



**UNIVERSIDADE FEDERAL DO CEARÁ**  
**CAMPUS DE QUIXADÁ**  
**TECNÓLOGO EM REDES DE COMPUTADORES**

**DENYS PEREIRA MACIEL**

**DETECÇÃO E ANÁLISE DE VULNERABILIDADES EM UM SERVIDOR OJS**  
**UTILIZANDO TESTE DE PENETRAÇÃO**

**QUIXADÁ**

**2019**

DENYS PEREIRA MACIEL

DETECCÃO E ANÁLISE DE VULNERABILIDADES EM UM SERVIDOR OJS  
UTILIZANDO TESTE DE PENETRAÇÃO

Trabalho de Conclusão de Curso submetido à  
Coordenação do Curso Tecnólogo em Redes de  
Computadores da Universidade Federal do Ceará como  
requisito parcial para obtenção do grau de Tecnólogo.  
Área de Concentração: Computação

Orientador: Prof. Dr. Arthur de Castro Callado

Coorientador: Prof. Me. Roberto Cabral Rabêlo Filho

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação  
Universidade Federal do Ceará  
Biblioteca Universitária  
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

---

M138d Maciel, Denys Pereira.  
Detecção e Análise de Vulnerabilidades em um Servidor OJS utilizando Teste de Penetração / Denys Pereira Maciel. – 2019.  
58 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2019.  
Orientação: Prof. Dr. Arthur de Castro Callado.  
Coorientação: Prof. Me. Roberto Cabral Rabêlo Filho.

1. Segurança computacional. 2. Teste de intrusão . 3. Código aberto. I. Título.

CDD 004.6

---

DENYS PEREIRA MACIEL

DETECÇÃO E ANÁLISE DE VULNERABILIDADES EM UM SERVIDOR OJS  
UTILIZANDO TESTE DE PENETRAÇÃO

Trabalho de Conclusão de Curso submetido à  
Coordenação do Curso Tecnólogo em Redes de  
Computadores da Universidade Federal do Ceará como  
requisito parcial para obtenção do grau de Tecnólogo.  
Área de Concentração: Computação

Aprovada em: \_\_/\_\_/\_\_\_\_.

BANCA EXAMINADORA

---

Prof. Dr. Arthur de Castro Callado (Orientador)

Universidade Federal do Ceará - UFC

---

Prof. Me. Roberto Cabral Rabêlo Filho (Coorientador)

Universidade Federal do Ceará - UFC

---

Prof. Me. Marcos Dantas Ortiz

Universidade Federal do Ceará - UFC

Aos meus pais e irmãos, pessoas de fundamental importância durante a minha formação.

## **AGRADECIMENTOS**

Agradeço primeiramente a Deus, por ter me dado forças durante esses 3 anos para superar os obstáculos e as dificuldades enfrentadas.

Aos meus pais Antônio de Pádua Lira Maciel e Maria Algediva Pereira Maciel, e aos meus irmãos Diego Pereira Maciel e David Pereira Maciel, por serem peças fundamentais durante a minha caminhada e por sempre terem me apoiado para fazer a faculdade longe de casa.

Aos meus tios José de Fátima e Maria Eridan Pereira, por terem me acolhido como um filho e por terem me dado um lar durante esses 3 anos.

Ao meu orientador, professor Arthur de Castro Callado, e ao meu coorientador, professor Roberto Cabral Rabêlo Filho, pela oportunidade, paciência, apoio, dicas e dedicação durante este trabalho.

Ao professor Marcos Dantas Ortiz, por ter se disponibilizado e tirado um pouco do seu tempo para participar da banca examinadora.

A todos os amigos que fiz na faculdade, em especial Iasmyn Magalhães, Bruno de Melo, Marcos Paulo e Jefferson Costa, por fazerem parte da minha vida acadêmica e por continuarem presentes até hoje.

## RESUMO

Este trabalho tem como objetivo detectar e analisar possíveis vulnerabilidades existentes em um servidor OJS local utilizando técnicas de teste de penetração. Para isso, foram utilizadas técnicas de teste de penetração automatizado, através do uso de *scanners* web que realizam varreduras em um sistema em busca de vulnerabilidades web, e também utilizando técnicas de teste de penetração manual com o auxílio da ferramenta Burp Suite, com o objetivo de encontrar o máximo de vulnerabilidades possíveis. Os *scanners* utilizados neste trabalho foram o Acunetix e o OWASP ZAP. A ferramenta Acunetix encontrou um número maior de vulnerabilidades, e as vulnerabilidades encontradas pela ferramenta OWASP ZAP que poderia prejudicar o OJS foram as mesmas encontradas pelo Acunetix. A versão do OJS utilizada neste trabalho foi a 3.1.1-4, por ser a versão mais atual e estável do sistema. Esta versão mostrou-se como um sistema web bastante seguro contra as atuais vulnerabilidades que mais afetam sistemas web atualmente, o que pode ser explicado pelo fato do OJS ser um sistema de código aberto e existirem fóruns de discussões feitos pelo PKP, onde administradores de revistas podem dar sugestões sobre melhorar a segurança do sistema.

**Palavras-chave:** Segurança computacional. Teste de intrusão. Código aberto.

## **ABSTRACT**

This work aims to detect and analyze potential vulnerabilities in a local OJS server using penetration testing techniques. For this, automated penetration testing techniques were used, through the use of web scanners that scan a system for web vulnerabilities, and also using manual penetration testing techniques with the aid of the Burp Suite tool, with the purpose of finding as many vulnerabilities as possible. The scanners used in this work were Acunetix and OWASP ZAP. The Acunetix tool encountered a larger number of vulnerabilities, and the vulnerabilities found by the OWASP ZAP tool that could harm OJS were the same as those found by Acunetix. The version of OJS used in this work was 3.1.1-4, as it is the most current and stable version of the system. This release proved to be a very secure web system against the current vulnerabilities that most affect web systems today, which can be explained by the fact that OJS is an open source system and there are discussion forums made by PKP, where magazine administrators can give suggestions on improving system security.

**Keywords:** Computacional security. Intrusion test. Open source.



## LISTA DE ILUSTRAÇÕES

Figura 1 - Topologia de rede deste trabalho. ....	27
Figura 2 - Tela de <i>login</i> do Acunetix. ....	28
Figura 3 - Resultado do <i>scan</i> com a ferramenta Acunetix. ....	30
Figura 4 - Mensagem de erro do servidor Apache. ....	31
Figura 5 - URL <a href="http://200.129.39.72/ojs/lib/pkp">http://200.129.39.72/ojs/lib/pkp</a> sendo acessada. ....	32
Figura 6 - Utilização do filtro <code>http.request.method == POST</code> no Wireshark. ....	33
Figura 7 - Credenciais de <i>login</i> e senha vistas em texto claro. ....	33
Figura 8 - Caixa <i>Remember Me</i> . ....	34
Figura 9 - Redefinindo senha do usuário. ....	35
Figura 10 - Tentativa de <i>SQL Injection</i> . ....	37
Figura 11 - Filtragem utilizada em ataques <i>SQL Injection</i> . ....	38
Figura 12 - Exemplo de XSS. ....	39
Figura 13 - Ataque XSS realizado na página de busca. ....	39
Figura 14 - Filtragem utilizada em ataques XSS. ....	40
Figura 15 - Emails de usuários do OJS. ....	40
Figura 16 - Arquivo PHP submetido no sistema. ....	42
Figura 17 - Geração do <i>Cookie</i> OJSSID. ....	43
Figura 18 - Campo oculto <code>csrfToken</code> no formulário de <i>login</i> . ....	44
Figura 19 - Configuração do <i>proxy</i> no navegador. ....	45
Figura 20 - Informações do administrador vistas na ferramenta Burp Suite. ....	46
Figura 21 - Informações do usuário comum vistas na ferramenta Burp Suite. ....	46
Figura 22 - Busca por <a href="http://localhost/ojs">http://localhost/ojs</a> no navegador. ....	53
Figura 23 - Criação da conta de administrador do OJS. ....	53
Figura 24 - Escolha das línguas usadas pelo OJS. ....	54
Figura 25 - Configurando o banco de dados pelo navegador. ....	55
Figura 26 - Tela de confirmação da instalação do OJS. ....	55

## LISTA DE TABELAS

Tabela 1 - Vulnerabilidades encontradas em diferentes versões do OJS.....	17
Tabela 2 - Comparação entre <i>pentest</i> manual e ferramentas para testes automatizados. ....	19
Tabela 3 – Relação dos trabalhos citados com este trabalho. ....	19
Tabela 4 – Vulnerabilidades exploradas. ....	49

## LISTA DE ABREVIATURAS E SIGLAS

OJS	<i>Open Journal Systems</i>
SEER	Sistema Eletrônico de Editoração de Revistas
PKP	<i>Public Knowledge Project</i>
<i>Pentest</i>	<i>Penetration Test</i>
PTES	<i>Penetration Test Execution Standard</i>
OSSTMM	<i>Open Source Security Testing Methodology Manual</i>
ISECOM	<i>Institute for Security and Open Methodologies</i>
TI	Tecnologia da Informação
TIC	Tecnologia da Informação e Comunicação
IP	<i>Internet Protocol</i>
GB	<i>Gigabyte</i>
LTS	<i>Long Term Support</i>
XML	<i>Extensible Markup Language</i>
XXE	<i>XML eXternal Entity</i>
OWASP	<i>Open Web Application Security Project</i>
ZAP	<i>Zed Attack Proxy</i>
SQL	<i>Structured Query Language</i>
XSS	<i>Cross-Site Scripting</i>
API	<i>Application Programming Interface</i>
URL	<i>Uniform Resource Locator</i>
ISSN	<i>International Standard Serial Number</i>
HTTP	<i>Hypertext Transfer Protocol</i>
HTTPS	<i>Hyper Text Transfer Protocol Secure</i>
HTML	<i>Hypertext Markup Language</i>
PDF	<i>Portable Document Format</i>
PHP	<i>PHP: Hypertext Preprocessor</i>
CSRF	<i>Cross-site Request Forgery</i>
Captcha	<i>Completely Automated Public Turing test to tell Computers and Humans</i>
<i>Apart</i>	

## SUMÁRIO

1. INTRODUÇÃO.....	13
1.1 Objetivos.....	14
1.1.1 <i>Objetivo Geral</i> .....	14
1.1.2 <i>Objetivos Específicos</i> .....	15
2. TRABALHOS RELACIONADOS.....	16
3. REFERENCIAL TEÓRICO.....	20
3.1 Segurança da Informação.....	20
3.1.1 <i>Análise de Vulnerabilidades</i> .....	21
3.2 OSSTMM.....	21
3.3 Teste de Penetração de Caixa Branca.....	22
3.4 Teste de Penetração de Caixa Preta.....	23
3.5 Ameaças comumente encontradas em aplicações web.....	23
4. PROCEDIMENTOS METODOLÓGICOS.....	26
4.1 Instalar e configurar o servidor OJS em uma máquina virtual.....	26
4.2 Instalar o sistema Kali Linux em uma máquina virtual.....	26
4.3 Topologia de rede deste trabalho.....	27
4.4 Pesquisar por vulnerabilidades existentes da versão 3.1.1-4.....	28
4.5 Escanear o OJS com a ferramenta Acunetix.....	28
4.6 Analisar as vulnerabilidades encontradas pelo Acunetix.....	29
4.7 Escanear o OJS com a ferramenta OWASP ZAP.....	29
4.8 Analisar as vulnerabilidades encontradas pelo OWASP ZAP.....	29
4.9 Realizar testes manuais.....	29
5. RESULTADOS.....	30
5.1 <i>Scan</i> com a ferramenta Acunetix.....	30
5.1.1 <i>Mensagem de erro da aplicação</i> .....	30
5.1.2 <i>Listagem de diretórios</i> .....	31
5.1.3 <i>Credenciais do usuário enviadas em texto claro</i> .....	32
5.1.4 <i>Senha com preenchimento automático ativado</i> .....	34
5.1.5 <i>Tentativas de adivinhação de senha na página de login</i> .....	36
5.2 Realização dos testes manuais.....	36
5.2.1 <i>SQL Injection</i> .....	36

5.2.2 XSS .....	38
5.2.3 <i>Emails de usuários encontrados no código fonte da página inicial</i> .....	40
5.2.4 <i>Upload de arquivos maliciosos</i> .....	41
5.2.5 <i>Tentativa de roubo de sessões através de cookies</i> .....	43
6. CONCLUSÃO .....	48
REFERÊNCIAS .....	50
APÊNDICE A - INSTALAÇÃO DO OJS .....	52
APÊNDICE B - CRIAÇÃO DE UMA REVISTA NO OJS .....	56
APÊNDICE C - CÓDIGO PHP PARA EXECUTAR COMANDOS VIA URL.....	57
ANEXO A - FUNÇÃO GERADORA DO CSRFTOKEN.....	58

## 1. INTRODUÇÃO

Com a ascensão da Internet como meio de compartilhar informações, não demorou muito para surgirem aplicações web responsáveis por fornecer serviços que necessitam de dados pessoais de usuários que usam seus serviços. Tetsky, Kharchenko e Uzun (2018) afirmam que “O desenvolvimento de tecnologias possibilitou a oferta de vários serviços online; rápido desenvolvimento foi dado às pequenas e médias empresas na Internet”.

Juntamente com essa ascensão, cresceu também o número de cibercriminosos que realizam ataques a diversas empresas com o objetivo de quebrar a confidencialidade e a integridade de seus dados, tentando prejudicar a empresa de alguma forma. A segurança de dados e informações é uma das maiores prioridades das empresas atualmente. Todas as empresas precisam proteger suas informações para criar uma vantagem competitiva (AL SHEBLI; BEHESHTI, 2018).

Com a contínua evolução da segurança da informação e das novas tecnologias que surgem, é praticamente impossível imaginar um sistema que esteja 100% seguro e livre de ataques, pois assim como um novo sistema surge, podem ser descobertas também novas vulnerabilidades que possivelmente serão exploradas por atacantes, cabendo aos responsáveis pela segurança tratar aquela nova vulnerabilidade o mais rápido possível.

O teste de penetração, também conhecido como *Pentest*, é uma técnica muito conhecida que tem como objetivo coletar informações de um sistema ou rede, utilizar ferramentas para detectar vulnerabilidades de segurança baseado nas informações coletadas e elaborar um relatório com todas as vulnerabilidades encontradas e recomendações de como corrigi-las. É importante ressaltar que o teste de penetração é feito por um hacker ético ou uma equipe com as devidas permissões dadas pela empresa ou organização. Bertoglio e Zorzo (2016) ressaltam que “*Pentest* é a tentativa controlada de penetrar um sistema ou rede a fim de detectar vulnerabilidades, empregando as mesmas técnicas que são utilizadas em um ataque propriamente dito”.

Existem três tipos de testes de penetração:

- Teste de caixa preta: o profissional não recebe nenhuma informação sobre o sistema ou rede que tentará invadir, ou seja, o próprio testador deve coletar informações sobre o alvo antes do ataque. Esse teste é muito importante, pois esse tipo de ataque é o que mais se assemelha aos ataques feitos por hackers com más intenções que querem, de alguma maneira, prejudicar a empresa dona do sistema invadido;

- Teste de caixa branca: a empresa ou organização que contratou o profissional fornece-lhe todo um conjunto de informações sobre o sistema e/ou rede, tais como código fonte de sistemas, detalhes do sistema operacional, portas utilizadas, quais serviços estão utilizando determinada porta e faixa de endereços IP (*Internet Protocol*). O teste de caixa branca é normalmente utilizado quando o alvo é um sistema web, pois o testador teria acesso ao código fonte da aplicação para facilitar a detecção de falhas;
- Teste de caixa cinza: neste teste, são fornecidas ao testador poucas informações sobre o sistema que será invadido, cabendo a ele coletar mais informações baseadas no que lhe foi dado e realizar a invasão. Este tipo de ataque é normalmente associado a um hacker que está dentro da empresa e, de alguma forma, obteve acesso não autorizado a documentos ou áreas do sistema que lhes são restritas.

O OJS (*Open Journal Systems*) é um sistema web para gerenciamento e publicação de revistas e periódicos. Foi lançado pela primeira vez em 2002 como software livre e distribuído gratuitamente pelo PKP (*Public Knowledge Project*). O OJS foi projetado para gerenciar o fluxo de trabalho da revista, desde a submissão do manuscrito até a revisão, o trabalho editorial e a publicação, oferecendo meios prontos para publicar uma edição online e gerenciar melhor os custos operacionais da revista (EDGAR; WILLINSKY, 2010). O OJS busca otimizar o sistema de publicação científica, reduzindo tempo, energia e dinheiro que seriam gastos em tarefas de secretaria e edição. Viabiliza o corte com despesas de impressão, oferecendo acesso online e gratuito aos leitores. No Brasil, o OJS é conhecido como SEER (Sistema Eletrônico de Editoração de Revistas).

Este trabalho tem como objetivo detectar e analisar possíveis vulnerabilidades existentes em um servidor OJS configurado localmente, utilizando técnicas de teste de penetração e ferramentas de *pentest* que auxiliem na detecção das possíveis vulnerabilidades existentes.

## 1.1 Objetivos

### 1.1.1 Objetivo Geral

Detectar e analisar possíveis vulnerabilidades existentes em um servidor OJS local utilizando técnicas de teste de penetração.

### **1.1.2 *Objetivos Específicos***

- Identificar falhas comumente encontradas em aplicações web em um servidor OJS;
- Avaliar como as vulnerabilidades encontradas poderiam prejudicar a segurança do OJS.

O restante deste trabalho está organizado conforme a seguir. No Capítulo 2 são apresentados os trabalhos relacionados. O Capítulo 3 contém o referencial teórico. O Capítulo 4 apresenta os procedimentos metodológicos. O Capítulo 5 contém os resultados. O Capítulo 6 apresenta a conclusão e em seguida são apresentadas as referências.



## 2. TRABALHOS RELACIONADOS

No trabalho de Parmar e Feleol (2013), é apresentado um estudo de caso de teste de penetração em uma empresa de grande porte com o objetivo de buscar vulnerabilidades existentes nos servidores da empresa e encontrar soluções de melhoria de segurança em seu ambiente de rede externo.

A metodologia PTES (*Penetration Test Execution Standard*) foi utilizada para a realização do *pentest* nessa empresa. Bertoglio e Zorzo (2015) ressaltam que “a metodologia PTES detalha instruções de como executar as tarefas que são requeridas para testar precisamente o estado da segurança em um ambiente“. A metodologia citada acima é composta de sete etapas:

- Interações pré-engajamento - nessa etapa, é feita uma discussão com o cliente sobre a definição do escopo do *pentest*. É nessa etapa que é definido tudo que o testador pode e o que não pode fazer durante os testes;
- Coleta de Informações - nessa etapa, é realizada uma coleta detalhada de informações sobre o alvo. Essas informações são usadas para determinar os tipos de ameaças existentes no alvo;
- Modelagem de Ameaças - nessa etapa, o testador analisa as informações coletadas para identificar quais as prováveis vulnerabilidades do alvo;
- Análise de Vulnerabilidades - nessa etapa, são realizados testes utilizando ferramentas automatizadas em busca de informações sobre as vulnerabilidades e as melhores formas de explorá-las;
- Exploração - a etapa de exploração é o momento que o testador obtém acesso a um sistema ou recursos, burlando controles de segurança que foram avaliados previamente;
- Pós-exploração - nessa etapa, o valor de cada máquina invada é determinado. O valor da máquina é determinado pela sensibilidade dos dados armazenados nela e pela utilidade da máquina em comprometer ainda mais a rede;
- Relatórios - nessa etapa, ocorre a apresentação dos resultados dos testes, incluindo as conclusões dos resultados alcançados e sugestões de direcionamento para eliminar ou reduzir os riscos de exploração das vulnerabilidades.

As fases do teste de penetração usando essa metodologia foram: Conversa com o gerente de TI (Tecnologia da Informação) da empresa para determinar as expectativas, os

objetivos e a definição do escopo; Coleta de informações (*footprint*); Rastreamento de portas utilizando a ferramenta Nmap (*Network Mapper*) para detectar quais portas estão abertas nos servidores e quais serviços estão rodando nelas. Após a análise das vulnerabilidades nos servidores, as falhas encontradas foram divididas em: falhas críticas (nenhuma encontrada), falhas de nível alto (uma encontrada), falhas de nível médio (dezesesseis encontradas) e falhas de nível baixo (duas encontradas).

A relação do trabalho acima com este se dá pelo fato de que ambos estão utilizando teste de penetração para encontrar vulnerabilidades e também pelo fato de que o trabalho citado utiliza uma metodologia específica para a realização dos testes. Porém, enquanto o trabalho citado utiliza a metodologia PTES para fazer um *pentest* em uma empresa, este trabalho usará a metodologia OSSTMM (*Open Source Security Testing Methodology Manual*) para detectar vulnerabilidades em um servidor OJS. Os detalhes da metodologia OSSTMM encontram-se na seção 3.2 deste trabalho.

No trabalho de Yunanri, Riadi e Yudhana (2018), os autores realizaram um teste de penetração em diferentes versões do sistema OJS utilizando o *scanner* OWASP (*Open Web Application Security Project*) ZAP (*Zed Attack Proxy*). O *scanner* utilizado realiza uma varredura no site selecionado para detectar possíveis vulnerabilidades web existentes no mesmo.

No trabalho citado, os autores não utilizaram uma metodologia específica para a realização do *pentest*. A ferramenta OWASP ZAP foi utilizada em três versões diferentes do OJS para detectar vulnerabilidades. As versões do OJS utilizadas nos testes foram a 3.0.0, 3.0.1 e 3.1.0. A Tabela 1 mostra o número de vulnerabilidades encontradas em cada versão e seus respectivos riscos.

Tabela 1 - Vulnerabilidades encontradas em diferentes versões do OJS.

Versão do OJS	Número de vulnerabilidades de risco alto	Número de vulnerabilidades de risco médio	Número de vulnerabilidades de risco baixo
3.0.0	1	5	5
3.0.1	1	5	6
3.1.0	0	4	4

Fonte: Yunanri, Riadi e Yudhana (2018).

Para determinar o risco das vulnerabilidades, os autores do trabalho citado utilizaram a determinação da ferramenta OWASP ZAP, pois ela classifica o risco de cada vulnerabilidade encontrada em três níveis (alto, médio e baixo) após o término do *scan*. Pôde-se notar que a versão mais atual do OJS utilizada no trabalho citado foi aquela com o menor número de vulnerabilidades encontradas pela ferramenta. Esse é um comportamento esperado, pois é normal que os desenvolvedores façam correções em problemas encontrados em versões antigas antes de lançar uma versão mais atual.

A relação do trabalho acima com este se dá pelo fato de que ambos utilizam a ferramenta OWASP ZAP para detectar vulnerabilidades em um sistema OJS, no entanto, enquanto o trabalho citado realiza testes em três versões diferentes do OJS, este trabalho realizará testes apenas na versão 3.1.1-4 (versão mais atual do sistema) do OJS, procurando vulnerabilidades comumente encontradas em aplicações web. A versão do OJS utilizada neste trabalho é a 3.1.1-4, versão mais atual e estável do sistema.

No trabalho de Nagpure e Kurkure (2017), os autores realizaram uma comparação entre o teste de penetração automatizado e o teste de penetração manual. Testes de penetração automatizados consistem em utilizar *scanners* que realizam uma varredura para detectar vulnerabilidades existentes. No entanto, o teste de penetração automatizado não garante que o máximo de vulnerabilidades sejam encontradas. Para complementar o teste automatizado, o *pentest* manual é utilizado para detectar vulnerabilidades que não foram detectadas nos testes anteriores.

No trabalho citado, os autores realizam testes automáticos e manuais em duas aplicações web: uma aplicação de comércio eletrônico e uma aplicação *Cloud*. As ferramentas utilizadas no teste automatizado foram Burp Suite (a funcionalidade *Scanner* do Burp Suite foi utilizada no trabalho citado, e esta funcionalidade só está presente na versão paga da ferramenta), Acunetix e OWASP ZAP. A Tabela 2 mostra a comparação feita entre as vulnerabilidades encontradas no teste manual e as vulnerabilidades encontradas pelas ferramentas utilizadas nos testes automatizados.

A relação do trabalho citado com este se dá pelo fato de que em ambos os trabalhos, as ferramentas Acunetix, OWASP ZAP e Burp Suite foram utilizadas. No entanto, neste trabalho foi utilizada a versão grátis da ferramenta Burp Suite. Além disso, a ferramenta Burp Suite foi utilizada neste trabalho para auxiliar nos testes manuais, diferente do trabalho citado, que utilizou a funcionalidade *scanner* do Burp Suite para realizar testes automáticos.

A Tabela 3 mostra de forma resumida como os trabalhos citados relacionam-se com este trabalho.

Tabela 2 - Comparação entre *pentest* manual e ferramentas para testes automatizados.

Vulnerabilidade encontrada	Teste manual	Burp Suite	Acunetix	OWASP ZAP
<i>Cross-site Scripting</i>	X	-	X	X
Injeção de SQL	X	-	X	X
<i>Clickjacking</i>	X	X	X	X
<i>Upload de arquivo</i>	X		X	-
Fraqueza no cache do servidor	X	X	X	-
<i>Directory Traversal</i>	X	X	X	X
<i>Bypass Authentication</i>	X	-	-	-
<i>Cross-site Request Forgery</i>	X	-	-	-

Fonte: Nagpure e Kurkure (2017).

Tabela 3 – Relação dos trabalhos citados com este trabalho.

Trabalho	Utiliza uma metodologia específica	Utiliza ferramentas para testes automatizados	Realiza testes de penetração no OJS
Parmar e Feleol (2013)	X	-	-
Yunanri, Riadi e Yudhana (2018)	-	X	X
Nagpure e Kurkure (2017)	-	X	-
Este trabalho	X	X	X

Fonte: Elaborada pelo autor.

### 3. REFERENCIAL TEÓRICO

Neste capítulo estão listados os principais conceitos relacionados a este trabalho.

#### 3.1 Segurança da Informação

A Segurança da Informação é um paradigma da TIC (Tecnologia da Informação e Comunicação) que surgiu com o intuito de mudar o modo como um usuário pode ter acesso a determinados dados de algum serviço e também dificultar a atacantes o acesso a dados restritos de uma organização. Ferreira (2003) define que “a segurança da informação protege a informação de diversos tipos de ameaças garantindo a continuidade dos negócios, minimizando os danos e maximizando o retorno dos investimentos e das oportunidades”.

Uma boa prática para alcançar esses objetivos é estabelecer políticas de segurança, tais como autenticação de usuários, configuração das devidas permissões que cada usuário possui dentro de um sistema ou rede, utilização de criptografia para esconder o real conteúdo de dados que trafegam por um servidor e configuração de um *firewall* para, por exemplo, liberar/proibir o tráfego de informações por determinadas portas em um servidor.

Um incidente de segurança é caracterizado quando uma das seguintes áreas é afetada:

- Confidencialidade - um recurso deve estar acessível apenas para a pessoa ou grupo que foi definido como usuário autorizado para dispor daquele acesso, e nenhum outro;
- Integridade - garantia da consistência da informação ao longo do seu ciclo de vida;
- Disponibilidade - garantir que a informação esteja sempre disponível para os usuários legítimos sempre que eles queiram acessá-la;
- Autenticidade - garantia de que uma informação ou documento foi elaborado/distribuído pelo autor a quem se atribui aquele documento ou informação;
- Não repúdio - o emissor de uma informação não pode negar que aquela informação foi enviada por ele;
- Privacidade - expressa a habilidade de um indivíduo em controlar a exposição e a disponibilidade de informações acerca de si.

No contexto deste trabalho, a segurança da informação relaciona-se ao fato de que um teste de penetração é uma maneira de encontrar possíveis vulnerabilidades em um sistema

que possam ser exploradas por um atacante, permitindo que este viole uma das áreas descritas acima.

### **3.1.1 Análise de Vulnerabilidades**

Uma vulnerabilidade é uma falha em um software ou uma configuração de sistema que pode ser explorada. Analisar as vulnerabilidades encontradas é uma fase importantíssima durante um teste de penetração, pois essa análise servirá de base para a realização da última etapa de um teste de penetração, que é a geração do relatório que detalha cada vulnerabilidade encontrada e sugere uma correção. Segundo Boyle e Panko (2012), a análise de vulnerabilidades tenta encontrar falhas rodando um programa que busca por vulnerabilidades em servidores que estão no escopo do *pentest*. Esses programas realizam uma bateria de ataques contra esses servidores e geram relatórios detalhados sobre as vulnerabilidades encontradas.

No contexto deste trabalho, a Análise de Vulnerabilidades relaciona-se pelo fato de que técnicas de teste de penetração serão utilizadas para encontrar vulnerabilidades em um servidor OJS, e posteriormente essas vulnerabilidades serão analisadas para avaliar como poderiam prejudicar o OJS.

## **3.2 OSSTMM**

Segundo Valdez (2013), o objetivo da metodologia OSSTMM é fornecer uma metodologia científica para examinar uma organização, realizando testes de segurança e fornecendo diretrizes para o auditor de sistemas. Essa metodologia foi desenvolvida pela organização ISECOM (*Institute for Security and Open Methodologies*) e é composta de quatro etapas:

- Indução - nesta etapa, o período de tempo em que os testes serão realizados e o tipo de teste devem ser especificados previamente;
- Interação - nesta etapa, os objetivos do teste de penetração devem ser estabelecidos entre o testador ou a equipe de profissionais contratada e a empresa/organização contratante;
- Inquérito - nesta etapa, o máximo de dados possíveis sobre o alvo devem ser coletados antes da realização dos testes;
- Intervenção - nesta etapa, os testes são iniciados e, ao final dos testes, o desempenho de segurança do sistema alvo é medido.

As quatro etapas da metodologia OSSTMM possibilitam que um testador, ou a equipe de testadores, realize um teste de penetração completo e com objetivos bem definidos, pois a metodologia estabelece desde o tempo em que os testes devem ser realizados até a medição da segurança do sistema alvo após o término dos testes realizados.

Diferentemente da metodologia OSSTMM, a metodologia PTES, utilizada em Parmar e Feleol (2013), divide o teste de penetração em sete etapas, sendo a primeira etapa a discussão do escopo do teste com o cliente (Interações pré-engajamento), e a última etapa sendo a geração do relatório para o cliente. Porém, para este trabalho, não faria sentido passar pelas fases de Interações pré-engajamento e geração de relatórios, pois não existem clientes envolvidos para discussão de escopo e entrega de relatórios para os mesmos.

A metodologia OSSTMM foi escolhida para este trabalho por se tratar de uma metodologia que consegue se alinhar perfeitamente com os objetivos estabelecidos para este trabalho através das suas quatro etapas. Na fase de indução, foram determinados o tempo de execução dos testes, que ocorreram entre os meses de Setembro a Novembro de 2019, assim como o tipo de teste utilizado, que foi o teste de caixa branca. Na fase de interação, os objetivos (identificar falhas comumente encontradas em aplicações web em um servidor OJS e avaliar como as vulnerabilidades encontradas poderiam prejudicar a segurança do OJS) também foram estabelecidos. Para realizar a fase de inquérito, foi feita uma coleta de dados sobre possíveis vulnerabilidades existentes na versão 3.1.1-4 do OJS com o objetivo de auxiliar na busca por vulnerabilidades. Por fim, na fase de intervenção, os testes deste trabalho foram iniciados. Ao final dos testes, a avaliação da segurança do OJS também foi realizada baseando-se nos resultados obtidos após os testes.

### **3.3 Teste de Penetração de Caixa Branca**

O teste de penetração de caixa branca possibilita um teste mais completo, pois o profissional, ou a equipe de profissionais, recebem toda uma gama de informações acerca do sistema ou rede que irão testar, tais como topografia da rede, senhas, IPs, *logins* e todos os outros dados que dizem respeito à rede, servidores, estrutura, potenciais medidas de segurança e *firewalls*. O teste de penetração de caixa branca é muito utilizado quando o alvo do teste é um sistema web, pois dessa maneira o testador possuirá acesso ao código fonte da aplicação alvo, facilitando assim a detecção de possíveis vulnerabilidades.

Esse tipo de teste relaciona-se com este trabalho pelo fato de que o servidor OJS será instalado e configurado pelo autor deste trabalho, ou seja, informações como sistema operacional do servidor, IP do servidor, *logins* e senhas de usuários, banco de dados utilizado

pelo servidor e a versão do banco já serão previamente conhecidas. O teste de caixa branca é normalmente considerado como uma simulação de um ataque por uma fonte interna. Também é conhecido como estrutural, caixa de vidro, caixa transparente e teste de caixa aberta (JIMÉNEZ, 2016).

### 3.4 Teste de Penetração de Caixa Preta

O teste de penetração de caixa preta possibilita testes mais semelhantes a ataques reais feitos por atacantes mal intencionados, pois nesse tipo de teste o testador, ou a equipe de testadores, não recebe nenhuma informação sobre o alvo em questão, cabendo a ele ter que pesquisar sobre informações e reunir o máximo de dados possíveis sobre o alvo para, assim, conseguir explorar possíveis vulnerabilidades baseado nas informações coletadas. Jiménez (2016) ressalta que “no teste de penetração de caixa preta, o testador não tem ideia sobre os sistemas que ele vai testar. Ele está interessado em coletar informações sobre a rede ou sistema alvo. Ele não examina nenhum código de programação”.

O testador poderia não saber, por exemplo, da existência de um *firewall* que está configurado para ajudar na proteção do sistema alvo. A utilização de um *firewall* poderia dificultar a realização dos testes, pois caso o testador decida utilizar um *scanner* para realizar uma varredura no sistema, ele poderia ter o seu endereço IP bloqueado pelo *firewall* por conta da existência de uma regra que bloqueie um determinado endereço IP após esse endereço ultrapassar determinado número de requisições por segundo.

### 3.5 Ameaças comumente encontradas em aplicações web

Existem ameaças que são comumente encontradas em aplicações web. O OWASP é uma comunidade *online* que disponibiliza gratuitamente artigos, metodologias, documentação e ferramentas com foco em segurança da informação em aplicações web. O documento mais conhecido do OWASP é o OWASP *Top Ten*, cujo objetivo é fornecer conscientização para a segurança de aplicações web. Esse documento traz uma lista das dez vulnerabilidades mais comuns encontradas em aplicações web. Os membros do projeto incluem uma variedade de especialistas em segurança de todo o mundo que compartilharam seus conhecimentos para produzir essa lista. O último documento foi publicado no ano de 2017 e seus itens são:

- **Injeção** - falhas de injeção, por exemplo, SQL (*Structured Query Language*) *injection* ocorrem quando dados não confiáveis são enviados para um interpretador como parte de um comando ou consulta ilegítima. Os dados



enviados pelo atacante podem enganar o interpretador, levando-o a executar os comandos não pretendidos ou exibir dados confidenciais;

- **Quebra de Autenticação** - as funções da aplicação web que lidam com autenticação e gestão de sessões de usuários são, muitas vezes, implementadas incorretamente, possibilitando ao atacante comprometer senhas e *tokens* de sessão, permitindo-lhe assumir a identidade de usuários legítimos e realizar ações indevidas;
- **Exposição de Dados Sensíveis** - muitas aplicações web não protegem dados sensíveis de forma adequada. Os atacantes podem roubar ou modificar esses dados mal protegidos para realizar fraudes com cartões de crédito, roubo de identidade, ou outros crimes. Dados sensíveis necessitam de proteções extras, tais como encriptação quando forem armazenados e quando estiverem trafegando;
- **XXE (XML eXternal Entity)** - ataques de XXE ocorre, quando uma entrada XML (*Extensible Markup Language*) contendo uma referência a uma entidade externa é processada por um analisador XML configurado de forma insegura. O XML possui uma característica que permite que o desenvolvedor aponte para um endereço onde o dado está armazenado, local ou remotamente. Com isso, um atacante pode alterar essa informação e solicitar dados locais ou remotos.
- **Quebra de Controle de Acesso** - restrições sobre o que os usuários autenticados estão autorizados a fazer nem sempre são corretamente verificadas. Os atacantes podem abusar destas falhas para ter acesso a funcionalidades ou dados para os quais não têm autorização, tais como dados de outras contas de utilizador, visualizar dados sensíveis, modificar os dados de outros utilizadores e alterar as permissões de acesso;
- **Configurações de Segurança Incorretas** - as más configurações de segurança são o aspecto mais observado nos dados recolhidos. Normalmente isto é consequência de configurações padrão inseguras, tais como armazenamento na nuvem sem qualquer restrição de acesso, cabeçalhos HTTP (*Hypertext Transfer Protocol*) mal configurados ou mensagens de erro com informações sensíveis. Todos os sistemas operacionais, *frameworks*, bibliotecas de código e aplicações devem ser configurados de forma segura, como também devem sempre estar atualizados;

- **XSS (*Cross-Site Scripting*)** - As falhas de XSS ocorrem sempre que uma aplicação inclui dados não confiáveis numa nova página web sem a validação ou filtragem apropriadas, ou quando atualiza uma página web existente com dados enviados por um usuário através de uma API (*Application Programming Interface*) do *browser* que possa criar JavaScript. O XSS permite que atacantes possam executar scripts no *browser* da vítima, os quais podem raptar sessões do usuário, desfigurar páginas web ou redirecionar usuários para sites maliciosos.
- **Desserialização Insegura** - essa vulnerabilidade normalmente leva à execução remota de código. Pode ser usada para realizar ataques, incluindo ataques por repetição, injeção e elevação de privilégios.
- **Utilização de Componentes Vulneráveis** - alguns componentes, tais como, bibliotecas, *frameworks* e outros módulos de *software*, são executados com os mesmos privilégios que a aplicação. Caso a aplicação utilize algum componente com alguma vulnerabilidade conhecida, um atacante poderia tirar proveito disso;
- **Registro e Monitorização Insuficiente** - O registro e monitorização insuficientes, em conjunto com uma resposta a incidentes inexistente ou insuficiente permite que os atacantes possam abusar do sistema de forma persistente, que o possam usar como entrada para atacar outros sistemas, e que possam alterar, extrair ou destruir dados.

Ameaças comumente encontradas em aplicações web relacionam-se com este trabalho pelo fato do OJS ser um sistema web e também pelo fato de que o objetivo deste trabalho é encontrar possíveis vulnerabilidades existentes em um servidor OJS.

## 4. PROCEDIMENTOS METODOLÓGICOS

Para realizar a detecção e análise das vulnerabilidades no servidor OJS, os passos a seguir foram realizados ao longo da execução deste trabalho.

- Instalar e configurar o servidor OJS em uma máquina virtual;
- Instalar o sistema Kali Linux em uma máquina virtual;
- Topologia de rede deste trabalho;
- Pesquisar por vulnerabilidades existentes na versão 3.1.1-4;
- Escanear o OJS com a ferramenta Acunetix;
- Analisar as vulnerabilidades encontradas pelo Acunetix;
- Escanear o OJS com a ferramenta OWASP ZAP;
- Analisar as vulnerabilidades encontradas pelo OWASP ZAP;
- Realizar testes manuais.

### 4.1 Instalar e configurar o servidor OJS em uma máquina virtual

Antes da realização dos testes para detecção das vulnerabilidades, foi necessário preparar o ambiente para a instalação e configuração do servidor OJS. A versão do OJS utilizada foi a 3.1.1-4, por ser a mais atual e estável. O servidor foi instalado em uma máquina virtual com sistema operacional Ubuntu 18.04 64 bits LTS (*Long Term Support*), 4GB (*Gigabyte*) de memória RAM e 50 GB de armazenamento. Esta versão do Ubuntu foi escolhida por ser a versão mais atual e estável do sistema. O OJS foi escolhido para este trabalho por se tratar de um sistema muito usado mundialmente para publicações de periódicos e também pelo fato de quase não existem trabalhos relacionados que realizam testes de penetração em um servidor OJS. O *software* VirtualBox foi utilizado para a criação da máquina virtual.

Os detalhes da instalação do servidor OJS encontram-se no Apêndice A. Os detalhes para a criação de uma revista no OJS encontram-se no Apêndice B.

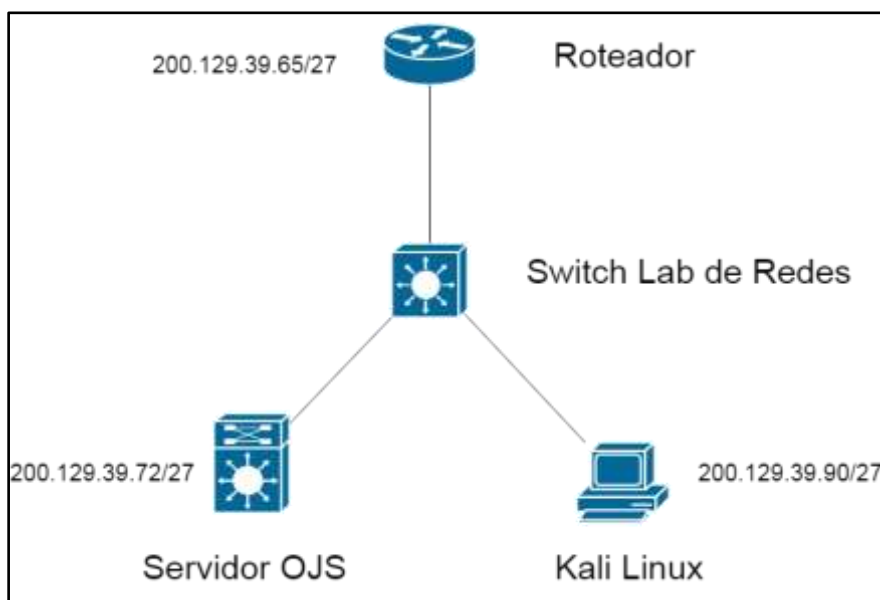
### 4.2 Instalar o sistema Kali Linux em uma máquina virtual

O sistema operacional Kali Linux, distribuição baseada no sistema operacional Debian, foi utilizado para a realização dos testes no servidor OJS por conta de oferecer várias ferramentas que automatizam o processo de escaneamento em aplicações web a procura de vulnerabilidades. Essas ferramentas facilitam bastante a vida de profissionais que estão realizando *pentests* em sistemas. O *software* VirtualBox foi utilizado para a criação da máquina virtual.

### 4.3 Topologia de rede deste trabalho

Todos os testes deste trabalho foram realizados no Laboratório de Redes da Universidade Federal do Ceará, Campus de Quixadá, em um ambiente controlado. A Figura 1 representa a topologia utilizada neste trabalho.

Figura 1 - Topologia de rede deste trabalho.



Fonte: Elaborada pelo autor.

Como pode ser visto na Figura 1, as duas máquinas estão interligadas através de um switch via cabo *Ethernet* categoria Cat5e para realizar a comunicação entre ambas. O switch, por sua vez, está conectado a um roteador também via cabo *Ethernet* categoria Cat5e. Uma informação importante que vale a pena ser ressaltada é que não existe nenhum tipo de *firewall* configurado no lado do servidor OJS para ajudar na segurança de seus dados.

Como o tipo de teste utilizado foi o teste de caixa branca, o uso de um *firewall* poderia dificultar a varredura feita pelos *scanners* utilizados. Durante a varredura, um grande número de requisições é feita ao servidor pelas ferramentas utilizadas, e um *firewall* poderia estar configurado com uma regra para bloquear um determinado endereço IP quando este ultrapasse um determinado número de requisições por segundo feitas ao servidor, impossibilitando assim o uso de *scanners*. Como todos os testes deste trabalho foram realizados em uma rede interna e em um ambiente controlado, não faria sentido configurar um *firewall* para dificultar a realização de varreduras dos *scanners* utilizados.

Em um teste de penetração de caixa preta, por exemplo, o testador poderia não saber da existência de um *firewall* com a regra descrita acima, o que impossibilitaria a realização de uma varredura no sistema com algum *scanner*.

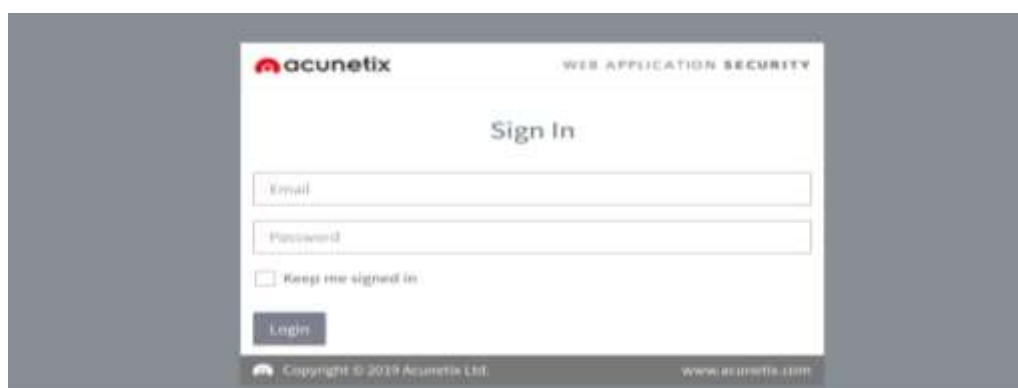
#### 4.4 Pesquisar por vulnerabilidades existentes da versão 3.1.1-4

Antes de iniciar os testes no servidor, foi feita uma pesquisa com o intuito de encontrar informações sobre vulnerabilidades existentes na versão 3.1.1-4 do OJS. Esta etapa do trabalho correspondeu à fase de Inquérito da metodologia OSSTMM, onde as informações sobre o alvo são coletadas. Foram encontradas informações sobre vulnerabilidades em versões mais antigas que já haviam sido resolvidas nas versões mais atuais. Especificamente sobre a versão 3.1.1-4, não foram encontradas informações relacionadas a vulnerabilidades existentes.

#### 4.5 Escanear o OJS com a ferramenta Acunetix

Após o término da pesquisa sobre possíveis vulnerabilidades existentes na versão 3.1.1-4, os testes utilizando a ferramenta Acunetix foram iniciados. Acunetix é um *scanner* de vulnerabilidades que faz uma varredura completa em aplicações web a procura de possíveis vulnerabilidades existentes. A ferramenta Acunetix é paga, porém é possível solicitar uma versão demonstrativa em seu site. Ao solicitar essa versão, um *script* será baixado e servirá para fazer a instalação no Kali Linux. Durante a instalação, informações de email e senha serão solicitadas para a criação da conta de administrador. Concluída a instalação, será possível acessar o Acunetix por um navegador através do endereço <https://kali:13443>, onde kali é o nome da máquina. A Figura 2 mostra a tela de *login* do Acunetix.

Figura 2 - Tela de *login* do Acunetix.



Fonte: Elaborada pelo autor.

Esse escaneamento foi feito com o intuito de automatizar ao máximo a realização dos testes e também para facilitar a busca por vulnerabilidades.

#### **4.6 Analisar as vulnerabilidades encontradas pelo Acunetix**

Após o término do *scan* com a ferramenta Acunetix, o próximo passo foi explorar as vulnerabilidades existentes encontradas pela ferramenta para avaliar como essas vulnerabilidades podem prejudicar o OJS. Um fator importante a se destacar sobre a ferramenta Acunetix é que, além de exibir as vulnerabilidades encontradas, também mostra o nível de criticidade de cada vulnerabilidade e possibilita que um relatório com as vulnerabilidades encontradas seja gerado. Por conta da versão utilizada do Acunetix ser a demonstrativa, informações como detalhes das vulnerabilidades e como resolvê-las não são disponibilizados.

#### **4.7 Escanear o OJS com a ferramenta OWASP ZAP**

Após o término da análise das vulnerabilidades encontradas pela ferramenta Acunetix, foi realizado um novo *scan* no OJS, dessa vez utilizando a ferramenta OWASP ZAP (OWASP *Zed Attack Proxy*). Assim como o Acunetix, o OWASP ZAP é um *scanner* de vulnerabilidades que faz uma varredura completa em aplicações web a procura de possíveis vulnerabilidades existentes. Esse *scan* foi realizado com o intuito de saber se novas vulnerabilidades não detectadas pelo Acunetix seriam detectadas pelo OWASP ZAP para serem exploradas.

#### **4.8 Analisar as vulnerabilidades encontradas pelo OWASP ZAP**

Após o término do *scan* com a ferramenta OWASP ZAP, percebeu-se que o número de vulnerabilidades detectadas por esta ferramenta foi menor quando comparado ao Acunetix, e todas as vulnerabilidades detectadas pelo OWASP ZAP que poderiam prejudicar o OJS também foram detectadas pelo Acunetix. Sabendo disso, considerou-se desnecessário analisar as vulnerabilidades detectadas pelo OWASP ZAP.

#### **4.9 Realizar testes manuais**

Após a análise das vulnerabilidades encontradas pelos *scanners* utilizados, o próximo passo foi realizar testes manuais sem o auxílio de *scanners* com o intuito de encontrar e analisar outras vulnerabilidades não encontradas pelos mesmos, tais como XSS, SQL *Injection*, *Upload* de arquivos maliciosos e CSRF (*Cross-site Request Forgery*).

## 5. RESULTADOS

### 5.1 Scan com a ferramenta Acunetix

Após a conclusão do escaneamento, o próximo passo foi explorar as vulnerabilidades pela ferramenta para saber como as vulnerabilidades encontradas poderiam prejudicar a segurança do OJS. A Figura 3 ilustra parte do resultado do *scan*.

Figura 3 - Resultado do *scan* com a ferramenta Acunetix.



Se...	Vulnerability	Count
	HTML form without CSRF protection	4
	User credentials are sent in clear text	0
	Clickjacking: X-Frame-Options header missing	1
	OPTIONS method is enabled	1
	Broken links	124
	Content type is not specified	355
	Error page web server version disclosure	1
	Password type input with auto-complete enabled	0

Fonte: Elaborada pelo autor.

#### 5.1.1 Mensagem de erro da aplicação

Esta vulnerabilidade encontrada pelo Acunetix relaciona-se ao fato do servidor Apache exibir mensagens de erro que podem conter informações sensíveis quando se é passada uma URL (*Uniform Resource Locator*) que não existe. A Figura 4 mostra um exemplo de exploração dessa vulnerabilidade ao utilizar, por exemplo, a URL `http://200.129.39.72/urlnaoexiste` em um navegador. O IP 200.129.39.72 é o endereço utilizado pela máquina do servidor OJS. Como pode ser visto na Figura 4, estão visíveis informações como o servidor web que está sendo utilizado, versão do sistema web, sistema operacional utilizado e porta que o servidor está utilizando.

Esta vulnerabilidade não se aplica diretamente ao OJS, mas é importante ressaltar que esse tipo de informação exposta pode ser útil para um atacante, pois o atacante poderia, por exemplo, detectar que uma versão antiga do servidor Apache está sendo utilizada. Sabendo disso, ele poderia pesquisar por vulnerabilidades existentes naquela versão específica e então realizar ataques à máquina que está hospedando o OJS.

Figura 4 - Mensagem de erro do servidor Apache.



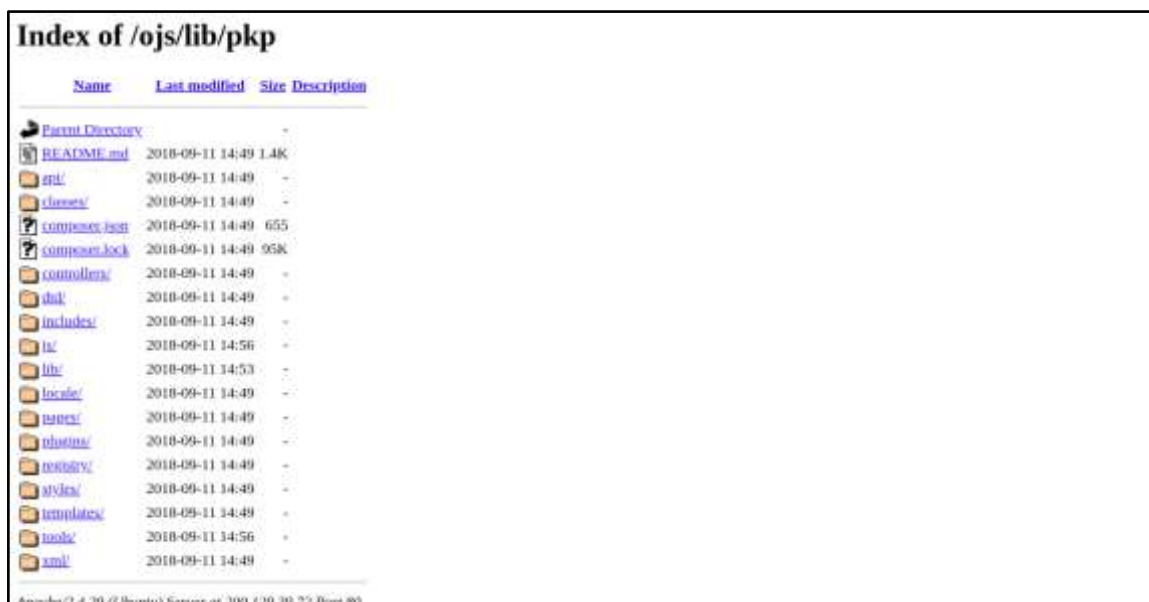
Fonte: Elaborada pelo autor.

### 5.1.2 Listagem de diretórios

Esta vulnerabilidade relaciona-se ao fato de ser possível que um usuário comum acesse diretórios e arquivos internos de um servidor web através do navegador. A Figura 5 mostra a URL <http://200.129.39.72/ojs/lib/pkp> sendo acessada através de um navegador. Não é uma boa prática deixar diretórios internos visíveis para usuários comuns poderem acessá-los, pois por um descuido do responsável pelo sistema, pode acontecer de arquivos sensíveis, como por exemplo arquivos de banco de dados ou arquivos de configuração da própria aplicação, ficarem expostos nesses diretórios e um usuário mal intencionado baixá-lo para sua máquina, examiná-lo e planejar algum tipo de ataque contra o sistema. Como é possível ver na Figura 5, ao acessar a URL <http://200.129.39.72/ojs/lib/pkp> do servidor OJS, foi possível ver vários subdiretórios e alguns arquivos internos do servidor, porém nenhum arquivo que comprometesse a confidencialidade ou a integridade dos dados do OJS foi encontrado, o que demonstra que os desenvolvedores do sistema preocuparam-se em manter arquivos importantes inacessíveis a usuários finais.



Figura 5 - URL <http://200.129.39.72/ojs/lib/pkp> sendo acessada.



Fonte: Elaborada pelo autor.

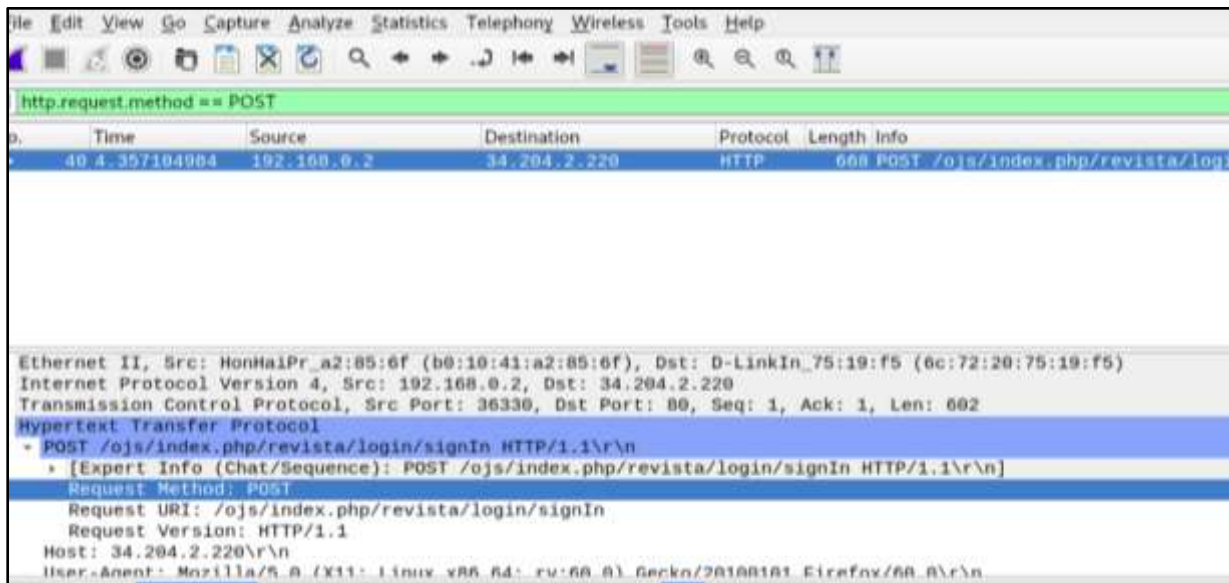
### 5.1.3 Credenciais do usuário enviadas em texto claro

Esta vulnerabilidade encontrada pelo *scanner* Acunetix relaciona-se mais ao servidor web Apache utilizado para hospedar o sistema OJS do que ao próprio OJS, mas considerou-se importante analisar esta vulnerabilidade encontrada por conta do seu possível impacto ao OJS.

Por padrão, o servidor Apache utiliza o protocolo HTTP para a transmissão de dados das aplicações hospedadas, porém este protocolo não utiliza nenhum tipo de criptografia durante a transmissão dos dados na rede. Informações cruciais para um usuário legítimo, como seu *login* e senha, passariam em texto claro durante o tráfego na rede. Apenas a utilização de uma ferramenta que capture o tráfego que passa por uma rede, como por exemplo, a ferramenta Wireshark, seria mais do que suficiente para que um atacante conseguisse ter acesso aos dados confidenciais de usuários que realizaram *login* no OJS através de uma captura de tráfego. A Figura 6 ilustra o uso da ferramenta Wireshark para capturar o pacote contendo a requisição de *login* através da expressão `http.request.method == POST` utilizada como filtro no Wireshark.

Ao clicar com o botão direito no pacote capturado e selecionar a opção *Follow*, e em seguida selecionar a opção *TCP Stream*, foi possível ver em texto claro as credenciais de *login* e senha do usuário que realizou *login* no momento da captura de pacotes. A Figura 7 ilustra os detalhes do pacote capturado.

Figura 6 - Utilização do filtro `http.request.method == POST` no Wireshark.



Fonte: Elaborada pelo autor.

Figura 7 - Credenciais de *login* e senha vistas em texto claro.



Fonte: Elaborada pelo autor.

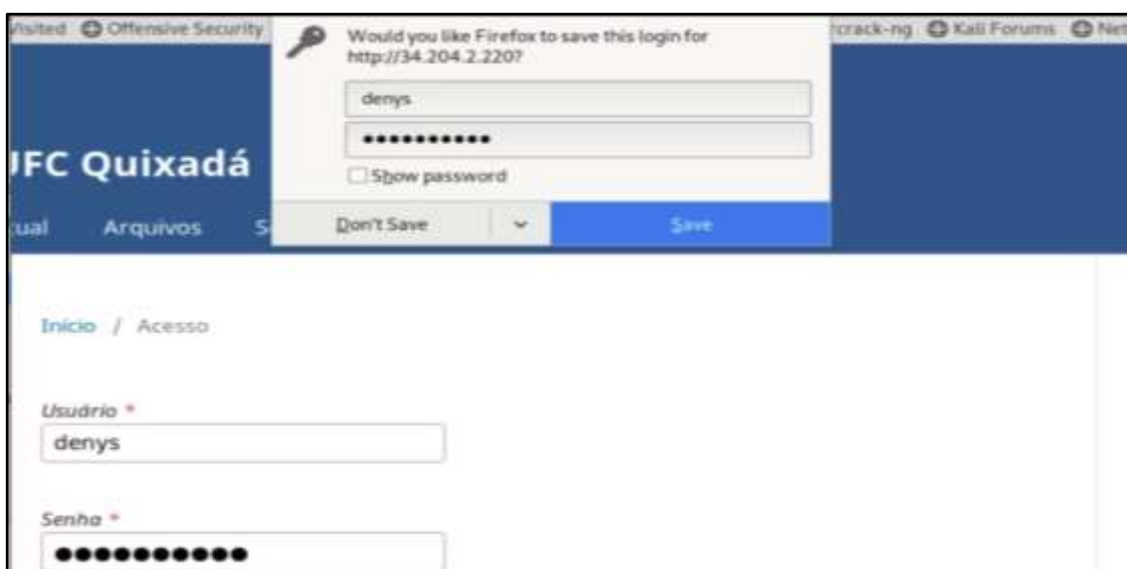
Esta é uma vulnerabilidade gravíssima encontrada pelo Acunetix por quebrar o conceito de confidencialidade de dados dos usuários legítimos do sistema, e também pode vir a quebrar o conceito de integridade dos dados, pois o atacante poderia também fazer alguma alteração nas submissões feitas por um usuário legítimo.

Esta vulnerabilidade pode facilmente ser resolvida através da ativação do protocolo HTTPS (*Hyper Text Transfer Protocol Secure*). Com a utilização do protocolo HTTPS, a troca de informações realizada entre o servidor e o cliente é feita utilizando criptografia nos dados durante a comunicação, ou seja, por mais que um atacante capture o tráfego da rede no momento em que um usuário realiza *login* para utilizar o sistema, suas credenciais não poderão mais ser vistas em texto claro como era possível no protocolo HTTP.

#### 5.1.4 Senha com preenchimento automático ativado

Alguns navegadores, como por exemplo Google Chrome e Mozilla Firefox, exibem uma pequena caixa *Remember Me* no topo da tela quando um usuário realiza *login* em alguma página, perguntando se o usuário deseja salvar suas credenciais utilizadas no momento do *login*, como pode ser visto na Figura 8.

Figura 8 - Caixa *Remember Me*.



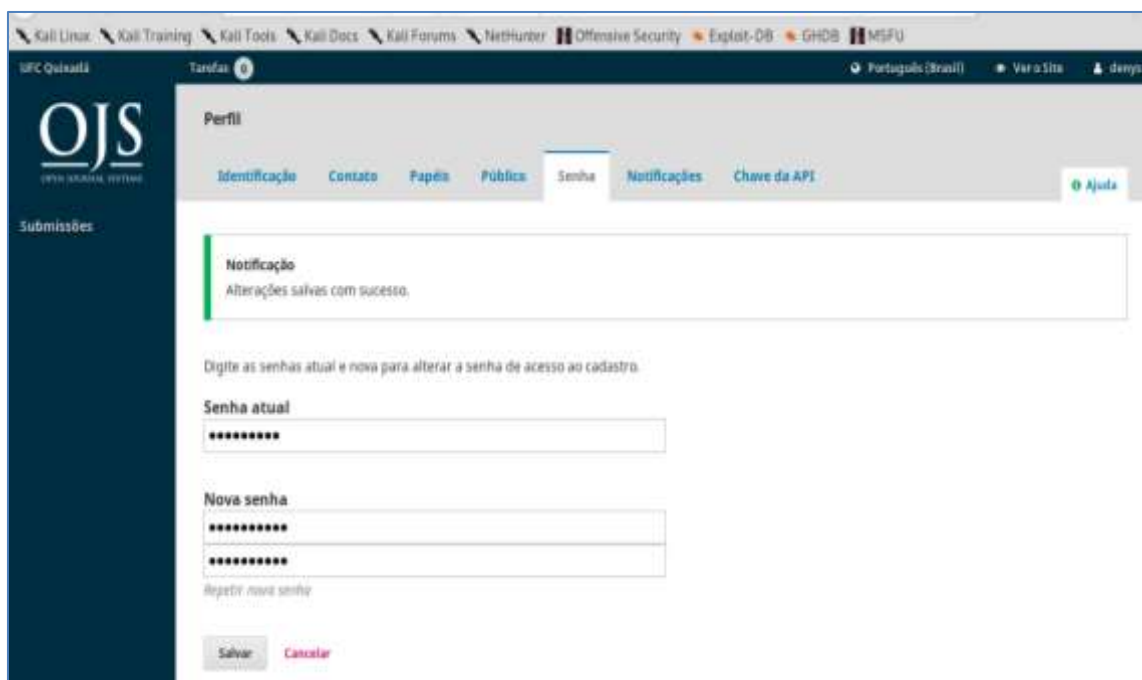
Fonte: Elaborada pelo autor.

Esta vulnerabilidade poderia ser explorada por um atacante que possui acesso local a máquinas utilizadas por usuários legítimos, pois caso um usuário deseje salvar sua senha através da opção *Remember Me*, um atacante com acesso local aquela máquina poderia

facilmente realizar *login* com as credenciais salvas e realizar ações não pretendidas por aquele usuário, como por exemplo a alteração de sua senha atual ou alterações em seus arquivos submetidos no OJS.

Um exemplo de exploração desta vulnerabilidade foi a alteração da senha do usuário. A senha foi salva através da opção *Remember Me*, e em seguida a página de perfil do usuário foi acessada. Na seção Senha, para redefinição de senha, os campos Senha atual, Nova senha e Repetir nova senha são requeridos para a alteração da senha. No entanto, por consequência da senha estar salva através da opção *Remember Me*, quando um click foi dado no campo Senha atual, o *login* do usuário logado apareceu logo abaixo. Ao clicar no *login*, o campo Senha atual foi preenchido automaticamente. Após preencher os campos Nova senha, Repetir nova senha e clicar no botão Salvar, foi possível redefinir a senha do usuário logado, como é possível ver na Figura 9.

Figura 9 - Redefinindo senha do usuário.

A screenshot of the OJS (Open Journal System) user profile page. The page is in Portuguese and shows the 'Perfil' (Profile) section. The 'Senha' (Password) tab is selected. A notification box at the top says 'Alterações salvas com sucesso.' (Changes saved successfully). Below the notification, there is a text prompt: 'Digite as senhas atual e nova para alterar a senha de acesso ao cadastro.' (Enter the current and new passwords to change the access password to the registration). There are three input fields: 'Senha atual' (Current password), 'Nova senha' (New password), and 'Repetir nova senha' (Repeat new password). The 'Senha atual' field is filled with asterisks. At the bottom, there are 'Salvar' (Save) and 'Cancelar' (Cancel) buttons. The browser's address bar shows the URL 'http://www.ojs.org.br/usuario/perfil/senha'.

Fonte: Elaborada pelo autor.

Esta vulnerabilidade também pode chegar a afetar os conceitos de confidencialidade e integridade dos dados de um usuário, e poderia ser resolvida adicionando ao código fonte do OJS a opção *autocomplete="off"* nos campos de *login* e senha do formulário da página de *login* do usuário.

### 5.1.5 Tentativas de adivinhação de senha na página de login

Esta vulnerabilidade relaciona-se ao fato de não existir um limite para o número de tentativas que um usuário pode tentar fazer *login* no OJS, ou seja, em um cenário de ataque real, um atacante poderia realizar ataques de força bruta na página de *login* à procura de senhas fracas existentes sem se preocupar com a quantidade de tentativas que ele teria. Como todos os *logins* de usuários e senhas utilizadas neste trabalho foram configurados pelo autor, não faria sentido realizar ataques de força bruta contra o sistema, por isso, optou-se por não realizá-los.

Uma possível solução para evitar ataques de força bruta seria através do uso de Captcha (*Completely Automated Public Turing test to tell Computers and Humans Apart*), que consiste em um campo que deve ser preenchido com uma palavra aleatória gerada pela própria ferramenta Captcha. Um usuário, além de fornecer suas credenciais de *login* e senha, também deve preencher este campo com a palavra aleatória para realizar *login* em um sistema.

## 5.2 Realização dos testes manuais

A seguir, serão descritos os testes manuais realizados no servidor OJS com o objetivo de detectar vulnerabilidades existentes não encontradas pelos *scanners*.

### 5.2.1 SQL Injection

Ataques de injeção de SQL ocorrem quando dados não confiáveis são enviados ao interpretador como parte de um comando ou consulta ao banco de dados. Esses dados são enviados ao interpretador com o intuito de enganá-los para executar operações no banco de dados. A consulta mais conhecida de *SQL Injection* é ' or 1=1, normalmente utilizada em campos de *login* e senha de um formulário ou até mesmo na própria URL do site, quando esta trabalha com parâmetros explicitamente na própria URL.

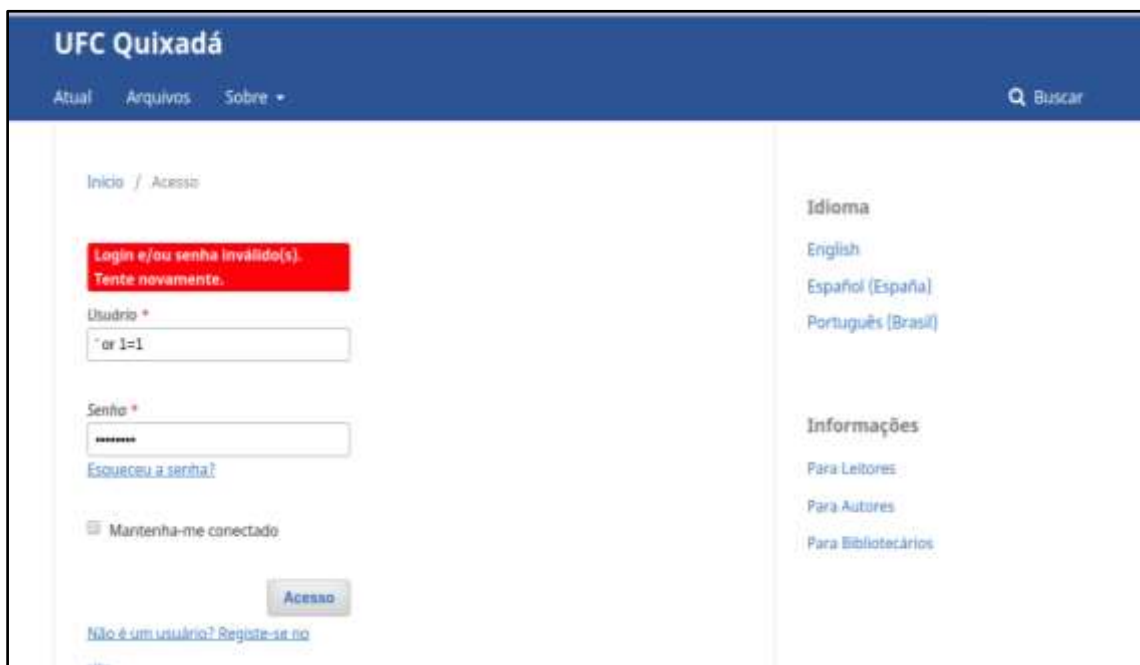
Optou-se por realizar ataques de *SQL Injection* por conta da possibilidade de realizar ataques ao banco de dados utilizado pelo OJS para roubar informações confidenciais, e também pelo fato de não se ter encontrado informações sobre essa vulnerabilidade no OJS. Para a realização dos testes de *SQL Injection* no OJS, a consulta ' or 1=1 foi utilizada nos campos de *login* e senha na página de *login* de usuário, porém não foi possível realizar o *login*, como pode ser visto na Figura 10. O OJS mostrou-se não vulnerável a *SQL Injection*, pois as entradas que os usuários colocam nos campos de *login* e senha são filtradas antes de serem enviadas ao interpretador. É possível verificar essa filtragem examinando o código

fonte da página de *login* do OJS após utilizar a entrada descrita acima. A Figura 11 ilustra a filtragem utilizada, onde é possível verificar que no campo de *login* do formulário utilizado, o caracter aspa simples (‘) foi substituído por &#039. Essa técnica de filtragem utilizada é conhecida como *Encoding*, que consiste em converter caracteres especiais para caracteres HTML (*Hypertext Markup Language*).

Ataques de *SQL Injection* também podem ocorrer via URL, quando a mesma utiliza campos do banco de dados diretamente para construir URL e também para exibir conteúdo de uma página. O site oficial da ferramenta Acunetix possui uma área para testes onde um usuário pode realizar ataques para testar habilidades. A URL <http://testphp.vulnweb.com/artists.php?artist=1>, que pertence a área de testes da ferramenta Acunetix, representa um exemplo de site suscetível a ataques de *SQL Injection* via URL, por conta de utilizar o campo *artist* diretamente na URL e este estar presente no banco de dados.

Para realizar esse tipo de teste, todas as funcionalidades possíveis do OJS foram acessadas em busca de URLs que poderiam estar suscetíveis a este tipo de ataque. Nenhuma URL suscetível foi encontrada, mostrando que, possivelmente, o OJS não é vulnerável a ataques de *SQL Injection*.

Figura 10 - Tentativa de *SQL Injection*.



Fonte: Elaborada pelo autor.

Figura 11 - Filtragem utilizada em ataques SQL Injection.

```

eldset class="fields">
<div class="username">
  <label>
    <span class="label">
      Usuário
      <span class="required">*</span>
      <span class="pkp_screen_reader">
        Obrigatório
      </span>
    </span>
    <input type="text" name="username" id="username" value="&#039; or 1=1" maxlength="32"
  </label>
</div>
<div class="password">
  <label>
    <span class="label">
      Senha
      <span class="required">*</span>
      <span class="pkp_screen_reader">
        Obrigatório
      </span>
    </span>
    <input type="password" name="password" id="password" value="" password="true" maxlengt

```

Fonte: Elaborada pelo autor.

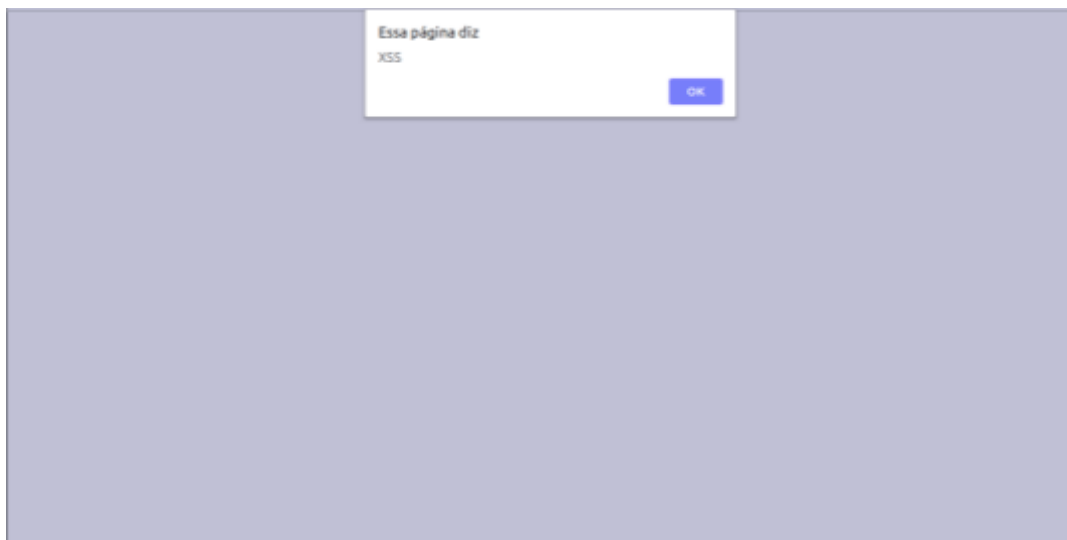
### 5.2.2 XSS

Ataques XSS permitem que atacantes possam executar comandos ou scripts no *browser* da vítima, os quais podem raptar sessões do usuário, desfigurar páginas web ou redirecionar usuários para sites maliciosos. O ataque XSS mais conhecido e utilizado para testes é o de inserção de um *alert* na linguagem JavaScript em uma campo de busca de um site através das *tags* `<script>alert('XSS')</script>`. Optou-se por realizar ataques de XSS por conta das consequências que um ataque bem sucedido teria. Ao inserir, por exemplo, as *tags* `<script>alert(document.cookie)</script>`, um atacante poderia ter acesso a *cookies* de usuários utilizados pelo sistema caso o mesmo seja vulnerável a esse tipo de ataque, o que possibilitaria um outro ataque de roubo de sessão, onde um atacante consegue realizar ações passando-se por outro.

Ao inserir essa *tag* em um campo de busca, caso o site seja vulnerável, um *alert* com a mensagem “XSS” irá aparecer na tela, como pode ser visto na Figura 12. Esse tipo específico de ataque XSS é chamado de *Reflected XSS*, pois uma entrada preenchida pelo usuário é imediatamente mostrada na tela sem antes passar por uma filtragem. Um atacante poderia, por exemplo, utilizar a *tag* `<script>alert(document.cookie)</script>` para conseguir acesso a *cookies* de usuários, roubar sua sessão e realizar ações passando-se por aquele usuário.



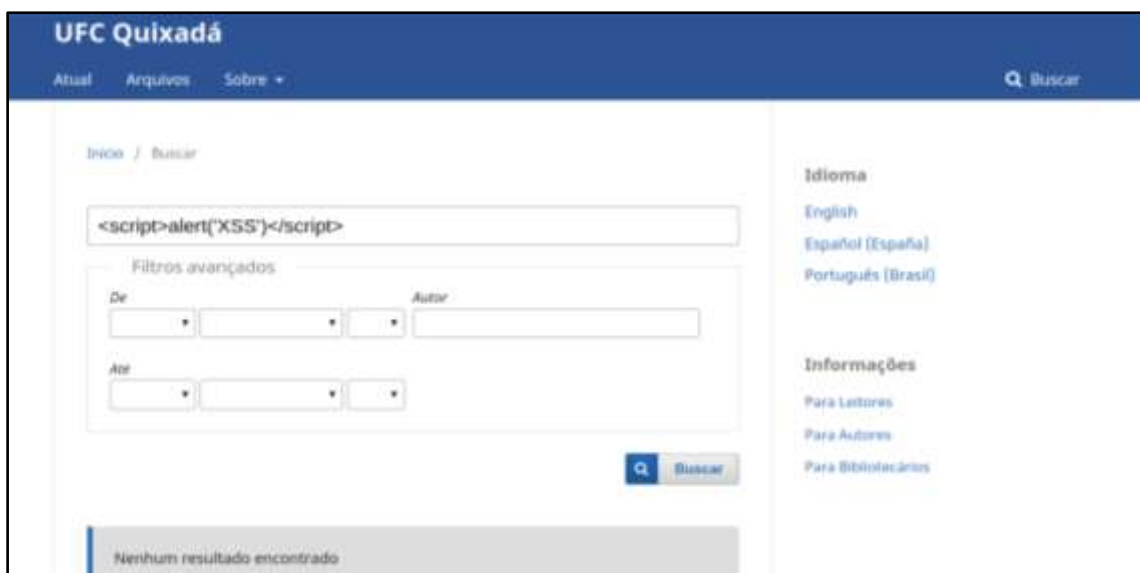
Figura 12 - Exemplo de XSS.



Fonte: Elaborada pelo autor.

O ataque citado acima foi testado na página de busca do OJS, onde usuários podem buscar por artigos publicados em uma revista. O OJS mostrou-se não vulnerável a este tipo de ataque XSS, pois assim como foi visto na seção 5.2.1 deste trabalho, a técnica de *Encoding* foi utilizada para filtrar a entrada de usuários, ou seja, os caracteres especiais < e > foram substituídos por caracteres HTML e, além disso, a entrada do usuário não é mostrada na tela caso a pesquisa não seja encontrada. A Figura 13 ilustra o ataque realizado na página de busca e a Figura 14 ilustra a técnica *Encoding* utilizada para filtrar a entrada dos usuários no campo de busca.

Figura 13 - Ataque XSS realizado na página de busca.



Fonte: Elaborada pelo autor.



Figura 14 - Filtragem utilizada em ataques XSS.

```

<li class="current">
  <h1>
    <h1> Buscar
  </h1>
</li>
</ol>
</nav>
<form class="cmp form" method="post" action="http://54.210.203.165/ojs/index.php/revista/search/search"
  <input type="hidden" name="csrfToken" value="09c36d30733ea49ebe8a53bc7da9cb94">
  <div class="search input">
    <label class="pkp_screen_reader" for="query">
      Pesquisar termo
    </label>
    <input type="text" id="query" name="query"
value="&lt;script&gt;alert(5#039;XSS&#039;)&lt;/script&gt;" class="query" placeholder="Buscar">
  </div>
  <fieldset class="search advanced">
    <legend>
      Filtros avançados
    </legend>
    <div class="date range">
      <div class="from">
        <label class="label">
          De
        </label>
        <select name="dateFromYear">
<option label="" value="" selected="selected"></option>

```

Fonte: Elaborada pelo autor.

### 5.2.3 Emails de usuários encontrados no código fonte da página inicial

Ao examinar o código fonte da página inicial do OJS, foi possível localizar o link <http://200.129.39.72/ojs/index.php/revista/gateway/plugin/WebFeedGatewayPlugin/atom>. Ao clicar neste link, foi possível obter emails de usuários cadastrados no OJS, como é possível ver na Figura 15.

Figura 15 - Emails de usuários do OJS.

```

http://54.210.203.165/ojs/index.php/revista/issue/feed 2019-11-22T23:45:15+00:00 OjsAdmin
denysmaciel22@gmail.com Open Journal Systems <p>Revista da Universidade Federal do Ceará - Campus
Quixadá</p> http://54.210.203.165/ojs/index.php/revista/article/view/2 2019-11-14T12:58:01+00:00 Denys
Pereira Maciel denysmaciel22@alu.ufc.br
<p>Resumo</p>
2019-11-14T12:57:51+00:00 ##submission.copyrightStatement##
http://54.210.203.165/ojs/index.php/revista/article/view/5 2019-11-22T23:40:17+00:00 Bruno de Melo Oliveira
bruno@email.com
<p>Resumo</p>
2019-11-22T23:40:17+00:00 ##submission.copyrightStatement##
http://54.210.203.165/ojs/index.php/revista/article/view/6 2019-11-22T23:45:15+00:00 Iasmyn Magalhães
Fernandes iasmyn@email.com
<p>Resumo</p>
2019-11-22T23:45:14+00:00 ##submission.copyrightStatement##

```

Fonte: Elaborada pelo autor.

Como é possível ver na Figura 15, foi possível encontrar 4 emails de usuários do OJS, entre eles, o email do usuário administrador do sistema (denysmaciel22@gmail.com). Os outros 3 emails são de usuários que realizaram publicações no OJS para testes.

Estes emails encontrados possibilitam ataques de Engenharia Social, que consistem em um atacante utilizar técnicas de persuasão para adquirir a confiança de usuários que utilizam algum sistema que o atacante está tentando invadir. No caso do OJS, um atacante poderia, por exemplo, passar-se pelo administrador de uma revista hospedada no OJS e mandar emails para os usuários requisitando uma troca de senha, comprometendo assim a confidencialidade dos dados de usuários e talvez até da integridade do sistema, dependendo do nível de acesso que um usuário que teve suas informações capturadas tenha no sistema. Esse tipo de ataque é conhecido como *phishing*, que é quando um atacante consegue “fisgar” um usuário para roubar suas informações.

Além de *phishing*, os emails expostos também possibilitam o envio de *spams* para os usuários. *Spams* referem-se a mensagens eletrônicas que são enviadas para um usuário sem o consentimento do mesmo e que, geralmente, são enviadas para um grande número de pessoas. Na maioria dos casos, o conteúdo de *spams* são propagandas, porém, nada impede que um atacante insira um vírus nessas mensagens para infectar as máquinas das vítimas ou até mesmo roubar seus dados.

#### **5.2.4 Upload de arquivos maliciosos**

O padrão utilizado pelo OJS para extensão de arquivos submetidos é o formato PDF (*Portable Document Format*), no entanto, é possível realizar *uploads* de arquivos com extensões diferentes, tais como PHP (PHP: *Hypertext Preprocessor*) e JavaScript.

Durante a pesquisa<sup>1</sup> por vulnerabilidades existentes na versão 3.1.1-4 do OJS, foi possível descobrir que versões mais antigas do OJS eram vulneráveis a este tipo de ataque, pois um usuário tinha acesso a um arquivo submetido no sistema antes que o mesmo fosse oficialmente publicado no sistema. Esse acesso era possível através da URL, pois versões antigas do OJS deixavam o diretório de arquivos submetidos dentro do mesmo diretório do OJS, e esses diretórios podem ser acessados via URL. Um atacante poderia, por exemplo, fazer *upload* de um arquivo malicioso em php que execute comandos passados como parâmetro via URL. Esta vulnerabilidade é conhecida como *Path Traversal* ou *Directory Traversal*, e permite que um invasor leia arquivos executados no sistema, arquivos com

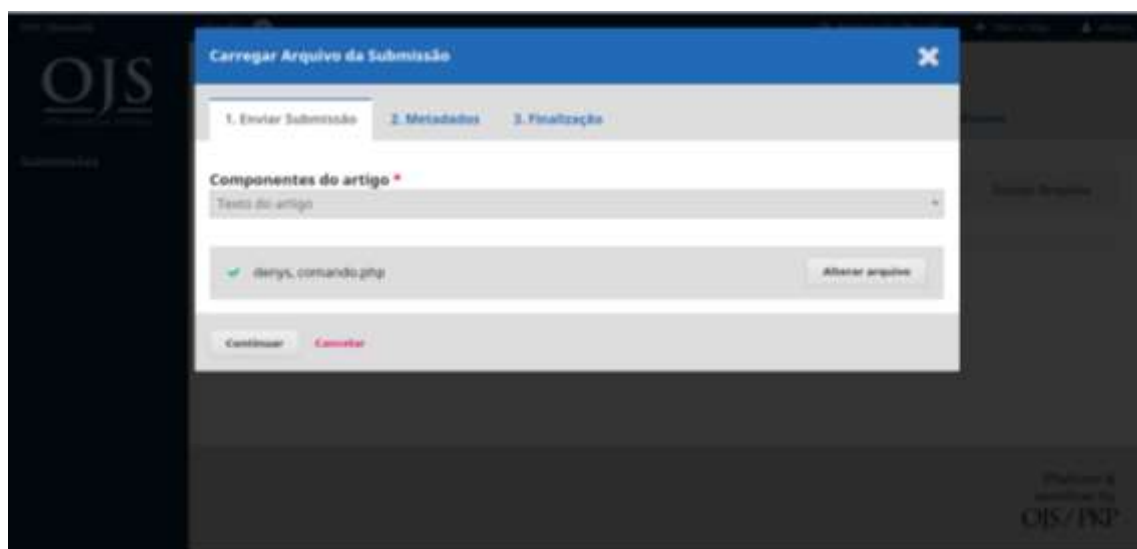
---

<sup>1</sup> <https://www.youtube.com/watch?v=koXBwZ-KE8A>

credenciais do sistema ou até mesmo arquivos confidenciais do sistema operacional. Optou-se por explorar esta vulnerabilidade na versão 3.1.1-4 por não saber se a vulnerabilidade já tinha sido devidamente corrigida na versão utilizada no trabalho.

Para testar esta vulnerabilidade na versão utilizada neste trabalho, um arquivo escrito em PHP que executa comandos via URL foi submetido no sistema, como pode ser visto na Figura 16. O código do arquivo PHP elaborado pelo autor pode ser visto no Apêndice C. Outra possibilidade de exploração desta vulnerabilidade seria o envio de *malwares*, que são códigos maliciosos que, se executados no computador de uma vítima, podem realizar alterações ou roubar informações. Um atacante poderia enviar um *malware* ao invés de um arquivo em PDF. Caso os revisores, por algum descuido, executarem o programa enviado pelo atacante, os mesmos poderiam ter suas informações roubadas ou suas máquinas infectadas por algum vírus.

Figura 16 - Arquivo PHP submetido no sistema.



Fonte: Elaborada pelo autor.

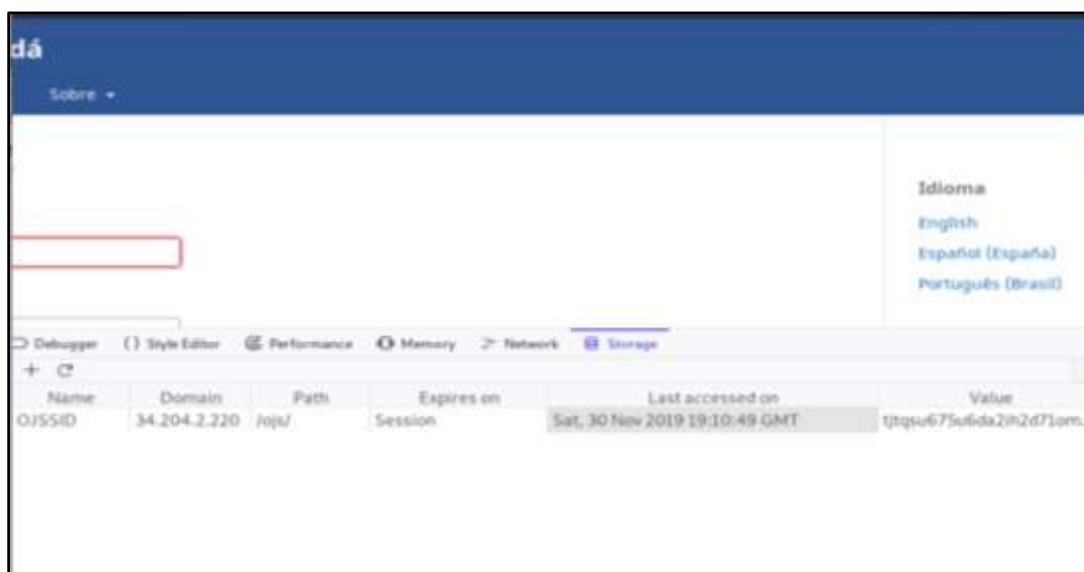
Após a submissão do arquivo, o próximo passo para testar esta vulnerabilidade foi tentar acessá-lo via URL para analisar se era possível executar comandos. A versão utilizada neste projeto mostrou-se não vulnerável contra esta vulnerabilidade, pois, diferente das versões mais antigas, a versão utilizada neste trabalho exige que o diretório para submissão de arquivos (/var/www/files/ foi o diretório utilizado neste trabalho) esteja em um diretório diferente de onde está o OJS (/var/www/html/ojs foi o diretório utilizado neste trabalho). Desta maneira, o diretório de submissões fica inacessível para usuários comuns, o que impede que o arquivo submetido seja executado via navegador.

Para tentar acessar o arquivo submetido, a URL utilizada foi `http://200.129.39.72/./files/`. Em sistemas Linux, a expressão `./` representa voltar um diretório. A URL `http://200.129.39.72/` carrega o arquivo `index.html` do servidor Apache que encontra-se no diretório `/var/www/html/`, ou seja, ao utilizar a URL `http://200.129.39.72/./`, o diretório que está sendo acessado será `/var/www/`. Sabendo disso, basta adicionar a palavra *files* ao final da URL para tentar acessar o diretório de submissões.

### 5.2.5 Tentativa de roubo de sessões através de cookies

Ao examinar o código do sistema, descobriu-se que o OJS utiliza um *cookie* de sessão chamado OJSSID para gerenciar as sessões de usuários logados no sistema. Esse *cookie* é gerado quando a página inicial do OJS é carregada, como é possível ver na Figura 17.

Figura 17 - Geração do *Cookie* OJSSID.



Fonte: Elaborada pelo autor.

Optou-se por realizar esse tipo de ataque por conta de não ter sido possível encontrar informações relacionadas a vulnerabilidades relacionadas a roubo de sessões em nenhuma versão do OJS durante a pesquisa por vulnerabilidades.

Ao realizar *login* no OJS, um novo *cookie* de sessão é gerado e utilizado para gerenciar a sessão do usuário logado, substituindo o *cookie* existente criado quando a página foi carregada. Caso um atacante, de alguma forma, consiga acesso ao *cookie* do usuário, ele poderia utilizá-lo para roubar a sessão desse usuário e assim realizar ações maliciosas passando-se por aquele usuário. Esse tipo de ataque é conhecido como CSRF.

Tentou-se realizar esse ataque ao OJS, utilizando uma aba normal do navegador e outra aba anônima do mesmo navegador. Cada aba possui *cookies* de sessão diferentes. Foi realizado *login* com usuários diferentes em cada aba. O usuário administrador foi utilizado na aba normal e um usuário comum foi utilizado na aba anônima. Após isso, o *cookie* de sessão da aba normal foi copiado e utilizado na aba anônima para analisar se era possível realizar ações do usuário administrador na aba anônima.

Em um primeiro momento este ataque não funcionou, pois no formulário de *login* do usuário existe um campo oculto chamado *csrfToken*, como é possível ver na Figura 18. Ao realizar a alteração do *cookie* na aba anônima, a sessão da aba anônima foi encerrada.

Figura 18 - Campo oculto *csrfToken* no formulário de *login*.

```
132                                     acesso
133                                     </h1>
134                                     </li>
135     </ol>
136 </nav>
137
138 <form class="cmp form cmp form login" id="login" method="post"
139 action="http://3.86.193.156/ojs/index.php/revista/login/signIn">
140   <input type="hidden" name="csrfToken" value="966639bc70cdb0792ff18c7b9bb362e3">
141
142   <input type="hidden" name="source" value="" />
143
144   <fieldset class="fields">
145     <div class="username">
146       <label>
147         <span class="label">
148           Usuário
149         <span class="required">*</span>
150         <span class="pkp_screen_reader">
151           Obrigatório
152         </span>
153       </span>
154       <input type="text" name="username" id="username" value="denys" maxlengt
155     </label>
```

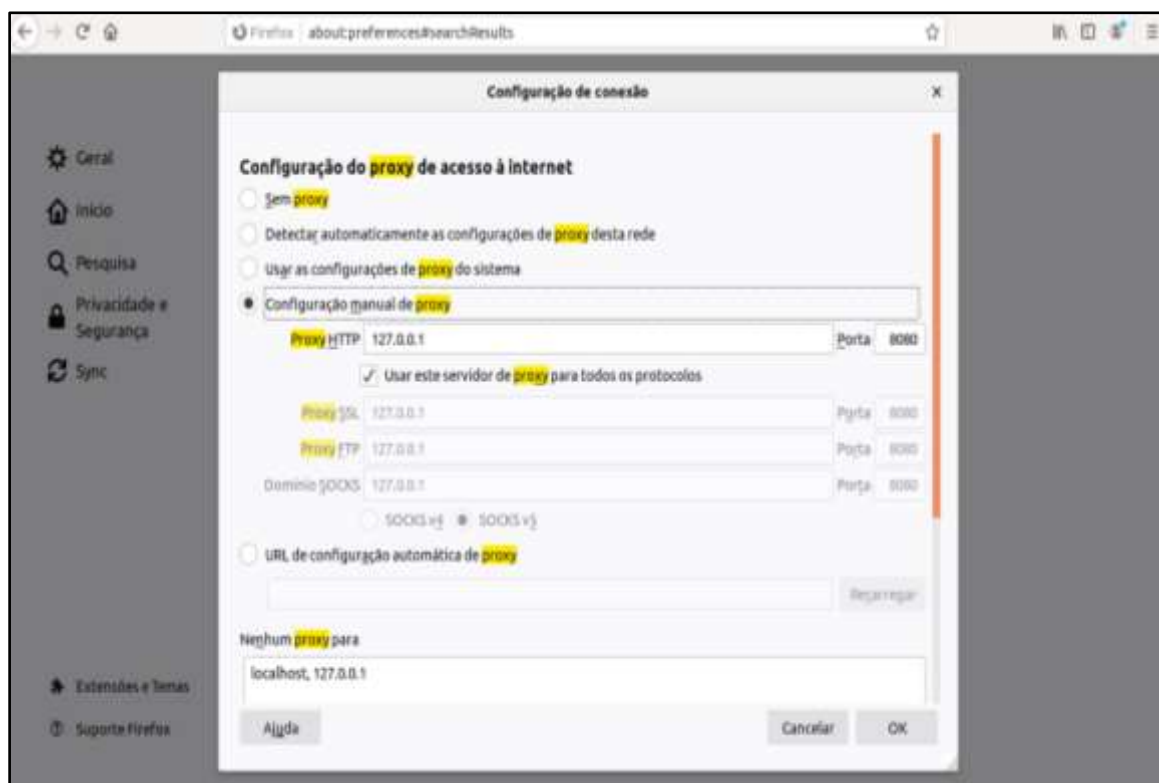
Fonte: Elaborada pelo autor.

Este campo oculto é um *token* que serve para proteger o OJS contra ataques CSRF, gerando um *hash* a partir da função *hash md5*, sempre que uma nova página do OJS é carregada. Este *token* possui duração de uma hora. A função geradora deste *token*, escrita em PHP, foi retirada do código fonte do sistema OJS e pode ser vista no Anexo A. O OJS realiza ações do usuário baseada neste *token* e no *cookie* de sessão a partir do momento em que o usuário realiza *login* no sistema. Para realizar um teste mais aprofundado, a ferramenta Burp Suite foi utilizada. Esta ferramenta atua como um *proxy* e permite a manipulação de requisições a um servidor. O Burp Suite possui uma versão paga e uma grátis, sendo que esta última já vem por padrão no Kali Linux.

Para realizar o ataque, foi preciso que navegador utilizasse um *proxy* com endereço IP 127.0.0.1 e porta 8080, porta padrão utilizada pelo Burp Suite. A configuração do navegador pode ser vista na Figura 19.

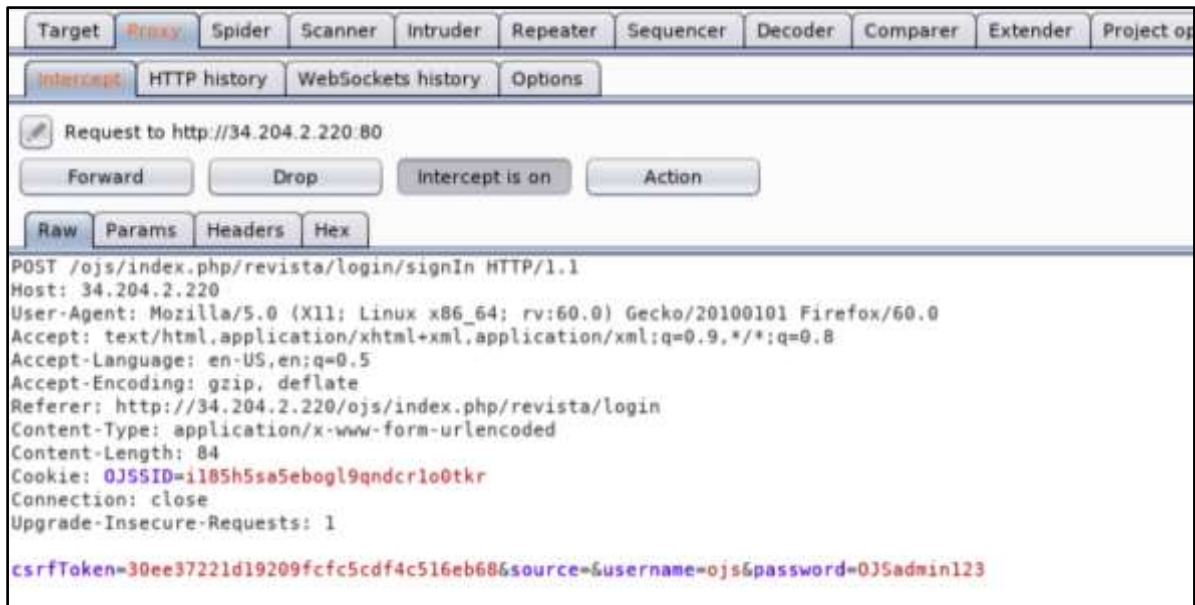
O cenário do teste com a ferramenta foi o mesmo descrito acima, o usuário administrador foi utilizado na aba normal e um usuário comum foi utilizado na aba anônima. Ao abrir a ferramenta Burp Suite, foi utilizada a aba *proxy* com a opção *Intercept is on* habilitada. A opção *Intercept is on* permite que todas as requisições que passem pelo *proxy* utilizado sejam capturadas pelo Burp Suite, permitindo assim que elas sejam manipuladas antes de serem de fato enviadas ao servidor. Ao realizar *login* com o usuário administrador, foi possível ver o conteúdo dos campos *csrfToken* e do *cookie* OJSSID. O mesmo processo foi feito na aba anônima com um usuário comum. A Figura 20 mostra as informações referentes ao usuário administrador e a Figura 21 mostra as informações referentes ao usuário comum através da ferramenta Burp Suite quando foi realizado *login* com ambas as contas.

Figura 19 - Configuração do *proxy* no navegador.



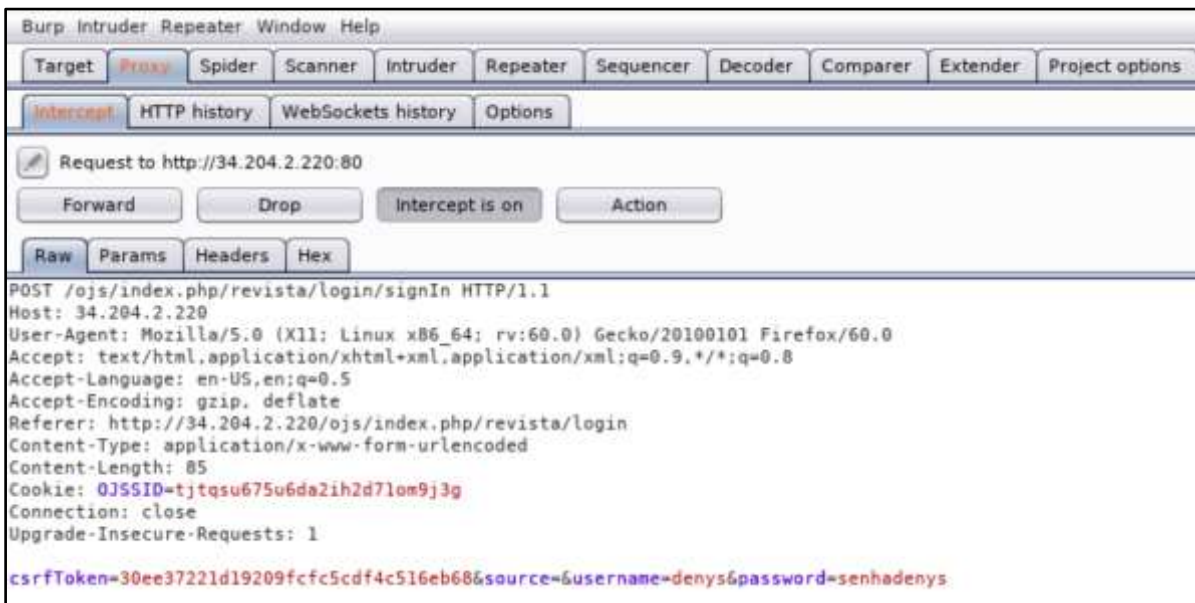
Fonte: Elaborada pelo autor.

Figura 20 - Informações do administrador vistas na ferramenta Burp Suite.



Fonte: Elaborada pelo autor.

Figura 21 - Informações do usuário comum vistas na ferramenta Burp Suite.



Fonte: Elaborada pelo autor.

Ao utilizar a ferramenta Burp Suite como um *proxy*, todas as requisições que passam por ela ficam travadas até que o botão *Forward*, que serve justamente para enviar a requisição ao servidor, seja acionado. Enquanto a requisição está travada no Burp Suite, é possível observar dados contidos na requisição, como *cookies* e *tokens*, por exemplo. Também é possível alterar os dados da requisição antes de enviá-los de fato ao servidor. Ao

capturar a sessão do usuário comum, o conteúdo do campo `csrfToken` foi substituído pelo valor correspondente ao usuário administrador e depois a requisição foi enviada ao servidor clicando no botão *Forward*. Um novo *cookie* de sessão foi gerado, porém, o valor desse *cookie* foi substituído pelo valor gerado na sessão do usuário administrador. Feito isso, tentou-se realizar ações que somente o usuário administrador tem permissão, como por exemplo alterar o nome de uma revista hospedada. O ataque CSRF foi realizado com êxito, pois foi possível realizar esta ação na sessão da aba anônima.

Vale ressaltar que este ataque foi realizado com sucesso porque tinha-se acesso ao navegador utilizado para o *login* do usuário administrador. Em um cenário de ataque real, o atacante precisaria, de alguma maneira, conseguir acesso a máquina que o usuário administrador está utilizando, modificar o *proxy* para o endereço IP do atacante e assim realizar o ataque de roubo de sessão.



## 6. CONCLUSÃO

O objetivo deste trabalho foi detectar e analisar possíveis vulnerabilidades existentes em um servidor OJS local utilizando técnicas de teste de penetração.

Foram utilizadas técnicas de teste de penetração automatizado, através do uso dos *scanners* Acunetix e OWASP ZAP, assim como também foram utilizadas técnicas de teste de penetração manual com o objetivo de encontrar o máximo de vulnerabilidades possíveis.

A versão do OJS utilizada neste trabalho (3.1.1-4) mostrou-se como um sistema web bastante seguro em relação aos ataques realizados durante a execução deste trabalho. Isso pode ser explicado pelo fato do OJS ser um sistema de código aberto e existirem fóruns de discussões feitos pelo PKP, órgão responsável pela criação e lançamento de novas versões do OJS. Nesses fóruns, usuários responsáveis pela administração de revistas expõem problemas encontrados durante o uso do OJS e também dão sugestões sobre como resolvê-las, contribuindo bastante para a contínua evolução da segurança do sistema OJS. A criação desses fóruns, juntamente com a contínua evolução da segurança no sistema, mostra que os desenvolvedores preocupam-se em prover o máximo de segurança possível para aqueles que desejam utilizar o sistema OJS. A Tabela 4 mostra, de forma resumida, informações referentes às vulnerabilidades exploradas neste trabalho. Na Tabela 4 são mostradas as vulnerabilidades exploradas, as consequências que cada vulnerabilidade poderia gerar ao sistema, o nível de criticidade de cada vulnerabilidade, uma forma de mitigação e a forma de mitigação feita pelo OJS.

Para trabalhos futuros, podem-se utilizar outros *scanners* para a detecção de vulnerabilidades no OJS, como por exemplo, o W3AF ou a versão paga do Burp Suite, chamada de Burp Suite PRO, para detectar possíveis vulnerabilidades existentes que não foram detectadas pelas ferramentas utilizadas neste trabalho. Pode-se também realizar um trabalho comparando as vulnerabilidades encontradas em um versão muito antiga do OJS, como por exemplo, a versão 2.2.1, com as vulnerabilidades encontradas na versão mais atual, que hoje é a 3.1.1-4.

Tabela 4 – Vulnerabilidades exploradas.

Vulnerabilidade	Consequência	Nível de criticidade	Forma de mitigação	Mitigação feita pelo OJS
Mensagem de erro da aplicação	Dados expostos do servidor web	Baixo	Ocultar informações de erro	-
Listagem de diretórios	Diretórios e arquivos confidenciais expostos a usuários	Médio	Não deixar diretórios e arquivos expostos	Arquivos expostos não comprometem a segurança do sistema
Credenciais enviadas em texto claro	Roubo de <i>login</i> e senha através da captura do tráfego na rede	Alto	Uso do protocolo HTTPS	-
Senha com preenchimento automático	Atacante com acesso local a máquina pode passar-se por outro usuário	Baixo	Desabilitar a opção de salvar informações de <i>login</i>	-
Tentativas de adivinhação de senhas	Ataques de força bruta	Baixo	Estabelecer um número máximo de tentativas de <i>login</i>	-
SQL Injection	Exposição de dados confidenciais do banco de dados	Alto	Filtrar a entrada de usuários antes de enviar ao banco de dados	Filtragem da entrada de usuários através da técnica <i>Encoding</i>
Cross-site Scripting	Roubo de sessões através de <i>cookies</i> e desfiguração de sites	Alto	Filtrar a entrada de usuários e não exibir a entrada de usuários diretamente na página	Filtragem da entrada de usuários através da técnica <i>Encoding</i>
Emails expostos	<i>Phishing</i> e envio de <i>spam</i>	Médio	Não exibir informações confidenciais de usuários	-
Upload de arquivos maliciosos	Execução de código pela URL da aplicação e inserção de <i>malwares</i> na aplicação	Alto	Estabelecer quais tipos de arquivos os usuários podem submeter no sistema	-
Roubo de sessão	Atacante pode passar-se por outro usuário	Médio	Não deixar <i>cookies</i> e <i>tokens</i> expostos na aplicação	Uso do <i>cookie</i> OJSSID e do <i>token</i> csrfToken

Fonte: Elaborada pelo autor.

## REFERÊNCIAS

SHEBLI, H. M. Z. A.; BEHESHTI, B. D. A study on penetration testing process and tools. 2018 **IEEE Long Island Systems, Applications and Technology Conference (LISAT)**, [S.l.], 2018.

TETSKY, A.; KHARCHENKO, V.; UZUN, D. Neural networks based choice of tools for penetration testing of web applications. 2018 **IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT)**. [S.l.]. 2018.

BERTOGLIO, D. D.; ZORZO, A. F.. Tramonto: Uma estratégia de recomendações para testes de penetração. **XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, v. 1315, [S.l.]. 2016.

BERTOGLIO, D. D.; ZORZO, A. F. **Um mapeamento sistemático sobre testes de penetração**. Porto Alegre: Faculdade de Informática PUC-RS, 2015.

EDGAR, B. D.; WILLINSKY, J. A Survey of Scholarly Journals Using Open Journal Systems. **Scholarly and Research Communication**, v. 1, n. 2, [S.l.], 2010.

BOYLE, R. J.; PANKO, R.R. **Corporate computer security**. Tradução. [S.l.]: Pearson, 2012.

PARMAR, P.; FELEOL, A. **Aplicação de Pentest em Sistemas Computacionais para Análise de Vulnerabilidades: um Estudo de Caso**. [S.l.]: Fundação Centro de Análise, Pesquisa e Inovação Tecnológica (FUCAPI), 2013.

JIMÉNEZ, R. E. L. D. Pentesting on web applications using ethical-hacking. In: 2016 **IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI)**. IEEE, [S.l.], 2016. p. 1-6.

YUNANRI, W.; RIADI, I.; YUDHANA, A. Analisis Deteksi Vulnerability Pada Web Server Open Jurnal System Menggunakan OWASP Scanner. In: **Jurnal Rekayasa Teknologi Informasi**. [S.l.], 2018. p. 1-8.

NAGPURE, S.; KURKURE, S. Vulnerability assessment and penetration testing of Web application. In: **2017 IEEE International Conference on Computing, Communication, Control and Automation (ICCUBEA)**. [S.l], 2017. p. 1-6.

VALDEZ, A. OSSTMM 3. In: **Revista de Información, Tecnología y Sociedad**, p.29. [S.l], 2013.

FERREIRA, F. **Segurança da informação**. [S.l]: Editora Ciência Moderna Ltda. 2003.

## APÊNDICE A - INSTALAÇÃO DO OJS

### Comandos utilizados para instalação do sistema OJS.

#### Instalação das dependências

```
sudo apt update
sudo apt install apache2
sudo apt -y install php php-dev php-pear php-json php-mysql php-xsl php-intl php-gd php-xml php-mbstring
sudo apt install mysql-server
```

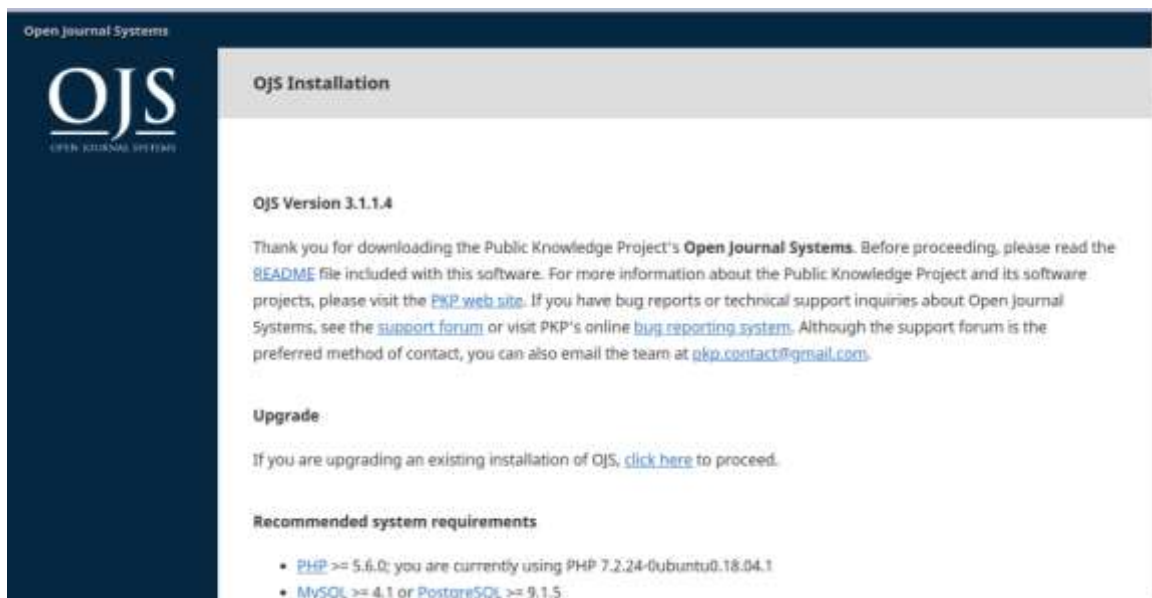
#### Criando e configurando o banco de dados

```
sudo mysql
mysql > create database ojsdb;
mysql > grant all on ojsdb.* to ojs@127.0.0.1 identified by 'OJSadmin123';
mysql > flush privileges;
mysql > exit;
```

#### Baixando o OJS

```
sudo mkdir -p /var/www/files
sudo chmod -R o+w /var/www/files
cd /var/www/html
sudo wget http://pkp.sfu.ca/ojs/download/ojs-3.1.1-4.tar.gz
sudo tar -xzf ojs-3.1.1-4.tar.gz
sudo mv ojs-3.1.1-4 ojs
sudo chmod -R o+w ojs
```

Após a execução de todos os comandos acima, será possível acessar o OJS via navegador para continuar a instalação. Para acessá-lo, escolha um navegador e digite na busca: <http://localhost/ojs>. A Figura 22 ilustra o resultado da busca feita pelo navegador.

Figura 22 - Busca por <http://localhost/ojs> no navegador.

Fonte: Elaborada pelo autor.

Nesta página existem campos que devem ser preenchidos para a criação da conta do administrador do OJS. A Figura 23 mostra os campos que devem ser preenchidos.

Figura 23 - Criação da conta de administrador do OJS.

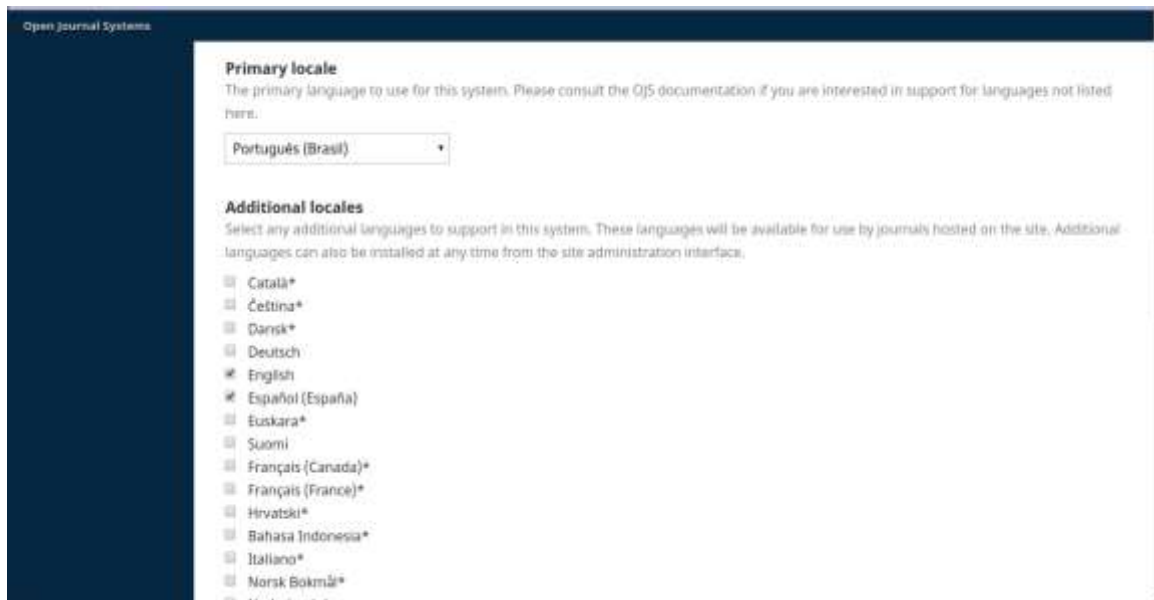
The image shows a web browser window displaying the 'Administrator Account' creation form. The page has a dark blue header with the 'Open Journal Systems' logo. The main content area is white and contains the following text: 'Administrator Account', a note that this user account will become the site administrator, and instructions that additional user accounts can be created after installation. Below this, there are four input fields: 'Username', 'Password', 'Repeat password', and 'Email'. At the bottom, there are 'Locale Settings' instructions regarding UTF-8 support and the mbstring library.

Fonte: Elaborada pelo autor.

Após preencher os campos da conta de administrador, o próximo passo é escolher a língua padrão que será usada pelo OJS na seção *Primary Locale*. A língua padrão escolhida

foi Português (Brasil). Em *Additional Locales*, é possível escolher línguas extras que o OJS poderá utilizar. As línguas extras escolhidas foram *English* e *Español (España)*, conforme pode ser visto na Figura 24.

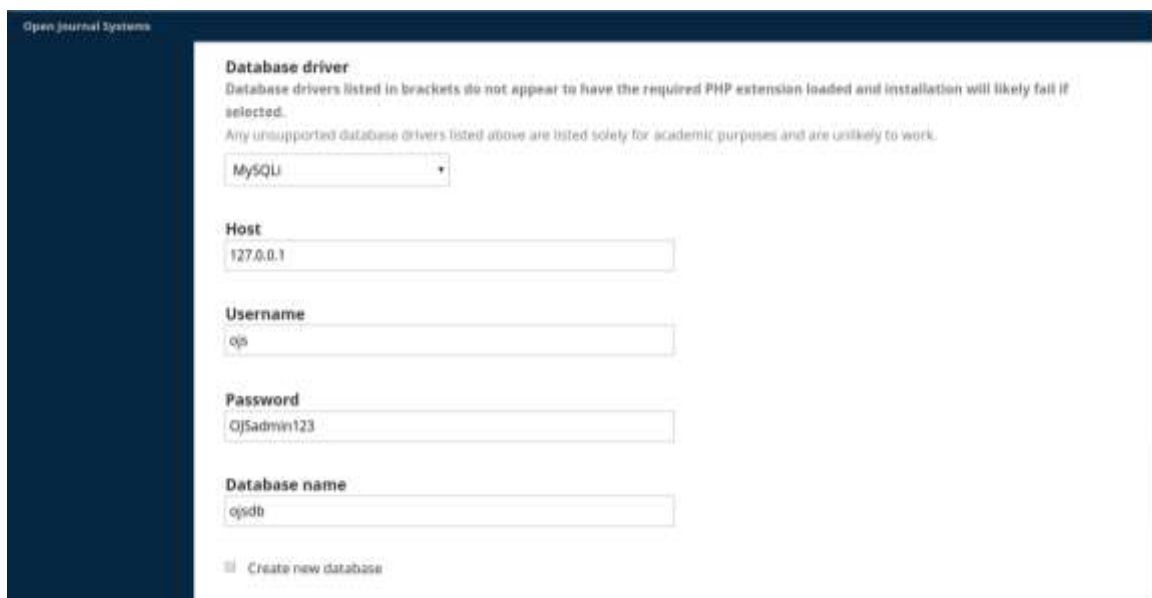
Figura 24 - Escolha das línguas usadas pelo OJS.



Fonte: Elaborada pelo autor.

Após escolher as línguas usadas pelo OJS, o próximo passo é ir até a seção *Database Settings* e realizar a configuração do banco de dados de acordo com os comandos usados no início deste apêndice. Em *Database driver*, selecione a opção MySQLi. No campo *Host*, preencha com o IP 127.0.0.1. O campo *username* foi preenchido com ojs. O campo *password* foi preenchido com OJSadmin123, mesma senha utilizada em um dos comandos do início do apêndice. O campo *Database name* foi preenchido com ojsdb, mesmo nome do banco criado através dos comandos. A opção *Create new database* (criar novo banco de dados) deve ser desmarcada, pois o banco de dados já está criado. Toda a configuração do banco de dados pode ser vista na Figura 25.

Figura 25 - Configurando o banco de dados pelo navegador.

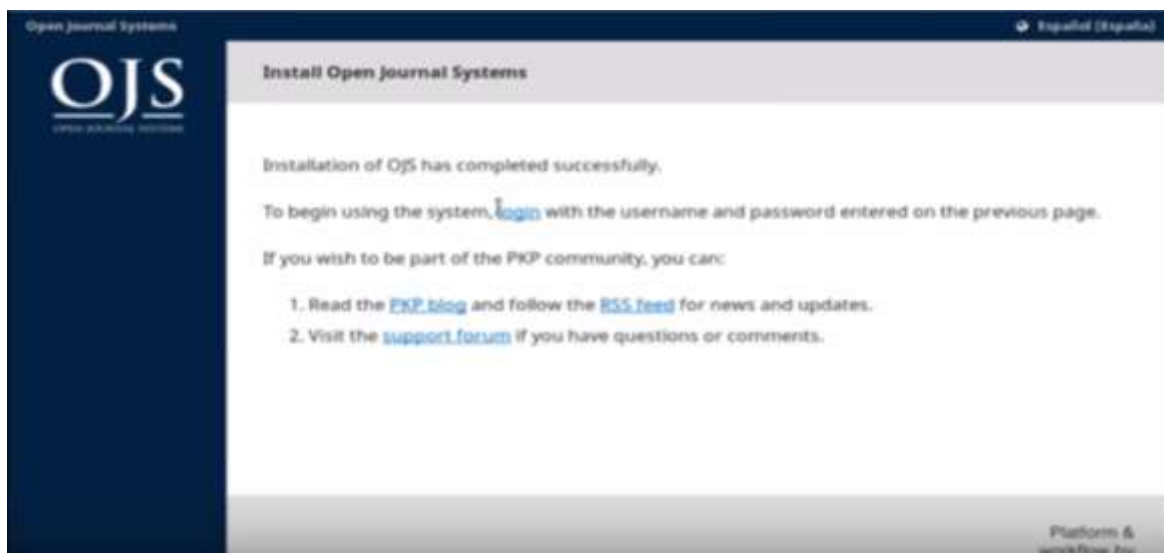


The screenshot shows the 'Open Journal Systems' database configuration interface. It features a dark blue header with the text 'Open Journal Systems'. Below the header, there is a section titled 'Database driver' with a warning message: 'Database drivers listed in brackets do not appear to have the required PHP extension loaded and installation will likely fail if selected. Any unsupported database drivers listed above are listed solely for academic purposes and are unlikely to work.' A dropdown menu is set to 'MySQL'. Below this, there are input fields for 'Host' (127.0.0.1), 'Username' (ojs), 'Password' (OJSadmin123), and 'Database name' (ojsdb). At the bottom left of the form area, there is a checkbox labeled 'Create new database'.

Fonte: Elaborada pelo autor.

Após todas as configurações, basta clicar no botão *Install Open Journal Systems* (Instalar *Open Journal Systems*) ao final da página. Com isso, uma nova página com uma mensagem de sucesso será exibida na tela conforme pode ser visto na Figura 26.

Figura 26 - Tela de confirmação da instalação do OJS.



Fonte: Elaborada pelo autor.



## APÊNDICE B - CRIAÇÃO DE UMA REVISTA NO OJS

Para a criação de uma revista de OJS, é necessário efetuar *login* com a conta de administrador criada durante a instalação. Ao efetuar o *login*, é necessário clicar no botão Criar revista e uma nova janela com os campos Título da revista, Descrição e a URL devem ser preenchidos. Após preencher esses campos, deve-se clicar no botão Salvar ao final da página.

O próximo passo é preencher os campos Editora, ISSN (*International Standard Serial Number*), Resumo da Revista, Equipe Editorial e Sobre a Revista. Preenchidos esses campos, deve-se clicar no botão Continuar ao final da página. Após isso, o próximo passo é preencher os dados do Contato Principal responsável pela revista. Os campos são Nome, Titulação, Email, Telefone e Instituição/Afiliação.

Preenchidos esses campos, o próximo passo é configurar a aparência da revista. Informações como logo, rodapé, cor do cabeçalho da revista devem ser preenchidos nesta seção. Ao final, deve-se clicar em continuar ao final da página.

O próximo passo é configurar as submissões da revista. Os campos Diretrizes para autores (normas bibliográficas e de formatação) e Condições para submissão devem ser preenchidos nesta seção. Ao final, deve-se clicar em continuar ao final da página. Após todas essas configurações, os usuários já existentes no sistema são exibidos na tela, podendo-se adicionar mais ou não. Ao clicar no botão Concluir, a revista terá sido criada.

**APÊNDICE C - CÓDIGO PHP PARA EXECUTAR COMANDOS VIA URL**

```
<?php
  echo shell_exec($_GET['cmd']);
?>
```

Código PHP utilizado para executar comandos via URL.

## ANEXO A - FUNÇÃO GERADORA DO CSRFToken

```

<?php
function getCSRFToken() {
    $csrf = $this->getSessionVar('csrf');
    if (!is_array($csrf) || time() > $csrf['timestamp'] + (60*60)) { // 1 hour token expiry
        // Generate random data
        if (function_exists('openssl_random_pseudo_bytes')) $data = openssl_random_pseudo_bytes(128);
        elseif (function_exists('random_bytes')) $data = random_bytes(128);
        else $data = sha1(mt_rand());

        // Hash the data
        $token = null;
        $salt = Config::getVar('security', 'salt');
        $algos = hash_algos();
        foreach (array('sha256', 'sha1', 'md5') as $algo) {
            if (in_array($algo, $algos)) {
                $token = hash_hmac($algo, $data, $salt);
            }
        }
        if (!$token) $token = md5($data . $salt);

        $csrf = $this->setSessionVar('csrf', array(
            'timestamp' => time(),
            'token' => $token,
        ));
    } else {
        // Extend timeout of CSRF token
        $csrf['timestamp'] = time();
        $this->setSessionVar('csrf', $csrf);
    }
    return $csrf['token'];
}
?>

```

Função geradora do csrfToken<sup>2</sup>.

<sup>2</sup><https://github.com/pkp/pkp-lib/blob/master/classes/session/Session.inc.php>