# Detecção e Análise de Vulnerabilidades em um Servidor OJS utilizando Teste de Penetração

**Denys Maciel[1], Arthur Callado[1], Roberto Filho[1], Marcos Ortiz[1]**

[1]Campus de Quixadá – Universidade Federal do Ceará (UFC)
Av. José de Freitas Queiroz, 5003 – 63.902-580 – Quixadá – CE – Brasil

denysmaciel22@gmail.com, {arthur, rbcabral, mdo}@ufc.br

***Abstract.*** *This article aims to detect and analyze potential vulnerabilities in a local OJS server using penetration testing techniques. For this, the OSSTMM automated penetration testing technique was 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, aiming at to find as many vulnerabilities as possible. The version of OJS evaluated in this article was 3.1.1-4. This release presents controls that inhibit the exploration of the most common vulnerabilities in web systems.*

***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, a técnica de teste de penetração automatizado OSSTMM foi utilizada, 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. A versão do OJS avaliada neste trabalho foi a 3.1.1-4. Esta versão apresenta controles que inibem a exploração das vulnerabilidades mais comuns em sistemas web.*

## 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. Segundo Tetsky e Kharchenko (2018) a oferta de vários serviços online causada pelo desenvolvimento das tecnologias auxiliou o desenvolvimento de 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 [Shebli e 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 a 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".

O OJS (*Open Journal Systems*) é um sistema web para gerenciamento e publicação de revistas e periódicos. Foi lançado 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 e 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 *in loco*, com IP público para acesso à Internet, utilizando técnicas de teste de penetração e ferramentas de *pentest* que auxiliem na detecção das possíveis vulnerabilidades existentes.

Este artigo procede da seguinte forma. A seção 2 apresenta os trabalhos relacionados. A seção 3 apresenta o referencial teórico. A seção 4 apresenta os procedimentos metodológicos. A seção 5 contém os resultados. A seção 6 apresenta a conclusão e os trabalhos futuros.

## 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*. Aqui 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 aos clientes, 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 por esse trabalho foram divididas em quatro tipos: falhas críticas (nenhuma encontrada), falhas de nível alto (uma encontrada), falhas de nível médio (dezesseis encontradas) e falhas de nível baixo (duas encontradas).

O trabalho citado [Parmar e Feleol 2013] relaciona-se com este (nosso) 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.

Já no trabalho de [Yunanri et al. 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. A ferramenta 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.

Os autores do trabalho citado utilizaram a ferramenta OWASP ZAP para determinar o risco das vulnerabilidades encontradas, pois a ferramenta classifica o risco de cada vulnerabilidade encontrada em três níveis (alto, médio e baixo) após o término do *scan* (verificação). 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.

**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 | 2 | 4 |

A relação do trabalho citado 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 realizou testes apenas na versão 3.1.1-4 (versão mais atual do sistema) do OJS, sobre vulnerabilidades comumente encontradas em aplicações web.

Em [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 PRO[1], Acunetix[2] e OWASP ZAP[3]. A Tabela 2 mostra a comparação feita entre as vulnerabilidades encontradas no teste manual e as vulnerabilidades encontradas pelas ferramentas utilizadas nos testes automatizados.

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

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

[1] https://portswigger.net/burp

[2] https://www.acunetix.com/web-vulnerability-scanner/demo/

[3] https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

A relação do trabalho citado com este se dá pelo fato de em ambos os trabalhos, as ferramentas Acunetix, OWASP ZAP e Burp Suite serem 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.

## 3. Referencial Teórico

Esta seção apresenta conceitos importantes relacionados a este trabalho.

### 3.1. 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].

### 3.2. 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 ao(s) testador(es) ter que pesquisar sobre informações e reunir o máximo de dados possíveis sobre o alvo para, assim, conseguir explorar possíveis vulnerabilidades com base 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.

### 3.3. Metodologia 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 deve ser coletado 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 ou geração de relatórios, pois não existem clientes envolvidos para discussão de escopo nem para entrega de relatórios aos 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.

## 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.

### 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 64bits LTS (*Long Term Support*), 4GB (Gigabyte) de memória RAM e 50 GB de armazenamento.

### 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 à procura de vulnerabilidades. Essas ferramentas facilitam bastante a vida de profissionais que estão realizando *pentests* em sistemas.

### 4.3. Topologia de rede deste trabalho

A Figura 1 ilustra a topologia de rede utilizada neste trabalho.

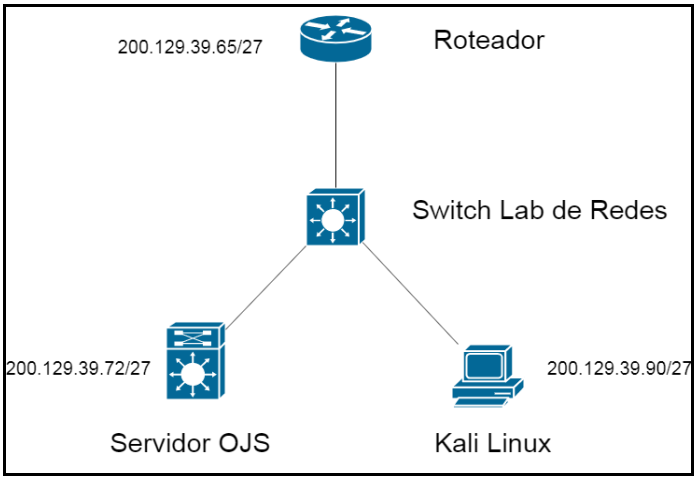


**Figura 1. Topologia de rede deste trabalho**

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.

### 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. 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 e analisar os resultados

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. 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.

### 4.6. Escanear o OJS com a ferramenta OWASP ZAP e analisar os resultados

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. 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. 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.

### 4.7. 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

Nesta seção os resultados coletados serão apresentados.

### 5.1. *Scan* com a ferramenta Acunetix

Após a conclusão do escaneamento, o próximo passo foi explorar as vulnerabilidades pela ferramenta para analisar como as vulnerabilidades encontradas poderiam prejudicar a segurança do OJS.

#### 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, como por exemplo http://200.129.39.72/urlnaoexiste. O IP 200.129.39.72 é o endereço utilizado pela máquina OJS. Ao inserir uma URL como essa, informações como o *software* servidor web que está sendo utilizado, a versão do sistema web, o sistema operacional utilizado e a porta que o servidor está utilizando podem ser vistas.

Um 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.

#### 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 2 mostra a URL http://200.129.39.72/ojs/lib/pkp sendo acessada através de um navegador.

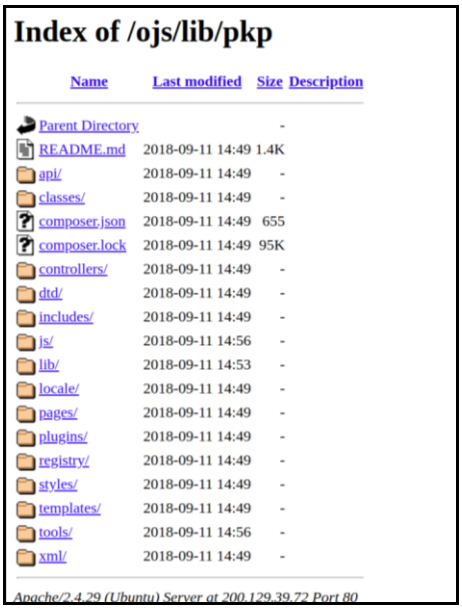


**Figura 2. URL http://200.129.39.72/ojs/lib/pkp sendo acessada**

Como é possível ver na Figura 2, 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.

### 5.1.3. Credenciais do usuário enviadas em texto claro

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 sua senha, passariam em texto claro durante o tráfego na rede. Com a utilização de uma ferramenta que capture o tráfego que passa por uma rede, como por exemplo, a ferramenta Wireshark, seria possível que um atacante conseguisse acesso aos dados confidenciais de usuários que realizaram *login* no OJS através de uma captura de tráfego.

### 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.*

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 mudar sua senha, por exemplo.

### 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.

### 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.

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*, pois as entradas que os usuários colocam nos campos de *login* e senha são filtradas antes de serem enviadas ao interpretador através de uma técnica conhecida como *Encoding*, que consiste em converter caracteres especiais para caracteres HTML.

#### 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>.

O ataque citado acima foi testado na página de busca do OJS, onde usuários podem buscar por artigos publicados em uma revista, porém, a técnica *Encoding* também filtrou a entrada de usuários, ou seja, os caracteres especiais < e > foram substituídos por caracteres HTML.

#### 5.2.3. E-mails encontrados no código fonte da página inicial

Ao examinar o código fonte da página inicial do OJS, foi possível localizar um link que exibe informações sobre arquivos submetidos. Entre essas informações, estavam e-mails de usuários utilizados neste trabalho para submissões de testes.

Estes e-mails 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 e-mails 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 e-mails 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. No entanto, é possível realizar *uploads* de arquivos com extensões diferentes, tais como PHP ou JavaScript. Durante a pesquisa 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 executasse comandos passados como parâmetro via URL. Esta vulnerabilidade é conhecida como *Path Traversal*, e permite que um invasor leia arquivos executados no sistema, arquivos com credenciais do sistema ou até mesmo arquivos confidenciais do sistema operacional.

Para testar esta vulnerabilidade na versão utilizada neste trabalho, um arquivo escrito em PHP que executa comandos via URL foi submetido no sistema. 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 a 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.

### 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. 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.

Tentou-se realizar esse ataque ao OJS, utilizando uma aba normal do navegador e outra aba anônima do mesmo navegador. 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.

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

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. O cenário do teste com a ferramenta foi o mesmo descrito acima. Ao abrir a ferramenta Burp Suite, foi utilizada a aba *proxy* com a opção *Intercept is on* habilitada. Esta opção 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. Ao utilizar a ferramenta Burp Suite como um *proxy*, todas as requisições que passam por ela ficam travadas até que o botão *Forward* seja acionado. Enquanto a requisição está travada no Burp Suite, é possível observar e alterar dados contidos na requisição.

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 através do 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 teve êxito, pois foi possível realizar a ação na sessão da aba anônima.

Vale ressaltar que este ataque foi realizado com sucesso porque se tinha 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. A Tabela 3 sumariza as vulnerabilidades exploradas neste trabalho.

**Tabela 3. Vulnerabilidades exploradas**

| Vulnerabilidade | Consequência | Nível de criticidade | Forma de mitigação | Mitigação feita pelo OJS | Vulnerabilidade |
|---|---|---|---|---|---|
| Mensagem de erro da aplicação | Dados expostos do servidor web | Baixo | Ocultar informações de erro | - | Mensagem de erro da aplicação |
| 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 | Listagem de diretórios |
| Credenciais enviadas em texto claro | Roubo de *login* e senha pela captura do tráfego na rede | Alto | Uso do protocolo HTTPS | - | Credenciais enviadas em texto claro |
| Senha com preenchimento automático | Atacante com acesso a máquina pode passar-se por outro usuário | Baixo | Desabilitar a opção de salvar informações de *login* | - | Senha com preenchimento automático |

| | | | | | |
|---|---|---|---|---|---|
| Tentativas de adivinhação de senhas | Ataques de força bruta | Baixo | Estabelecer um número máximo de tentativas de login | - | Tentativas de adivinhação de senhas |
| 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 Encoding | SQL Injection |
| Cross-site Scripting | Roubo de sessões através de cookies 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 Encoding | Cross-site Scripting |
| Emails expostos | Phishing e envio de spam | Médio | Não exibir informações confidenciais de usuários | - | Emails expostos |
| Upload de arquivos maliciosos | Execução de código pela URL da aplicação e inserção de malwares na aplicação | Alto | Estabelecer quais tipos de arquivos os usuários podem submeter no sistema | - | Upload de arquivos maliciosos |
| Roubo de sessão | Atacante pode passar-se por outro usuário | Médio | Não deixar cookies e tokens expostos na aplicação | Uso do cookie OJSSID e do token csrfToken | Roubo de sessão |

Um resumo do grau de risco das vulnerabilidades encontradas neste trabalho é mostrado na Tabela 4.

**Tabela 4. Vulnerabilidades encontradas por este trabalho**

| Versão do OJS | Número de vulnerabilidades de risco alto | Número de vulnerabilidades de risco médio | Número de vulnerabilidades de risco baixo |
|---|---|---|---|
| 3.1.1-4 | 4 | 3 | 3 |

## 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 um sistema web bastante seguro em relação aos ataques realizados durante a execução deste trabalho. As principais vulnerabilidades não estão relacionadas ao sistema em si, mas à configuração da infraestrutura usada (uso de HTTPS no servidor, segurança das pastas do servidor, tipos de arquivos autorizados para upload, roubo de cookies no cliente etc.). 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.

Para trabalhos futuros, podem-se utilizar outros *scanners* para a detecção de vulnerabilidades no OJS, por exemplo, o W3AF ou a versão paga do Burp Suite, para detectar possíveis vulnerabilidades que não foram detectadas pelas ferramentas utilizadas neste trabalho. Pode-se também comparar as vulnerabilidades encontradas em uma 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.

## Referências


Shebli, H. M. Z. A. e Beheshti, B. D. (2018) "A study on penetration testing process and tools", In: **IEEE Long Island Systems, Applications and Technology Conference** (LISAT).

Bertoglio, D. D. e Zorzo, A. F. (2015) "Um mapeamento sistemático sobre testes de penetração", In: **Relatório Técnico: no. 84**. Porto Alegre: Faculdade de Informática PUC-RS.

Bertoglio, D. D. e Zorzo, A. F. (2016) "Tramonto: Uma estratégia de recomendações para testes de penetração", In: **XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**.

Edgar, B. D e Willinsky, J. (2010) "A Survey of Scholarly Journals Using Open Journal Systems", In: **Scholarly and Research Communication**.

Jiménez, R. E. L. D (2016) "Pentesting on web applications using ethical-hacking", In: **IEEE 36th Central American and Panama Convention** (CONCAPAN XXXVI).

Nagpure, S. e Kurkure, S. (2017) "Vulnerability assessment and penetration testing of Web application", In: **IEEE International Conference on Computing, Communication, Control and Automation** (ICCUBEA).

Parmar, P. e Feleol, A. (2013) "Aplicação de Pentest em Sistemas Computacionais para Análise de Vulnerabilidades: um Estudo de Caso", In: **Encontro Regional de Computação e Sistemas de Informação** (ENCOSIS 2013). Fundação Centro de Análise, Pesquisa e Inovação Tecnológica (FUCAPI).

Tetsky, A. e Kharchenko, V. (2018) "Neural networks based choice of tools for penetration testing of web applications", In: **IEEE 9th International Conference on Dependable Systems, Services and Technologies** (DESSERT).

Valdez, A. (2013) "OSSTMM 3", In: **Revista de Información, Tecnología y Sociedad**.

Yunanri, W., Riadi, I. e Yudhana, A. (2018) "Analisis Deteksi Vulnerability Pada Web Server Open Jurnal System Menggunakan OWASP Scanner", In: **Jurnal Rekayasa Teknologi Informasi**.