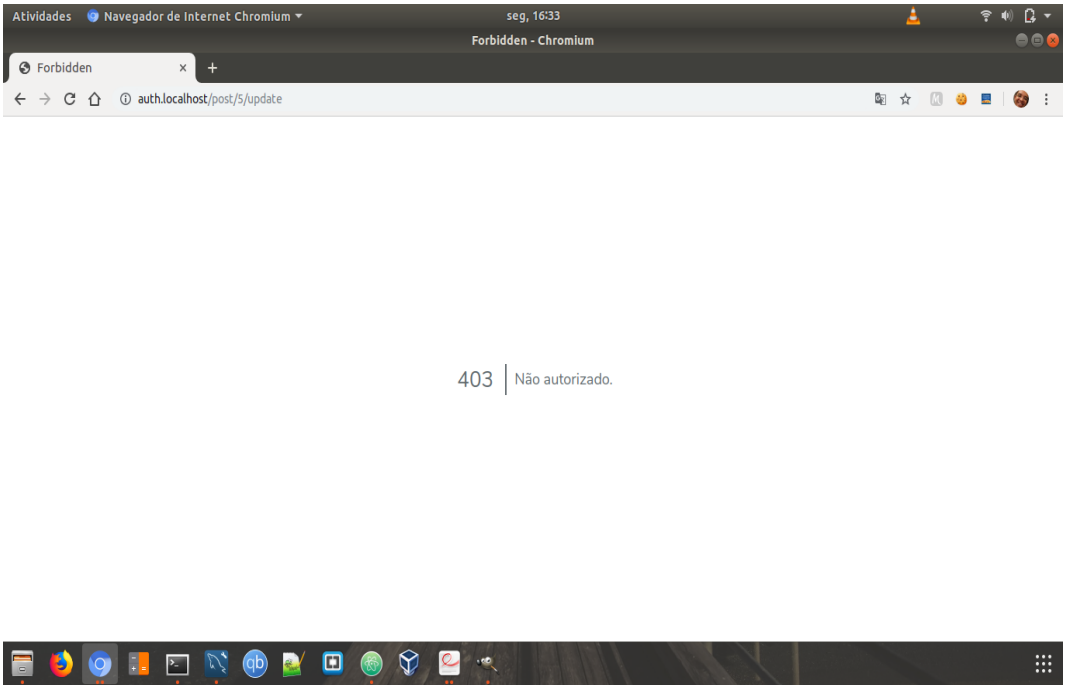
que o usuário não tem permissão para determinado recurso. A figura 3.20 apresenta o erro ao tentar visualizar registros sem permissões.

Código 3.4: Código-fonte do método *update* do *controller*

```
1  public function update($id)
2  {
3      $post = Post::find($id);
4
5      if(Gate::denies('update-post', $post))
6          abort(403, 'Nao autorizado.');
7
8
9      return view('updatePost', ['post' => $post]);
10 }
```

Fonte: O AUTOR

Figura 3.20: Erro ao acessar conteúdo não autorizado



Fonte: O AUTOR

3.5.2 Resultado

Através da lista de controle de acesso (ACL) o Laravel consegui controlar as permissões dos usuários definindo regras, que no framework é definido como *ability* (OTWELL, 2019).

O controle de autorizações no Laravel é realizado através da *Facade Gate*, *helpers* nos *controlllers*, e *middlewares* através do modelo *User*, assim também como algumas diretivas do *Blade* (STAUFFER, 2017).

Para a criação da regra de autorização é necessário dois itens: uma chave em modo *string* (Exemplo: *update-post*) e um *closure* que retorna um *boolean*. O *closure* deve conter o usuário autenticado, primeiro parâmetro, e os demais parâmentros serão os objetos que serão verificados os acessos. Exemplo de uma regra de autorização (*ability*) está na figura 3.3

Logo, o Laravel garanti segurança contra a vulnerabilide de Quebra de Controle de Acesso dependendo apenas dos desenvolvedores implementarem as regras de autorizações para garantir a integridade dos dados.

3.6 A6:Configurações de segurança incorretas

Não compreende as configurações de segurança apenas ao *framework* Laravel, pois o mesmo está interligado com outros softwaes para seu funcionamento completo, como por exemplo banco de dados, servidores *web*, *frameworks*, maquinas virtuais.

O Laravel pode utilizar-se de configurações internas para maior proteção das aplicações, como a definição da chave de criptografia para assegurar a encriptação e decriptação dos dados, outro exemplo é a desabilitação do modo *debug* quando o sistema estiver em produção evitando a apresentação da estrutura do sistema. No entanto, algumas configurações podem acarretar na vulnerabilidade da aplicação para vários ataques como por exemplo: a apresentação de mensagens de erros da aplicação sem nenhum filtro divulgando informações sensíveis, os cabeçalhos HTTP mal configurados, a senha de acesso ao banco de dados definida com a senha padrão ou uma senha fraca dando permissão de acesso ao superusuário, a má configuração do domínio podendo facilitar a quebra de autenticação do usuário.

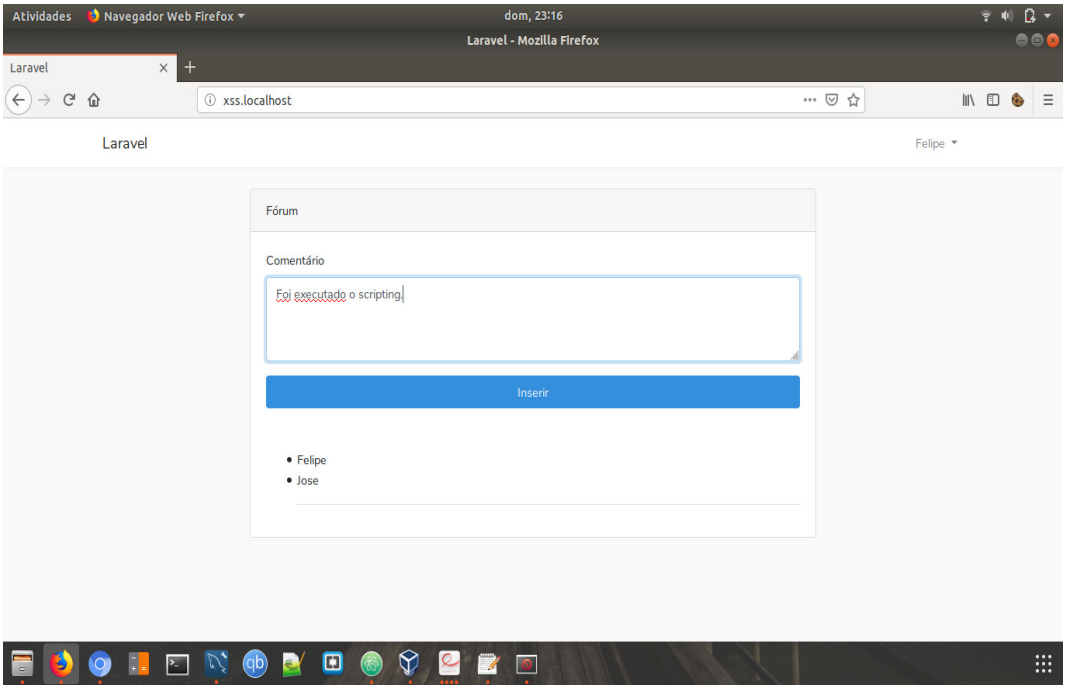
Logo, a proteção desta vulnerabilidade independe do uso do *framework* Laravel, tendo em vista que as configurações definidas podem comprometer a aplicação contra algum ataque externo através de análise em outros softwares interligados com o *website*.

3.7 A7:Cross-Site scripting (XSS)

3.7.1 Cenário

Foi desenvolvido um pequeno site de Fórum, onde o usuário realiza o login e pode adicionar um comentário e os demais usuários que acessarão o Fórum poderão visualizar os comentários antes inseridos.

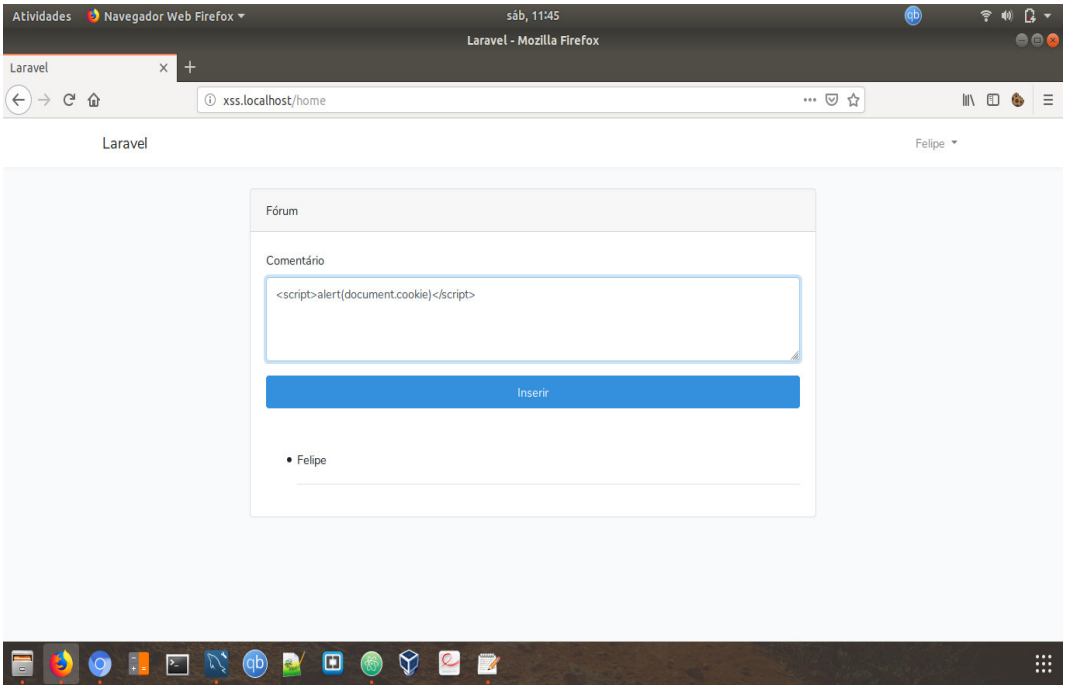
Figura 3.21: Tela principal do site para testes do XSS



Fonte: O AUTOR

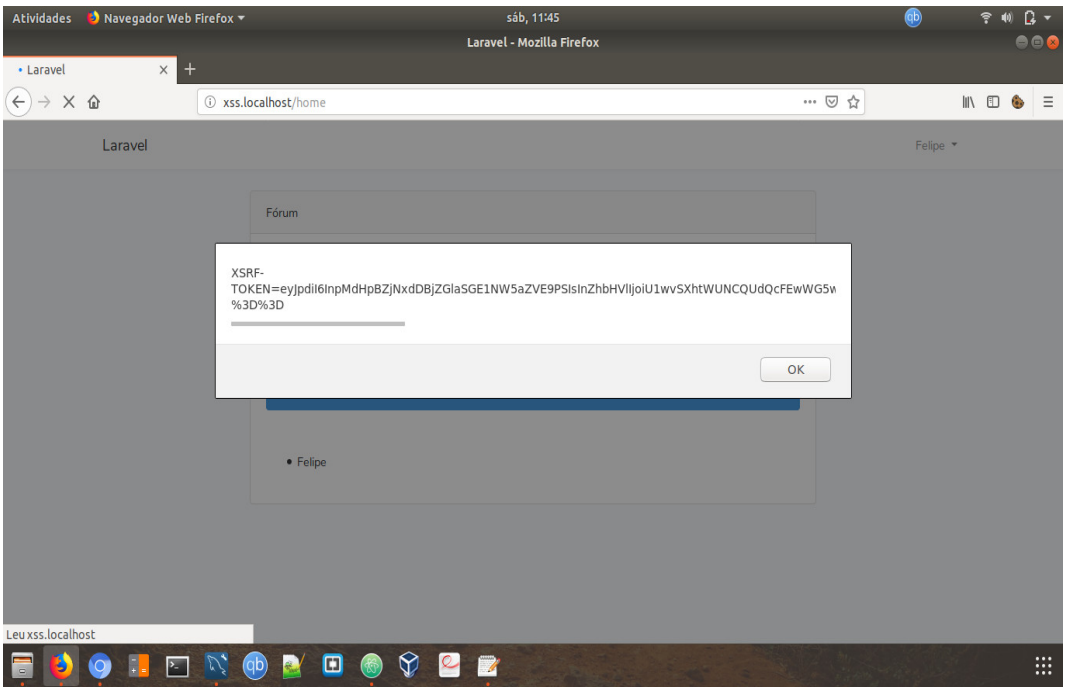
Ao inserir um comentário utilizando TAGs HTML, é possível verificar a execução de *scripts* no navegador caso o desenvolvedor não tenha filtrado os caracteres especiais que poderiam ser executados nos inputs da aplicação. Isso acontece devido o Laravel permitir a execução de código PHP diretamente nas *Views* sem uso de um *template*.

Figura 3.22: Introdução de *Script* no campo de entrada da aplicação Fórum



Fonte: O AUTOR

Figura 3.23: Execução do *Script* para obtenção do *cookie* da aplicação



Fonte: O AUTOR

Após a inserção do código malicioso no *input* do fórum, é possível verificar o *cookie* da aplicação, conforme a figura 3.23

O laravel possuí um *template engine* chamado *Blade*, que possui uma sintaxe concisa e um poderoso modelo de herança e extensibilidade.

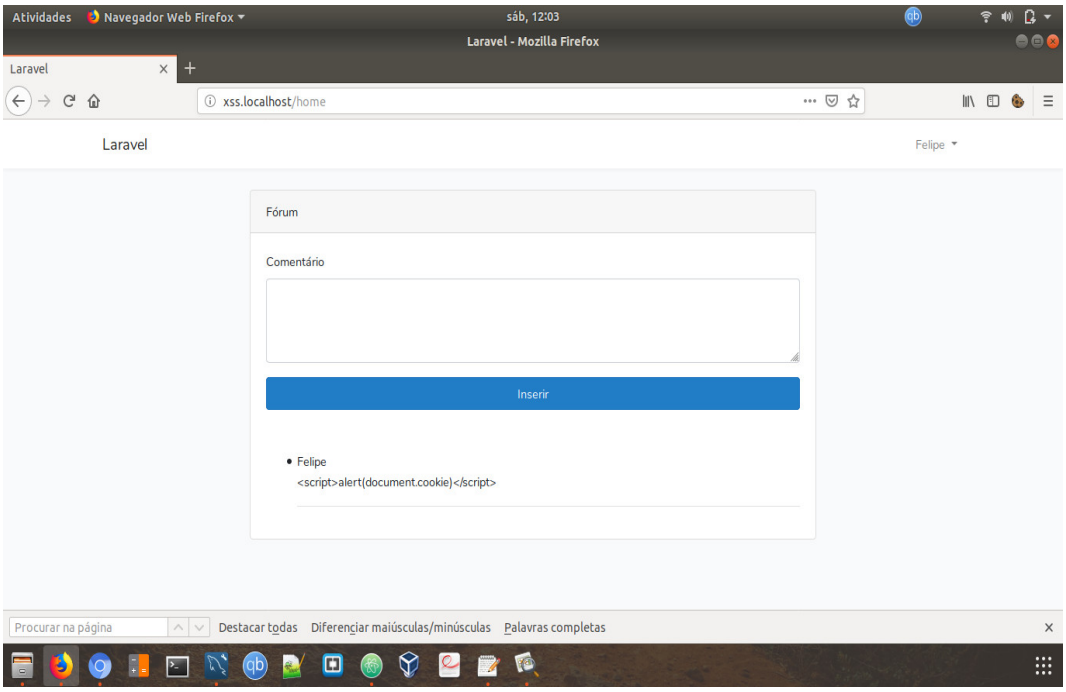
O *blade* introduz uma convenção em que suas tags, chamadas de "diretivas", são iniciadas com '@' e as chaves duplas de abertura e fechamento são utilizadas para ecoar as variáveis (*{{ $variavel }}*) (STAUFFER, 2017).

Código 3.5: Código *Blade* sendo definido como padrão para execução do comentário inserido no fórum

```
1 <ul>
2     @foreach ($usuarios as $u)
3       <li>{{ $u->name }}</li>
4       <!-- <?= $u->comment ?> -->
5       {{ $u->comment }}
6     @endforeach
7     <hr/>
8 </ul>
```

Fonte: O AUTOR

Figura 3.24: Execução do *Script* para obtenção do *cookie* da sessão após substituir código PHP pelo *template Blade*



Fonte: O AUTOR

O *blade* escapa tudo que é ecoando utilizando o padrão *htmlentities* do PHP, isso com o intuito de proteger o usuário contra *scripts* maliciosos. Com isto pode-se verificar no código 3.6 que o texto malicioso inserido é visualizado com caracteres de escape.

Código 3.6: Código-fonte da página onde os caracteres de *scripts* inseridos são substituídos por caracteres de escape

```
1  <ul>
2      <li>Felipe</li>
3          <!-- <script>alert(document.cookie)</script> -->
4          &lt;script&gt;alert(document.cookie)&lt;/script&gt;
5
6      <li>Jose</li>
7          <!--  -->
8      <hr/>
9  </ul>
```

Fonte: O AUTOR

3.7.2 Resultado

Após a execução do *script* malicioso inserido nos campos de entrada da aplicação e dependedo de como ela foi desenvolvida, o código inserido poderá ficar armazenado no banco de

dados, e toda vez que executar a página o código malicioso será executado.

No entanto, todos os *cookies* criados pelo Laravel são criptografados e assinados com código de autenticação não sendo possível ler ou modificar por parte do cliente (OTWELL, 2019). Logo a obtenção do *cookie* através de *scripts* para roubo de sessão, não é permitido no Laravel.

O *Blade* garante que os caracteres especiais inseridos no *input* do site sejam ecoados com caracteres de escape, isto porque o *blade* utiliza o padrão *htmlentities* do PHP. Logo o código *{{ $variavel }}* é equivalente a *<? htmlentities($variavel) ?>*.

No código 3.5, a linha 4 é referente a execução para visualizar o comentário inserido. Da forma que se encontra, está sujeito a ataques XSS. Ao utilizar o *Blade*, linha 5, o *template* gera saidas de escape, não permitindo assim a execução de *scripts*, protegendo a aplicação contra XSS.

3.8 A8:Desserialização insegura

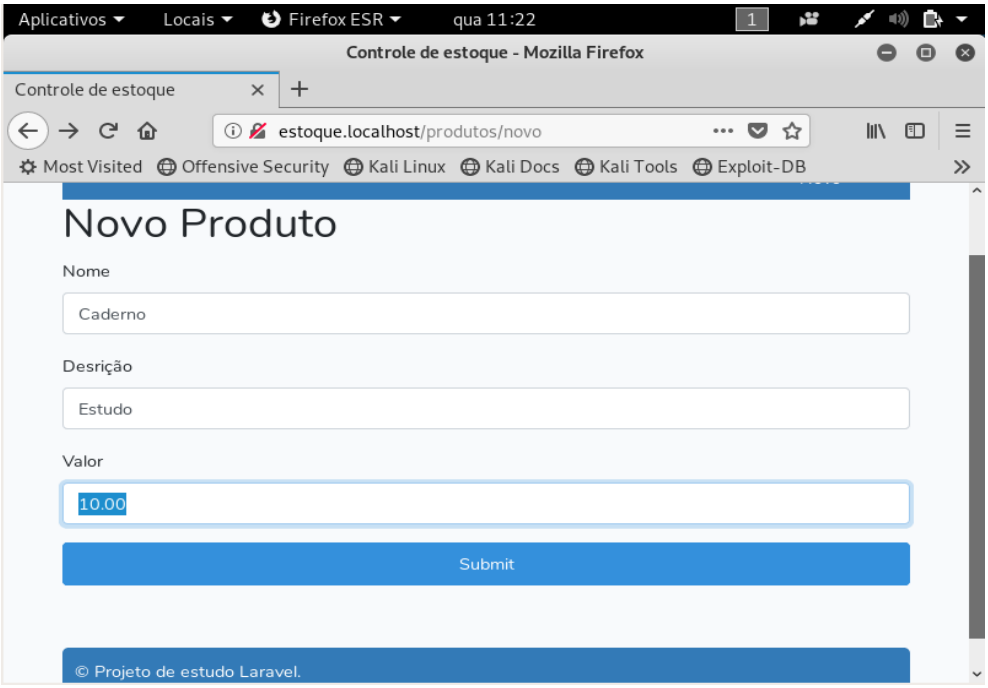
3.8.1 Cenário

Foi realizado a criação de uma API para funções de cadastrar, recuperar, atualizar e deletar um objeto produto. Para uso desta API foi utilizado o projeto de estoque, apresentado na avaliação da vulnerabilidade de Injeção, para realizar as operações nela definidas.

No desenvolvimento de APIs é comum realizar a serialiação para o formato que será transportado pela rede. Há alguns formatos de serialização utilizados, detre eles estão o JSON, XML e YAML. O Laravel faz uso do JSON para serialização e envio dos dados e os seus controladores de recursos já estão estruturados com os padrões e verbos REST (STAUFFER, 2017).

Nesta avaliação de vulnerabilidade é cadastrado um produto no sistema de estoque através da API criada utilizando o Guzzle, biblioteca HTTP para PHP, para assim efetivar o armazenamento do objeto no banco de dados.

Figura 3.25: Tela de cadastro de um produto

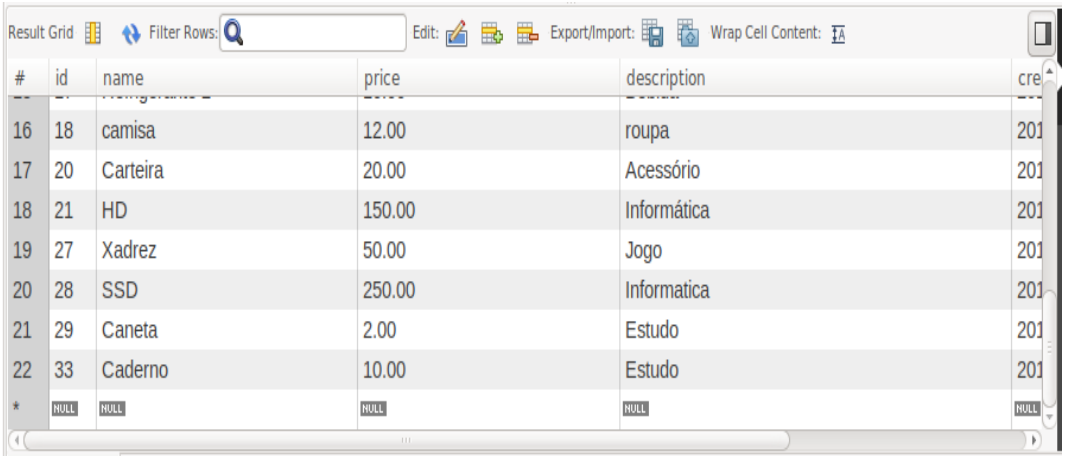


Fonte: O AUTOR

.

Após o cadastro do produto pode ser verificado no banco de dados que o objeto foi inserido corretamente, figura 3.26. No entanto, no momento da inserção através da API no sistema de estoque, o software Wireshark foi utilizado para verificar os dados trafegados na rede e foi possivel capturar os valores armazenados nos atributos do produto que estão serializados em JSON (Figura 3.27).

Figura 3.26: Armazenamento do produto no banco de dados

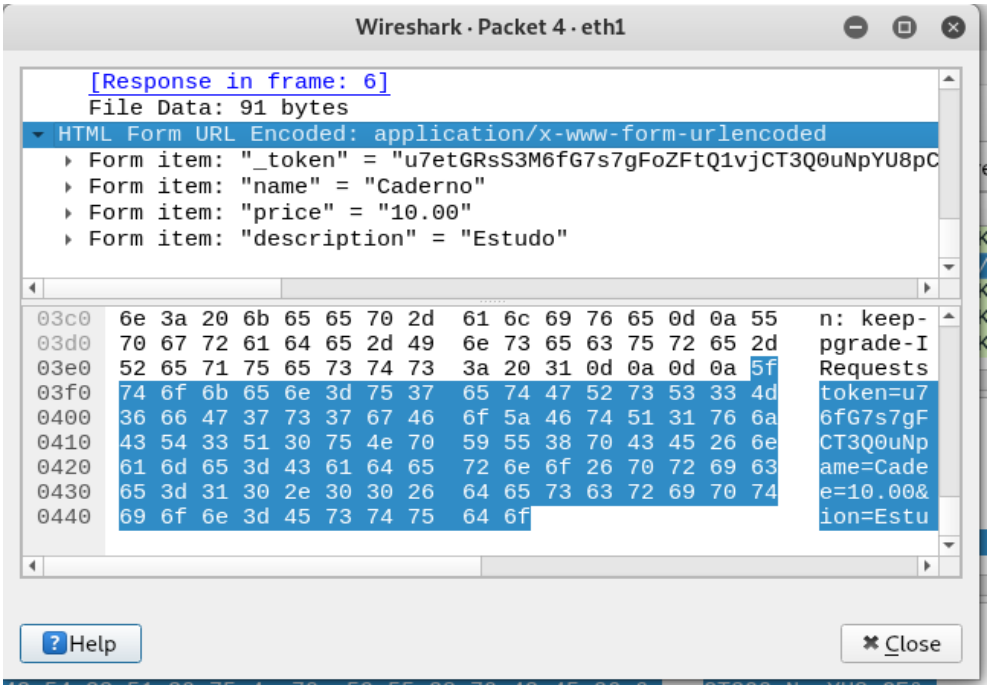


#	id	name	price	description	cre
16	18	camisa	12.00	roupa	201
17	20	Carteira	20.00	Acessório	201
18	21	HD	150.00	Informática	201
19	27	Xadrez	50.00	Jogo	201
20	28	SSD	250.00	Informatica	201
21	29	Caneta	2.00	Estudo	201
22	33	Caderno	10.00	Estudo	201
*	NULL	NULL	NULL	NULL	NULL

Fonte: O AUTOR

.

Figura 3.27: Captura dos valores serializados através do Wireshark



Fonte: O AUTOR

Após adquirir os valores enviados através da API, foi realizado a inserção do mesmo produto através do Postman, uma ferramenta que fornece suporte a requisições de API. Em seguida, Foi observado no banco de dados (Figura 3.28) que o objeto foi reinserido da mesma forma que foi capturado pelo wireshark, gerando uma duplicação de dados.

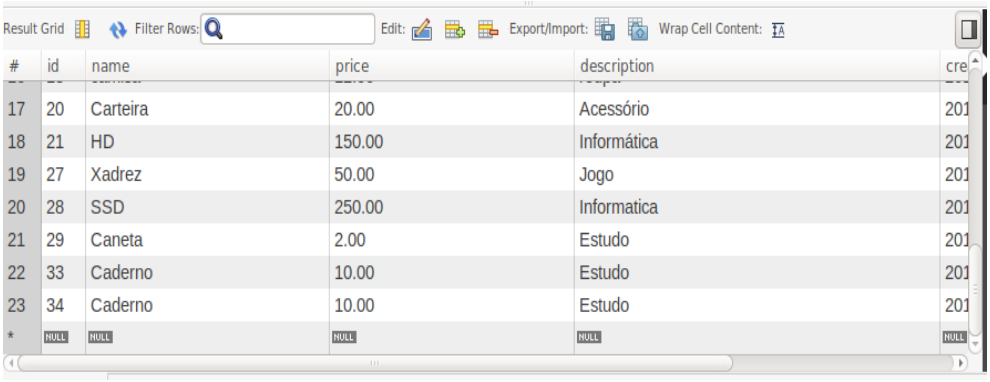
Código 3.7: Utilização da ferramenta Postman para duplicar a inserção do produto no banco de dados

```
{
    "data": {
        "current_page": 5,
        "data": [
            {
                "id": 29,
                "name": "Caneta",
                "price": 2,
                "description": "Estudo",
                "created_at": "2019-07-03␣14:21:42",
                "updated_at": "2019-07-03␣14:21:42"
            },
            {
                "id": 33,
                "name": "Caderno",
                "price": 10,
```

```
                "description": "Estudo",
                "created_at": "2019-07-03␣15:01:09",
                "updated_at": "2019-07-03␣15:01:09"
            },
            {
                "id": 34,
                "name": "Caderno",
                "price": 10,
                "description": "Estudo",
                "created_at": "2019-07-03␣18:38:47",
                "updated_at": "2019-07-03␣18:38:47"
            }
        ]
    }
}
```


Fonte: O AUTOR

Figura 3.28: Visualização do banco de dados após o uso da ferramenta Postman



#	id	name	price	description	cre
17	20	Carteira	20.00	Acessório	201
18	21	HD	150.00	Informática	201
19	27	Xadrez	50.00	Jogo	201
20	28	SSD	250.00	Informatica	201
21	29	Caneta	2.00	Estudo	201
22	33	Caderno	10.00	Estudo	201
23	34	Caderno	10.00	Estudo	201
*	NULL	NULL	NULL	NULL	NULL

Fonte: O AUTOR

.

3.8.2 Resultado

Como verificado no experimento, os dados serializados e enviados através de uma API Laravel, podem ser desserializados no trafego e visualizados, conseguindo o atancante ter acesso a informações sigilosas e efetuar até um novo ataque alterando os valores.

Estes ataques de desserialização podem ocorrer senão vierem de fontes seguras, mas se não for possivel verificar a origem destas fontes que utilizarão o sistema, uma maneira de proteção é realizar a autenticação dos dados entre a origem e o destino, garantindo desta forma a confidencialidade e integridade das informações.

Logo, o Laravel não provê mecanismos de segurança contra esta vulnerabilidade, sendo resposabilidade do desenvolvedor a implemetação de processos de proteção dos dados no envio dos mesmos pela rede, mesmo estando serializados.

3.9 A9:Utilização de componentes vulneráveis

O Laravel utiliza o *Composer* como gerenciador de pacotes e dependências. Vários projetos do Laravel utiliza componentes externos, onde são criados por instituições e usuários diversos. O Laravel possui até o presente desenvolvimento deste trabalho, 15.680 pacotes conforme (THUAU, 2019), disponibilizados para serem instalados.

Não sendo viável a realização dos testes em cada componente desenvolvido devido a quantidade existente. Um exemplo de um componente externo que pode estar vulnerável e comprometer o projeto se inserido no mesmo é o uso do XML no Laravel, podendo gerar a vulnerabilidade A4:Entidades Externas de XML (XXE) descrita na OWASP Top Ten. Logo o Laravel pode ser considerando vulnerável a vulnerabilidade A9.

3.10 A10:Registo e monitorização insuficiente

O registro e o monitoramento de ataques não compete ao *framework* realizar. O próprio desenvolvedor deve implementar estas funcionalidades na camada de aplicação para que assim possa detectar tentativas de invasão, diminuindo ou anulando as investidas do atacante.

3.11 Conclusão dos experimentos

Conforme os experimentos realizados sobre cada vulnerabilidade da OWASP Top Ten utilizando o *framework* Laravel, foi diagnoticado que o *framework* possui defesas contra algumas das vulnerabilidades descritas no OWASP Top Ten, entre elas estão a Injeção, Exposição a Dados Sensíveis, Entidades Externas de XML (XXE), Quebra de Controle de Acesso e *Cross-Site Scripting* (XSS).

Com relação as demais vulnerabilidades, para o Laravel alguns mecânismos de defesa não compotem ao *framework*, o que é o caso da vulnerabilidade A6 - Configuração de Segurança Incorretas, em que as aplicações que o *framework* interage para o seu completo funcionamento (Servidor, Banco de Dados) podem estar vulneráveis. E a vulnerabilidade A10 - Registro e Monitorização Insuficiente, o desenvolvedor deve realizar a devida verificação sem necessariamente uso do *framework*.

Para as vulnerabilidades de Quebra de Autenticação, Desserialização Insegura e Utilização de Componentes Vulneráveis o Laravel 5.8 não apesenta mecanismos de defesa.

Quadro 3.1: Análise das vulnerablidades web x Defesas internas no Laravel

OWASP Top Ten	Laravel - defesas		
	Sim	Não	N/A
A1:Injeção	X		
A2:Quebra de Autenticação		X	
A3:Exposição de Dados Sensíveis	X		
A4:Entidades Externas de XML (XXE)	X		
A5:Quebra de Controle de Acessos	X		
A6:Configurações de Segurança Incorretas			X
A7:Cross-Site Scripting(XSS)	X		
A8:Desserialização Insegura		X	
A9:Utilização de Componentes Vulneráveis		X	
A10:Registo e Monitorização Insuficiente			X

Fonte: O AUTOR

Capítulo 4

Conclusões

Este trabalho tem como objetivo fortalecer a importância de considerar aspectos de segurança durante o desenvolvimento das aplicações *Web*. As definições de segurança da informação e aplicações *web* foram apresentadas, permitindo identificar os componentes que manipulam informações sigilosas e portanto, devem ser protegidos contra usuários mal-intencionados.

Foram analisadas as dez vulnerabilidades presentes no projeto OWASP Top Ten 2017 e as consequências que cada uma pode causar em aplicações *web*.

Com o avanço de uso de tecnologias para agilizar o processo de desenvolvimento, muito desenvolvedores acabam fazendo uso de *frameworks*, tendo em vista que estes softwares já possuem pacotes e algumas dependências já estudadas e integradas para o bom funcionamento do projeto desenvolvido com ele. Através das vulnerabilidades descritas no Top Ten, foi realizado pesquisas no *framework* Laravel para examinar as técnicas de segurança presentes no *framework* para proteção das informações.

Foi desenvolvido alguns exemplos de sites utilizando o *framework* Laravel para assim realizar testes de penetração do tipo caixa preta. Para a vulnerabilidade de Injeção foi desenvolvido um site de estoque e realizado testes manuais na aplicação e também através da ferramenta SQLMAP. Foi identificado que o Laravel tem mecanismos de proteção contra esta vulnerabilidade através do uso do *Eloquent ORM* e da técnica de parâmentro de rotas. Para a vulnerabilidade de Quebra de autenticação, foi utilizado um sistema de anotações e através do gerenciador de *cookies* EditThisCookie foi possivel realizar o roubo de sessão, logo o Laravel não tem ferramentas de proteção contra esta vulnerabilidade.

A exposição a dados sensíveis é uma vulnerabilidade que compreende o armazenamento e o trafego das informações pela rede. O Laravel através de funções nativas de que realizam a encriptação e decriptação das informações traz proteção contra esta vulnerabilidade. O trafego dos dados não compete ao *framewok* Laravel a proteção e sim do protocolo de comunicação utilizado, como por exemplo HTTP ou HTTPS. Para a vulnerabilidade de XXE, o Laravel por padrão não utiliza XML e sim o JSON para serialização e solicitações, sendo protegido contra esta vulrabilidade. No entanto, é possível utilizar XML através de pacotes criados por terceiros,

mas não recomendado.

O Laravel possui desde a versão 5.2 um controle de autorização através do uso da ACL (*Access Control List*). Fazendo uso deste recurso e da criação de *ability* o *framework* permite acesso a determinados conteúdos apenas a usuários autorizados. Para a vulnerabilidade de Configurações de Segurança Incorretas não depende apenas do Laravel a devida proteção, considerando que o *framework* faz uso de outros softwares para o seu funcionamento e estes software podem apresentar falhas nas suas configurações e assim permitir ataques a aplicação.

Contra a vulnerabilidade de XSS foi utilizado um site de fórum para verificação de inserção de *scripts*. O Laravel faz uso de um *template engine* chamado *Blade* que garante que caracteres especiais inseridos em campos de entrada da aplicação sejam ecoados com caracteres de escape, não efetivando assim o ataque de XSS. O Laravel também possui proteção contra os *cookies*, pois os mesmos são criptografados e assinados com código de autenticação não sendo possível a leitura ou alteração por parte do cliente. O Laravel não faz proteção a desserialização insegura, tendo em vista que para o objeto ser enviado de forma segura, mesmo ele sendo serializado, faz-se necessário a implementação por parte do desenvolvedor de formas de autenticação entre a origem e o destino ou também registrar excessões no ato de desserialização sem autorização, alertando desta forma no destinatário que o pacote foi interceptado.

O *framework* estudado também faz uso de diversos componentes externos em vários projetos, componentes estes, criados por usuários e instituições diversas. Logo, devido a possibilidade de utilização de pacotes não testados e criados por qualquer usuário o *framework* é vulnerável a vulnerabilidade A9 (Utilização de Componentes Vulneráveis). Para a vulnerabilidade A10, referente a monitorização insuficiente e registro, não cabe ao Laravel realizar proteção contra esta vulnerabilidade, o desenvolvedor da aplicação deve implementar funções na camada de aplicação para reconhecer tentativas de invasão.

Com a popularização das aplicações Web e do uso corriqueiro de *frameworks* por parte dos desenvolvedores, a segurança se tornou um tópico de extrema importância para que os dados dos usuários não sejam expostos. Logo, a introdução de vulnerabilidades deve ser evitada desde o início do desenvolvimento da aplicação. Das dez vulnerabilidaes descritas no OWASP Top Ten o Laravel não fornece mecânismos de proteção contra cinco. Duas vulnerabilidades (A6, A10) não competem ao uso do *framework* a devida proteção tendo em vista fatores externos, e as outras três vulnerabilidaes (A2, A8, A9) o Laravel 5.8 não apresenta bloqueio de ataques. As outas cinco vulnerabilidades listadas no Top Ten (A1, A3, A4, A5, A7), o Laravel fornece recursos para proteção contra ataques mal-intencionados, cabendo aos desenvolvedores o uso correto dos recursos presente no *framework* para a segurança da informação.

Como o mercado de aplicações para *smartphones* tem crescido nos últimos anos devido as possibilidades que o aparelho pode proporcionar, será analisado em trabalhos futuros o estudo das principais vulnerabilidades presentes em aplicações *mobile* que são desenvolvidas através do *framework* da Google, Flutter, tanto para sistemas operacionais IOS quanto para Android. Tendo como referência a Mobile Top Ten, projeto desenvolvido pela OWASP que lista as prin-

cipais vulnerabilidades encontradas em aplicações para *smartphones*.

Referências bibliográficas

[1] ADERMANN, N. ; BOGGIANO, J. *Composer*. Disponível em: `https://getcomposer.org/`. Acesso em: 29 junho 2019, 2019.

[2] AMOROSO, E. G. *Fundamentals of Computer Security Technology ª Edição*. 1th. Prentice Hall, 1994.

[3] ANIF, M. ; DENTHA, A. ; SINDUNG, H. Designing internship monitoring system web based with Laravel framework. In *2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*. (2017), pp. 112–117.

[4] ANU, P. ; VIMALA, S. A survey on sniffing attacks on computer networks. In *2017 International Conference on Intelligent Computing and Control (I2C2)*. (2017), pp.

[5] ASTHANA, A. ; SOBTI, A. ; KANE, A. *Postman*. Disponível em: `https://www.getpostman.com/`. Acesso em: 01 julho 2019, 2019.

[6] BISHOP, M. *Introduction to Computer Security ª Edição*. 1th. Addison-Wesley Professional, 2004.

[7] CAPANO, F. *EditThisCookie*. Disponível em: `http://www.editthiscookie.com/`. Acesso em: 18 junho 2019, 2014.

[8] CERN *Where the web was born*. Disponível em: `https://home.cern/science/computing/birth-web/short-history-web`. Acesso em: 30 abril 2019, 2019.

[9] CHIAVENATO, I. *Gestão de pessoas : O novo papel dos recursos humanos nas organizaçõesª Edição*. 4th. Manole, 2014.

[10] CHU, G. ; LISITSA, A. Poster: Agent-based (BDI) modeling for automation of penetration testing. In *16th Annual Conference on Privacy, Security and Trust (PST)*. (2018), pp.

[11] DEHALWAR, V. ; KALAM, A. ; KOLHE, M. L. ; ZAYEGH, A. Review of Web-Based Information Security Threats in Smart Grid. In *7th International Conference on Power Systems (ICPS)*. (2017), pp. 849–853.

[12] GAUTHIER, F. ; MERLO, E. Fast Detection of Access Control Vulnerabilities in PHP Applications. In *19th Working Conference on Reverse Engineering*. (2012), pp. 247–256.

[13] GIL, A. C. *Como Elaborar Projetos de Pesquisa ª Edição*. 4th. Atlas, 2002.

[14] GORDON, S. R. ; GORDON, J. R. *Sistemas de Informação - Uma Abordagem Gerencial ª Edição*. 3th. LTC, 2006.

[15] ISO/IEC27002 *Tecnologia da Informação - Técnicas de Segurança - Código de Prática para a Gestão de Segurança da Informação ª Edição*. 1th. ABNT NBR, 2005.

[16] JAN, S. ; NGUYEN, C. D. ; BRIAND, L. Known XML Vulnerabilities Are Still a Threat to Popular Parsers and Open Source Systems. In *International Conference on Software Quality, Reliability and Security*, (2015), pp. 233–241.

[17] JOY, J. ; SINGH, D. P. A Generic Framework design to enhance capabilities of an Enterprise Test Automation Framework. In *2015 Conferência Internacional sobre Computação Aplicada e Teórica e Tecnologia da Comunicação (iCATccT)*. (2015), pp. 207–212.

[18] KALI *Kali Documentation*. Disponível em: `https://docs.kali.org/documentation`. Acesso em: 15 junho 2019, 2019.

[19] MCCOOL, S. *Laravel Starter ª Edição*. 1th. Packt Publishing, 2012.

[20] MINAYO, M. C. D. S. Análise qualitativa: teoria, passos e fidedignidade. *Ciência Saúde Coletiva* 17, 3 (2012), 621–626.

[21] MUNIZ, J. ; LAKHANI, A. *Web Penetration Testing with Kali Linuxª Edição*. 1th. Packt Publishing, 2013.

[22] OTWELL, T. *Laravel Documentation*. Disponível em: `https://laravel.com/docs/5.8`. Acesso em: 11 junho 2019, 2019.

[23] OWASP *OWASP Top Ten Project*. Disponível em: `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. Acesso em: 30 março 2019, 2017.

[24] PFLEEGER, C. P. ; PFLEEGER, S. L. ; MERGULIES, J. *Security in Computing ª Edição*. 5th. Prentice Hall, 2015.

[25] PMI, P. M. I. *Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®) ª Edição*. 5th. Project Management Institute, 2013.

[26] PONTES, E. L. G. *Praticando a Segurança da Informação ª Edição*. 1th. Brasport, 2008.

[27] SADEGHIAN, A. ; ZAMANI, M. ; MANAF, A. A. SQL Injection Vulnerability General Patch Using Header Sanitization. In *International Conference on Computer, Communication, and Control Technology (I4CT 2014)*. (2014), pp. 239–243.

[28] SHRIVASTAVA, A. ; CHOUDHARY, S. ; KUMAR, A. XSS Vulnerability Assessment and Prevention in Web Application. In *2nd International Conference on Next Generation Computing Technologies (NGCT-2016)*. (2016), pp. 850–853.

[29] SINGH, H. ; DUA, M. Detection prevention of website vulnerabilities: Current Scenario and Future Trends. In *2nd International Conference on Communication and Electronics Systems*. (2017), pp. 429–435.

[30] STAUFFER, M. *Desenvolvendo com Laravel^a Edição*. 1th. Novatec, 2017.

[31] STEWART, J. M. ; TITTEL, E. ; CHAPPLE, M. *CISSP Certified Information Systems Security Professional Study Guide ^{a} Edição*. 3th. Sybex, 2005.

[32] THUAU, J. *Packalyst - Laravel*. Disponível em: `https://packalyst.com/`. Acesso em: 24 junho 2019, 2019.

[33] TRENDS, G. *Google Trends*. Disponível em: `https://trends.google.com.br/trends/explore?geo=BR&q=%2Fm%2F0jwy148`. Acesso em: 29 junho 2019, 2019.