
**UNIVERSIDADE FEDERAL DO CEARÁ**
**CAMPUS QUIXADA**
**GRADUAÇÃO EM REDES DE COMPUTADORES**

**FELIPE FERREIRA SAMPAIO**

# UMA ANÁLISE PRÁTICA DAS PRINCIPAIS VULNERABILIDADES EM APLICAÇÕES WEB BASEADO NO TOP 10 OWASP

**QUIXADA**
**2021**

FELIPE FERREIRA SAMPAIO

# UMA ANÁLISE PRÁTICA DAS PRINCIPAIS VULNERABILIDADES EM APLICAÇÕES WEB BASEADO NO TOP 10 OWASP

Trabalho de Conclusão de Curso apresentado ao Curso de Tecnologia em Redes de Computadores do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Redes de Computadores.

Orientador: Prof. Dr. João Marcelo Uchôa de Alencar

QUIXADA
2021

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)

---

S183a Sampaio, Felipe Ferreira.
Uma análise prática das principais vulnerabilidades em aplicações web baseado no top 10 OWASP / Felipe Ferreira Sampaio. – 2021.
60 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Redes de Computadores, Quixadá, 2021.
Orientação: Prof. Dr. João Marcelo Uchôa de Alencar.

1. Projeto Aberto de Segurança em Aplicações Web.. 2. Segurança computacional. 3. Aplicações web. I. Título.

CDD 004.6

---

FELIPE FERREIRA SAMPAIO

# UMA ANÁLISE PRÁTICA DAS PRINCIPAIS VULNERABILIDADES EM APLICAÇÕES WEB BASEADO NO TOP 10 OWASP

Trabalho de Conclusão de Curso apresentado ao Curso de Tecnologia em Redes de Computadores do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de tecnólogo em Redes de Computadores.

Aprovada em: ___/___/___

BANCA EXAMINADORA

_____

Prof. Dr. João Marcelo Uchôa de Alencar (Orientador)
Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Jefferson de Carvalho Silva
Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Jeandro de Mesquita Bezerra
Universidade Federal do Ceará (UFC)

## AGRADECIMENTOS

Aos meus pais pelo carinho e apoio que me deram ao longo dessa jornada.

Aos meus colegas de trabalho da NetworkSecure pela experiência de trabalhar como pentester.

Ao time de CTF IFCE pelas competições disputadas e conhecimento adquirido.

Aos professores que me deram apoio no campus de Quixadá.

Ao meu orientador por tomar esse papel importante na minha formatura.

À Universidade Federal do Ceará pelo conhecimento adquirido.

## RESUMO

Com a popularização das aplicações *web*, falhas de segurança também foram surgindo e sendo cada vez mais exploradas por hackers. Dentro disso a OWASP, uma entidade reconhecida internacionalmente que busca aumentar a segurança de *softwares* ao redor do mundo, passou a divulgar periodicamente a cada 4 anos uma lista das vulnerabilidades em aplicações *web* mais visadas por atacantes. Essa lista é denominada Top 10 OWASP, que fora utilizada como base para esse trabalho, ao qual tem como objetivo analisar de maneira prática, as 10 vulnerabilidades divulgadas no Top 10 OWASP mais recente até o momento da escrita deste trabalho. A análise de cada vulnerabilidade será dividida em 3 etapas, sendo elas: descrição, demonstração e mitigação. A partir do detalhamento do processo por trás de cada problema, ao final do trabalho será descrito uma série de tópicos, que poderão servir de auxílio para que desenvolvedores possam validar, se seus projetos em aplicações *web* seguem os princípios básicos para serem considerados seguros.



**Palavras-chave:** Projeto Aberto de Segurança em Aplicações Web. Segurança computacional. Aplicações web.

## ABSTRACT

With the popularization of web applications, security flaws were also emerging and being increasingly exploited by hackers. Within this, OWASP, an internationally recognized entity that seeks to increase the security of software around the world, began to periodically disclose every 4 years a list of vulnerabilities in web applications most targeted by attackers. This list is called Top 10 OWASP, which was used as the basis for this work, which aims to analyze in a practical way, the 10 vulnerabilities disclosed in the most recent Top 10 OWASP at the time of writing this work. The analysis of each vulnerability will be divided into 3 stages, namely: description, demonstration and mitigation. From the detailing of the process behind each problem, at the end of the work a series of topics will be described, which can help developers to validate if their projects in web applications follow the basic principles to be considered safe.



**Keywords:** Open Web Application Security Project. Computer security. Web Applications.

## LISTA DE ILUSTRAÇÕES

## LISTA DE TABELAS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| API | Application Programming Interface |
| CMS | Content Management System |
| CSS | Cascading Style Sheets |
| DNS | Domain Name System |
| DOM | Document Object Model |
| DTD | Document Type Definition |
| HTML | HyperText Markup Language |
| HTTP | Hyper Text Transport Protocol |
| ID | Identificação Digital |
| IP | Internet Protocol |
| OS | Operating System |
| OSI | Open Systems Interconnection |
| OWASP | Open Web Application Security Project |
| PHP | Hypertext Preprocessor |
| POO | Programação Orientada a Objetos |
| RCE | Remote Command Execution |
| S3 | Terceiro segmento sacral |
| SO | Sistema Operacional |
| SQL | Structured Query Language |
| URL | Uniform Resource Locator |
| XML | EXtensible Markup Language |
| XSS | Cross-site scripting |

**SUMÁRIO**

# 1 INTRODUÇÃO

A *internet* como conhecemos pode executar aplicações de diversos tipos e funcionalidades diferentes, porém, no início um dos primeiros ambientes em que o usuário doméstico teve a oportunidade de interagir foram os aplicativos *web*, eles sempre foram o foco principal quando o assunto é "navegar" em busca de informações pela internet. Dentro disso, observando o uso típico da internet pela maioria dos usuários atualmente, ainda é possível afirmar que o principal meio de navegação são aplicações *web*, ou aplicações que executam sob o protocolo HTTP (*Hypertext Transfer Protocol*), que é responsável por transmitir os hipertextos os quais interagimos diariamente em nossos *desktops* e dispositivos móveis, seja em busca de notícias, jogos, redes sociais e outros meios que estamos acostumados a usufruir (LINS, 2013).

Recentemente com a popularização de aplicativos, tanto versões *desktop* quanto versões móveis, viu-se que muitas vezes o usuário ficava preso a uma determinada instalação da aplicação ao qual necessitava utilizar, tendo que sempre estar com o mesmo dispositivo para ter acesso à aplicação. Diante disso foram surgindo implementações dos programas e aplicações tradicionais para versões *web* ou também chamadas de *online* onde sua utilização torna-se muito mais portátil, possibilitando ao usuário apenas saber o endereço *web* da aplicação em questão, e então poderia utilizar de qualquer lugar, podendo citar como exemplos atuais, a versão *WhatsApp Web* e *Telegram Web*, onde o utilizador não precisa ter o *software* instalado no dispositivo, basta apenas visitar o endereço e ter acesso às funcionalidades da aplicação como se a mesma fosse uma instalação local em sua máquina ou *smartphone*, porém, sem consumir o armazenamento em disco necessário de uma instalação tradicional, fazendo disso um dos principais motivos para se optar por acessar versões *web* de *softwares* tradicionais (QUINN, 2010).

Diante de toda essa abundância da utilização de tecnologias *web* quanto à migração ou expansão de programas tradicionais para versões *web*, também foram descobertas milhares de vulnerabilidades na maioria das aplicações desenvolvidas, que acabam colocando em risco dados sensíveis de usuários e de organizações que fazem a utilização de aplicações *web*. No ano de 2015 o CERT.br (Grupo de Resposta a Incidentes de Segurança para a Internet no Brasil) apontou que ocorreram cerca de 722.205 notificações incidentes de segurança relacionados a servidores *web*, onde os atacantes aproveitam-se de vulnerabilidades *web* presentes nos sistemas alvos para hospedar páginas falsas do sistema financeiro que são utilizados para roubar credenciais dos usuários. Em alguns casos também utilizam dos servidores vulneráveis para a prática de *spam* de modo a alcançar o maior número de vítimas possíveis para as páginas clonadas (NIC.BR, 2016).

Dentro disso, uma entidade denominada OWASP (*Open Web Application Security Project*) que é uma comunidade *online* sem fins lucrativos, reconhecida internacionalmente, atua com o objetivo de aumentar a segurança de *softwares* ao redor do mundo, também trabalha em pesquisas e levantamentos sobre diversas categorias de vulnerabilidades que mais impactam os sistemas e aplicações na internet, vem divulgando através de estudos uma lista das top 10 vulnerabilidades em aplicações *web* mais recorrentes de tempos em tempos, que foi o documento

utilizado como base para este projeto.(OWASP, 2017a)

Nesse contexto, esse trabalho teve como objetivo fazer uma análise prática das principais vulnerabilidades em aplicações *web* de acordo com a OWASP, sendo cada tópico divido em descrição, demonstração e mitigação. Tal estudo será capaz de servir de auxílio para analistas de segurança como também desenvolvedores, que necessitem entender o processo por trás de cada vulnerabilidade do top 10. Ao fim da explanação de todos os tópicos, será descrito uma lista dos principais passos de mitigação para cada falha abordada, que servirá de auxílio para desenvolvedores a fim de criar aplicações mais seguras, ou aumentar a segurança de aplicativos já disponíveis.

# 2 OBJETIVOS

## 2.1 Objetivo Geral

Demonstrar de maneira prática, o processo por trás das vulnerabilidades em aplicações *web* mais recorrentes dos últimos anos, baseando-se na metodologia da OWASP. Com isso obter uma lista das principais recomendações para tornar uma aplicação mais segura.

## 2.2 Objetivos específicos

- Expor métodos de exploração e mitigação para cada vulnerabilidade.
- Elaborar um roteiro sobre boas práticas que devem ser implementadas para que uma aplicação *web* seja considerada segura.

# 3 TRABALHOS RELACIONADOS

Nessa seção serão discutidos trabalhos relacionados destacando as semelhanças e diferenças com o projeto desenvolvido nesse documento.

## 3.1 Análise de Vulnerabilidades Web e de Ferramentas de Código Aberto Para Exploração

Novaski Neto (2019) apresentou um trabalho que teve como base o OWASP Top 10 *web*. Tendo como objetivo analisar 3 vulnerabilidades relacionadas a falta de sanitização da entrada do usuário na aplicação, sendo elas: Injeção (A1), *Cross Site Scripting* (A7) e Entidade Externa do XML (A4). Essa análise foi executada com a utilização de 14 *softwares* de código aberto selecionados pelo autor. Os testes ocorreram em um laboratório da própria OWASP chamado *WebGoat*, trata-se de um ambiente com diversas vulnerabilidades a serem exploradas para fins didáticos. Como resultado foi possível avaliar quais utilitários de código aberto se saíram melhores para a exploração da falha abordada.

A análise proposta no trabalho citado também baseia-se na metodologia da OWASP, porém, aborda todas as 10 vulnerabilidades do Top 10. Novaski Neto (2019) tem como objetivo analisar o desempenho de 14 softwares. Entretanto, este trabalho busca detalhar o processo de exploração por trás de cada vulnerabilidade, também com o auxílio de laboratórios, porém não se restringindo a utilização de ferramentas específicas. Ferramentas adequadas para cada caso foram selecionadas para as devidas demonstrações.

## 3.2 Understanding The Top 10 OWASP Vulnerabilities

Bach-Nutman (2020) em seu trabalho pela *Bournemouth University*, executou um estudo mais teórico sobre as vulnerabilidades presentes no último top 10 OWASP *web* de 2017. Cada tópico busca descrever os detalhes da falha, bem como demonstrar alguns exemplos ilustrativos. Ao final, discutem-se os possíveis impactos que empresas e usuários estarão sujeitos, em casos de ataques bem sucedidos direcionados ao problema de segurança discutido.

O trabalho citado também tem como base o Top 10 OWASP, e tem como objetivo o entendimento de todas as falhas do documento, entretanto, busca uma visão mais teórica sobre os problemas abordados. Por outro lado, este trabalho visa demonstrações de maneira prática, onde foram utilizados ambientes onde cada vulnerabilidade pode ser reproduzida em passos.

## 3.3 OWASP top ten - What is the state of practice among startups?

Søhoel (2018) em seu trabalho fez um teste de segurança em 5 servidores *web* desenvolvidos por *startups*. As *startups* foram entrevistadas e os testes de penetração baseados no top 10 OWASP foram conduzidos. Os resultados mostram que nenhuma das empresas estava utilizando uma abordagem ampla no aspecto de segurança. Duas das cinco empresas estavam

familiarizadas com a OWASP e haviam tomado algumas medidas arbitrárias para proteger-se contra as vulnerabilidades conhecidas. As outras três empresas pareciam pouco preocupadas com a segurança e tinham um ponto de vista de que a segurança deveria ser implementada assim que o serviço estivesse funcionando e estável.
Os testes mostram que as três empresas que não se preocupam com a segurança, todas apresentam sérias falhas de segurança. Para as duas empresas que trabalham ativamente com segurança, embora algumas falhas de segurança tenham sido encontradas, elas eram menos grave. Em geral, as empresas tiveram mais sucesso em evitar falhas de implementação, como injeção de SQL e XSS, enquanto as falhas de segurança relacionadas a arquitetura do sistema eram mais comuns. A injeção de SQL e o XSS também foram mais amplamente conhecidas entre as *startups* do que as outras dez principais vulnerabilidades do OWASP.

O trabalho citado também tem como base o Top 10 OWASP, porém tem como objetivo avaliar a segurança de 5 startups em relação às falhas apontadas pela OWASP. Por outro lado, este trabalho busca demonstrar ao leitor o processo por trás de cada uma das falhas, ao final é elaborado um roteiro, onde o leitor pode verificar suas próprias aplicações e validar se segue recomendações básicas para ser considerada segura.

## 3.4 Mitigando Ataque Aplicando Boas Práticcas No Desenvolvimento Seguro de Aplicações Web

Cabral Neto e Brandão (2019) buscaram demonstrar de forma prática 4 das 10 vulnerabilidades do último top 10 OWASP, sendo elas: A1-Injeção, A2-Quebra de Autenticação, A4 - Entidade Externa do XML e A7 - Cross Site Scripting. As demonstrações de exploração contaram com o auxílio do laboratório bWAPP, uma ferramenta que possui diversas vulnerabilidades para serem exploradas para fins didáticos. Após cada exploração também foram abordadas correções para o problema discutido, com o intuito de apresentar boas práticas no desenvolvimento seguro de aplicações web.

O trabalho citado é o mais similar a este em desenvolvimento, buscando demonstrar de maneira prática as falhas do Top 10 OWASP com a utilização de laboratórios, porém Cabral Neto e Brandão (2019) demonstram 4 das 10 vulnerabilidades. Por outro lado, esse trabalho demonstra também de maneira prática todas as falhas da OWASP que são passíveis de laboratórios.

## 3.5 Um guia para análise de segurança de aplicativos na plataforma Android

Em seu trabalho de conclusão de curso pela Universidade Federal do Ceará, Campus Quixadá, Gomes (2017) fez um guia para análise de segurança de aplicativos na plataforma Android, que também se embasava em um documento da OWASP o Mobile Top 10 2016. É utilizado as top 10 vulnerabilidades em aplicações móveis que constam no documento da OWASP para a construção do guia que pode ser de auxílio para auditorias de segurança em

aplicações do sistema *Android*. Também foi abordado as sugestões de soluções para cada uma das vulnerabilidades abordadas.

O trabalho citado também baseia-se na metodologia da OWASP, utilizando o Top 10 OWASP Mobile, que são as 10 principais vulnerabilidades em aplicações para smartphones, o autor também faz demonstrações de maneira prática e tem como objetivo a criação de um guia para análise de segurança em aplicativos *Android*. Por outro lado, este trabalho baseia-se o Top 10 OWASP de vulnerabilidades em aplicações *web*, tendo como objetivo o entendimento por trás do processo de cada vulnerabilidade citada.

## 3.6 Comparativo

Na tabela 1 foram selecionados seis tópicos para comparação de outras pesquisas com o trabalho proposto, é possível observar que todos na metodologia da OWASP. Três dos trabalhos não utilizam a lista completa, e um dos trabalhos utiliza o Top 10 OWASP de vulnerabilidades em aplicações móveis, e não em aplicações web como o restante dos estudos.

**Tabela 1 – Comparação entre trabalhos**

| | NOVASKI, 2019 | BACH-NUTMAN, 2020 | SØHOEL, 2018 | CABRAL;BRANDÃO, 2019 | GOMES,2017 |
|---|---|---|---|---|---|
| Metodologia OWASP | ( X) | ( X ) | ( X ) | ( X ) | ( X ) |
| Análise | ( X) | | | ( X ) | ( X ) |
| Laboratórios | ( X) | | | ( X ) | ( X ) |
| Lista OWASP Completa | | ( X ) | ( X ) | | ( X ) |
| Top 10 Web | ( X) | ( X ) | ( X ) | ( X ) | |
| Top 10 Android | | | | | ( X ) |

Fonte: Autor

# 4 FUNDAMENTAÇÃO TEÓRICA

A seguir, serão apresentados os conceitos utilizados neste trabalho.

## 4.1 Segurança da Informação

A segurança da informação é a garantia da confidencialidade, integridade, disponibilidade, autenticidade e não repúdio de todas as informações e dados indispensáveis para uma organização e/ou indivíduo. Essa garantia contempla tanto ambientes físicos como virtuais.

De acordo com Sêmola (2014), "podemos definir segurança da informação como uma área de conhecimento dedicada à proteção de ativos da informação contra acessos não autorizados, alterações indevidas ou sua indisponibilidade."

Conforme ressalta o autor, a segurança da informação pode ser definida como uma área que exige dos profissionais conhecimentos específicos. Esses profissionais devem garantir que as informações não sofram alterações ou acesso insuficiente, e que as informações estejam sempre disponíveis para acesso autorizado.

A segurança da informação tem um grande desafio, conscientizar o setor humano sobre sua importância, como ressalta Ferreira e Araújo (2008) , "os colaboradores devem manter suas senhas como informação confidencial. Não deve ser permitido compartilhá-las, nem acessar sistemas de outros colaboradores sem autorização expressa".

Por todas essas razões, é correto afirmar que, ter uma boa gestão de segurança da informação não é uma tarefa fácil, porém com os conhecimentos adequados e realizando com eficiência cada etapa, no final terá um ambiente corporativo seguro. Assim no final teremos como garantir a confidencialidade, integridade, disponibilidade e autenticidade das informações da organização.

## 4.2 Vulnerabilidade

Vulnerabilidade ou falha é um elo fraco de determinado objeto, ou sistema, no caso deste trabalho são abordadas as vulnerabilidades que estão presentes em aplicações *web*. Se um sistema possui uma vulnerabilidade, dependendo de sua gravidade é possível obter total acesso ao mesmo, podendo modificá-lo e acessar informações que antes eram consideradas confidenciais, como dados dos usuários da aplicação (NEGRI MORENO, 2017).

Neste trabalho serão abordadas as principais vulnerabilidades em sistemas que rodam sobre o protocolo HTTP, comumente conhecidos como aplicações *web.* De acordo com a metodologia da OWASP, será visto desde a descrição de uma vulnerabilidade até o processo de exploração e formas de mitigação da mesma.

## 4.3 OWASP

A OWASP (*Open Web Application Security Project*) é uma entidade *online* reconhecida internacionalmente, que busca aumentar a segurança de *softwares* ao redor do mundo, seu objetivo maior é tornar visível a segurança das aplicações, de modo que organizações e pessoas consigam elaborar decisões conscientes em relação aos principais riscos de segurança em suas aplicações. Para atingir seu objetivo, a OWASP desenvolve projetos diversos, que podem ser documentações ou *softwares*, todos sob licenças livres, sendo seu acesso prático e fácil.
Entre seus vários projetos, há a lista das top 10 de vulnerabilidades *web* que é divulgado periodicamente a cada três ou quatro anos, nessa lista estão documentadas as falhas mais comuns e mais exploradas em aplicações *web* ao redor do mundo, tal lista é baseada na experiência de diversos profissionais da área de segurança da informação onde os mesmos compartilham conhecimentos, e elegem os riscos mais críticos no presente cenário das ameaças existentes que podem afetar as aplicações.

Pesquisadores e auditores de segurança utilizam essa metodologia da OWASP para se guiarem quando desejam seguir um padrão reconhecido para procurar por falhas, ou demonstrar em análises e estudos como funcionam as vulnerabilidades em aplicações *web* (OWASP, 2017a).

## 4.4 OWASP Top 10 2017

Lista das top 10 vulnerabilidades presentes no documento da OWASP que serão estudadas e demonstradas em laboratórios.

- A1 - Injeção
- A2 - Quebra de Autenticação
- A3 - Exposição de Dados Sensíveis
- A4 - Entidades Externas de XML (XXE)
- A5 - Quebra de Controle de Acesso
- A6 - Configurações de Segurança Incorretas
- A7 - Cross-Site Scripting (XSS)
- A8 - Desserialização Insegura
- A9 - Utilização de Componentes Vulneráveis
- A10 - Registro e Monitorização Insuficiente

(OWASP, 2017b)

# 5 METODOLOGIA

Esta seção tem como objetivo expor a metodologia utilizada para construção desse trabalho.

## 5.1 Laboratórios

Para cada vulnerabilidade a ser demonstrada, foi escolhido a maneira mais prática para reproduzi-la, dentro disso, para algumas foi necessário a criação de laboratórios locais simples. Para o restante fora utilizados laboratórios como *WebSecurity Academy* e *Vulnweb* das instituições *Portswigger* e *Acunetix* respectivamente. Organizações que possuem como foco a conscientização e ensino sobre segurança em aplicações *web* para desenvolvedores e analistas de segurança. Algumas vulnerabilidades tornariam-se complexas para a criação de um laboratório do zero, portanto, os ambientes citados foram de apoio para este trabalho.

## 5.2 Definir ordem das vulnerabilidades

Como todas as vulnerabilidades analisadas contaram com laboratórios independentes, foi definido a demonstração na mesma ordem da lista da OWASP, ou seja, da falha mais explorada para a menos explorada.

## 5.3 Descrição da falha

Inicialmente para cada tópico, foi explicado de forma teórica o processo envolvido em cada item, como a falha ocorria e o que o sucesso de sua exploração poderia causar em uma aplicação. Como fonte de informação foi utilizado o conteúdo do *WebSecurity Academy* da (PORTSWIGGER, ).

## 5.4 Demonstração da falha

Foi utilizado um laboratório para 8 das falhas estudadas, e abordado todos os passos para o sucesso de sua exploração. Sendo considerado sucesso obter acesso aos dados sensíveis da aplicação, ou causar alguma anomalia na mesma, que não seria possível sem a presença da falha. A demonstração do tópico “A6 - Configurações de Segurança Incorretas“ não necessitava de laboratório, e o tópico ”A10 - Registro e monitorização insuficiente“ não necessitava de demonstração devido seu processo ser mais subjetivo.

## 5.5 Métodos de correção e mitigação

Após demonstrar de maneira prática o processo por trás de cada falha, foi abordado as principais maneiras de mitigação para o problema em questão, alguns casos necessitavam apenas implantar um filtro para bloquear determinadas entradas do usuário, sendo preciso alterar seu código fonte.

No caso de aplicações de terceiros, na maioria das vezes é necessário atualizar componentes da aplicação, quando se utiliza um CMS como *WordPress* por exemplo, onde muitas vezes tem-se o uso de *plugins* desatualizados, então é necessário baixar novos recursos com as atualizações de segurança mais recentes, tendo em sua documentação todos os detalhes de uma possível correção de segurança.

## 5.6 Lista de boas práticas

Após ter sido abordado desde a descoberta da vulnerabilidade até sua exploração e correção, foi discutido boas práticas a serem seguidas para a construir uma aplicação segura, ou verificar se uma aplicação atende os requisitos para ser considerada segura de acordo com a metodologia da OWASP.

Dentro das 10 principais vulnerabilidades em aplicações *web* demonstradas, é possível encontrar vulnerabilidades que levam a outras, ou também no caso da correção de uma em específica acabar corrigindo outras que ainda não se tinha conhecimento, por exemplo, a atualização de um componente vulnerável, que futuramente poderiam surgir novas falhas para aquela versão que estava em uso.

Foram criados tópicos a serem seguidos e verificados na aplicação, para determinar se a mesma segue as normas de boas práticas, que resultam do estudo das 10 principais vulnerabilidades em aplicações *web* da OWASP.

# 6 ESTUDO

Esta seção tem como objetivo demonstrar em detalhes cada vulnerabilidade do top 10 OWASP, incluindo exemplos práticos, métodos de correção e mitigação.

## 6.1 A1 - Injeção

Essa falha ocorrem porque a entrada controlada pelo usuário é interpretada como comandos ou parâmetros reais pelo aplicativo. Os ataques de injeção dependem de quais tecnologias estão sendo utilizadas e como exatamente a entrada é interpretada por essas tecnologias. Alguns exemplos comuns incluem:

- Injeção de SQL: Ocorre quando a entrada controlada pelo usuário é passada para consultas SQL. Como resultado, um invasor pode encaminhar novas consultas SQL na requisição, para manipular o resultado da solicitação.
- Injeção de comando: Ocorre quando a entrada do usuário é passada para os comandos do sistema. Como resultado, um invasor pode executar comandos arbitrários no *shell* do servidor da aplicação.

Se um invasor conseguir passar uma entrada interpretada corretamente, ele poderá fazer o seguinte:

- Acessar, modificar e excluir informações em um banco de dados quando essa entrada for passada para as consultas do banco. Isso significa que um invasor pode roubar informações confidenciais, como dados pessoais de usuários e credenciais.
- Executar comandos arbitrários no servidor da aplicação, que permitiria a um invasor obter acesso aos sistemas dos usuários. Isso permitiria que ele roubasse dados confidenciais e realizasse mais ataques contra a infraestrutura vinculada ao servidor no qual o comando é executado.

### *6.1.1 OS Command Injection*

A injeção de comando ocorre quando o código do lado do servidor faz uma chamada de sistema na máquina hospedada. É uma vulnerabilidade que permite que um invasor aproveite a chamada de sistema feita para executar comandos no *shell* no servidor.

Às vezes, isso nem sempre termina em algo malicioso, como um simples '*whoami*' ou apenas leitura de arquivos. Isso não é tão ruim. Mas o problema em relação à injeção de comandos é que possibilita muitas opções para o invasor. Algo crítico que ele poderia fazer seria gerar um *shell* reverso para se tornar o usuário com o qual o servidor da *web* está sendo

executado. Um simples; '*nc -e /bin/bash*' é tudo o que é necessário para obter um *shell* reverso em um servidor linux.

Uma vez que o invasor tenha um shell reverso no servidor web, ele pode tentar escalar privilégios para torna-se um usuário *root*, que terá controle sobre todo o servidor.

### *6.1.2 Exemplo*

Considerando um cenário hipotético: A empresa X iniciou o desenvolvimento em um *shell* baseado na *web*, mas acidentalmente o deixou exposto na Internet. O desenvolvimento não está concluído, mas é o suficiente para que um invasor consiga explorar a vulnerabilidade de injeção de comandos.

Abaixo tem-se o código vulnerável

**Código 6.1 – Código vulnerável**

```
<?php
if (isset($_GET["cmd"])) {
    $cmdString = $_GET["cmd"];
    passthru($cmdString);
}
?>
```

O código acima faz o seguinte:

1) Verifica se o parâmetro "*cmd*" está definido
2) Se for, a variável '*$cmdString*' obtém o que foi passado para o campo de entrada
3) O programa então executa a função '*passthru($cmdString)*' que executa o que é inserido na entrada e, em seguida, passa a saída diretamente de volta para o navegador.

### *6.1.3 Demonstração*

Abaixo na figura 1 temos o código acima em execução, onde um atacante utilizando do parâmetro 'cmd' passou os valores 'date;pwd;ls' onde a saída desse conjunto de comandos é retornada ao usuário.

**Figura 1 – Laboratório Injeção de comandos**


Fonte: Autor

### *6.1.4 Mitigação*

A principal maneira para evitar ataques de injeção é garantir que a entrada de usuários não seja interpretada como consultas ou comandos. Existem diferentes maneiras de fazer isso:

- Utilizando uma lista de permissões: Quando a entrada é enviada ao servidor, esses dados são comparados a uma lista de requisições seguras ou caracteres. Se for considerada segura, será processada. Caso contrário, deverá ser rejeitada e o aplicativo gerar um erro.
- Retirando a entrada: Se a entrada apresentar caracteres perigosos, esses dados serão removidos antes de serem processados.

Caracteres ou entradas perigosas são classificados como qualquer entrada que pode alterar a forma como os dados posteriores são processados. Não é necessário construir listas de permissões manualmente ou até mesmo eliminar a entrada, existem várias bibliotecas que já realizam essas ações.

## 6.2 A2 - Quebra de autenticação

A autenticação e o gerenciamento de sessão constituem os principais componentes das aplicações *web* modernas. A autenticação permite que usuários obtenham acesso a áreas restritas das aplicações, verificando suas identidades. A maneira mais comum de autenticação é utilizando um formulário de login e senha.

Após um usuário inserir suas credenciais, o servidor então verifica. Se estiverem corretas, o servidor fornecerá ao navegador do usuário um *cookie* de sessão. Um *cookie* de sessão é necessário, pois os servidores da *web* utilizam SSL/TLS para se comunicar, que não possuem estado. Integrar *cookies* de sessão significa que o servidor saberá quem está enviando quais dados. O servidor pode então rastrear as ações dos usuários na aplicação.

Se um invasor conseguir encontrar falhas em um mecanismo de autenticação, ele obterá acesso às contas de outros usuários. Isso permitiria ao invasor acessar dados confidenciais (dependendo da finalidade do aplicativo).

Algumas falhas comuns nos mecanismos de autenticação incluem:

- Ataques de força bruta: Se uma aplicação *web* utiliza nomes de usuário e senhas, um invasor pode efetuar ataques de força bruta que permitem tentar adivinhar as credenciais, com base em várias tentativas de autenticação.
- Uso de credenciais fracas: Os aplicativos da *web* devem definir políticas de senhas fortes. Se os aplicativos permitirem que os usuários definam senhas como ’123456’ ou senhas comuns, um invasor será capaz de adivinhá-las facilmente e acessar contas de usuários.

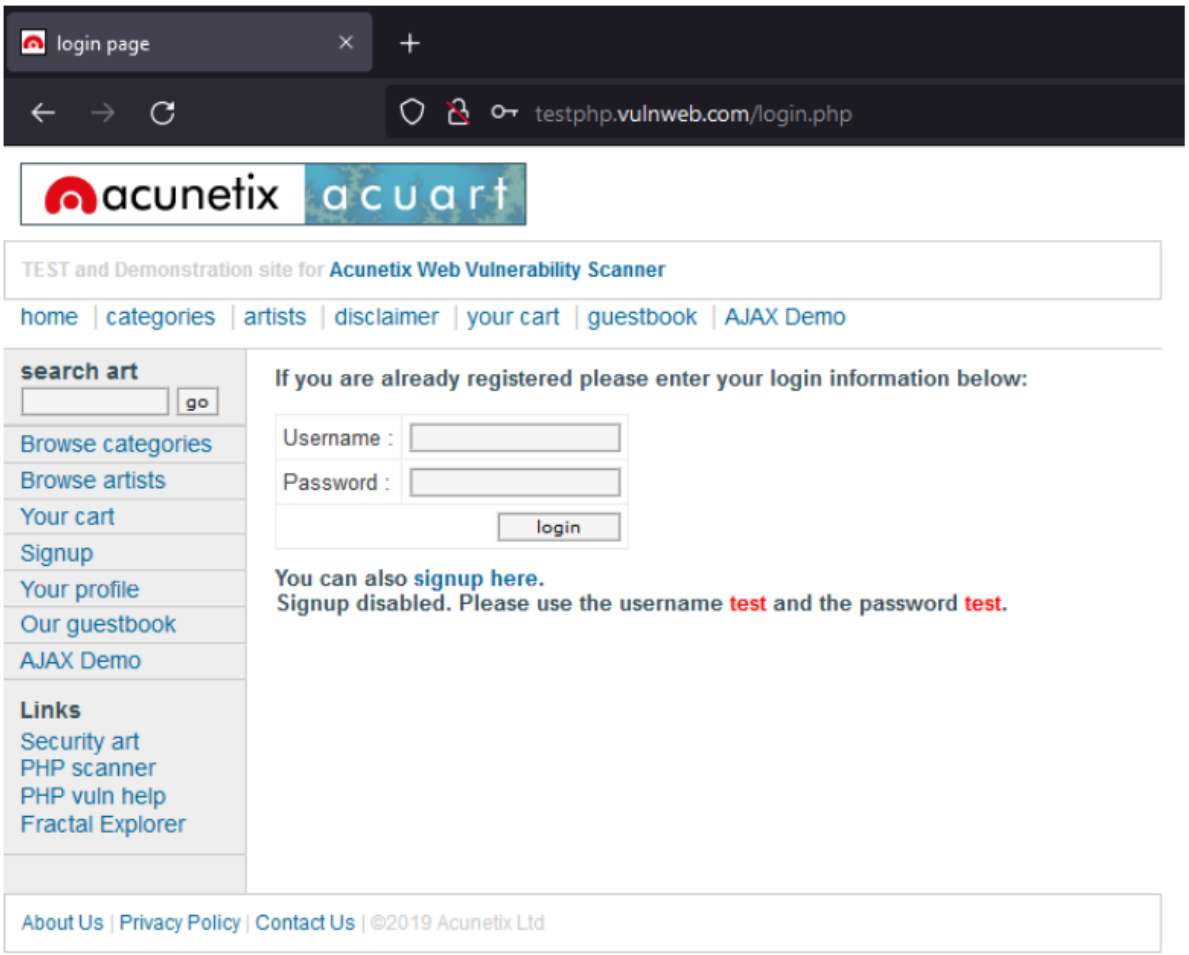
- *Cookies* de sessão fracos: Os *cookies* de sessão são a forma como o servidor rastreia os usuários. Se os *cookies* de sessão contiverem valores previsíveis, um invasor pode definir seus próprios *cookies* de sessão e acessar contas de usuários.

### *6.2.1 Demonstração*

Nessa demonstração será utilizado o laboratório de vulnerabilidades *Vulnweb* da empresa de segurança *Acunetix*, que dispõe de um ambiente preparado para receber testes de segurança para fins didáticos.

No exemplo abaixo figura 2 tem-se um formulário de login, o atacante então poderia tentar adivinhar as credenciais com base em senhas comuns, para isso seria necessária alguma ferramenta que automatize o processo para aumentar as chances de sucesso.

**Figura 2 – Laboratório Quebra de Autenticação**




Fonte: Autor

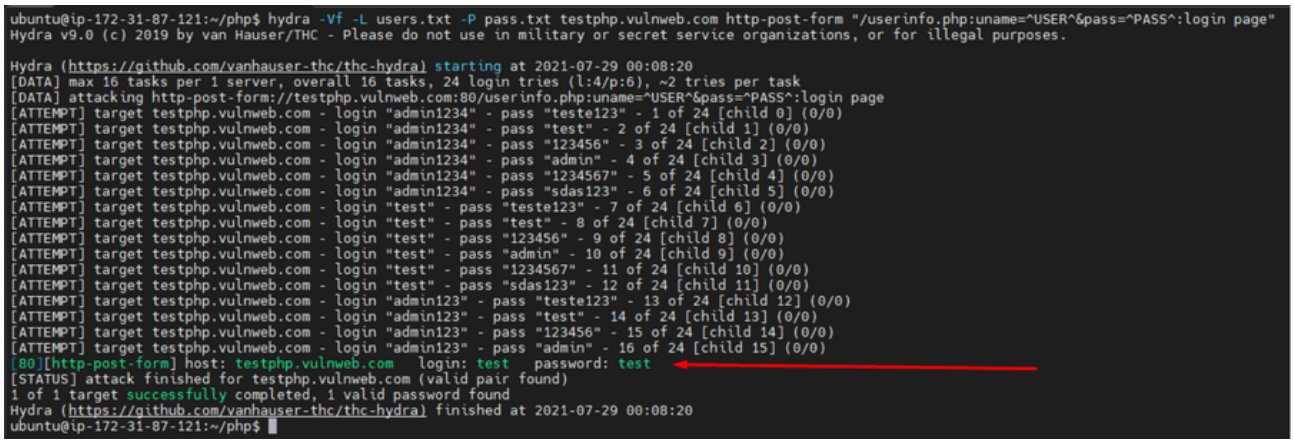
Abaixo na tabela 2 foi gerada uma tabela simples para efetuar um ataque de força bruta no formulário da aplicação acima.

**Tabela 2 – Wordlist**

| Usuário | Senha |
|---|---|
| admin1234 | admin |
| test | admin123 |
| admin123 | 123456 |
| admin | test |

Fonte: Autor

**Figura 3 – Hydra**




Fonte: Autor

`Comando: $ hydra -Vf -L users.txt -P pass.txt testphp.vulnweb.com http-post-form "/userinfo.php:uname=^USER^&pass=^PASS^:login page"`

No exemplo acima na figura 3 foi utilizada a ferramenta *THC Hydra* que permite efetuar ataques de força bruta em diversos protocolos rapidamente. Foi definido para que a *wordlist* acima fosse testada de maneira combinada, ou seja, todos os usuários foram testados com todas as senhas. Dessa forma, poucos segundos foram necessários para adivinhar a credencial da aplicação (test;test).

### *6.2.2 Mitigação*

- Para evitar ataques de adivinhação de senha, é preciso certificar que o aplicativo aplique uma política de senha forte.
- Para evitar ataques de força bruta, deve-se verificar que o aplicativo aplique um bloqueio automático após um certo número de tentativas inválidas. Isso evitaria que um atacante efetuasse novos ataques de força bruta.

- Implementar autenticação multifator: Se um usuário possuir vários métodos de autenticação, por exemplo, utilizando nome de usuário e senha, e recebendo um código em seu dispositivo móvel. Seria difícil para um invasor obter acesso à ambas as credenciais para obter invadir a sua conta.

## 6.3 A3 - Exposição de dados sensíveis

Quando um site divulga acidentalmente dados confidenciais, muitas vezes, são dados diretamente ligados aos clientes (nomes, datas de nascimento, informações financeiras etc.), mas também podem ser informações mais técnicas, como nomes de usuários e senhas.

Em níveis mais complexos, isso muitas vezes envolve técnicas como um "Ataque *Man in The Middle*", em que o invasor forçaria as conexões do usuário através de um dispositivo que ele controla e, em seguida, aproveitaria a criptografia fraca em quaisquer dados transmitidos para obter acesso às informações trafegadas.

Muitos exemplos são muito mais simples, e vulnerabilidades podem ser encontradas em sites que podem ser explorados sem nenhum conhecimento avançado de rede. Em alguns casos, os dados confidenciais podem ser encontrados diretamente no próprio servidor de hospedagem.

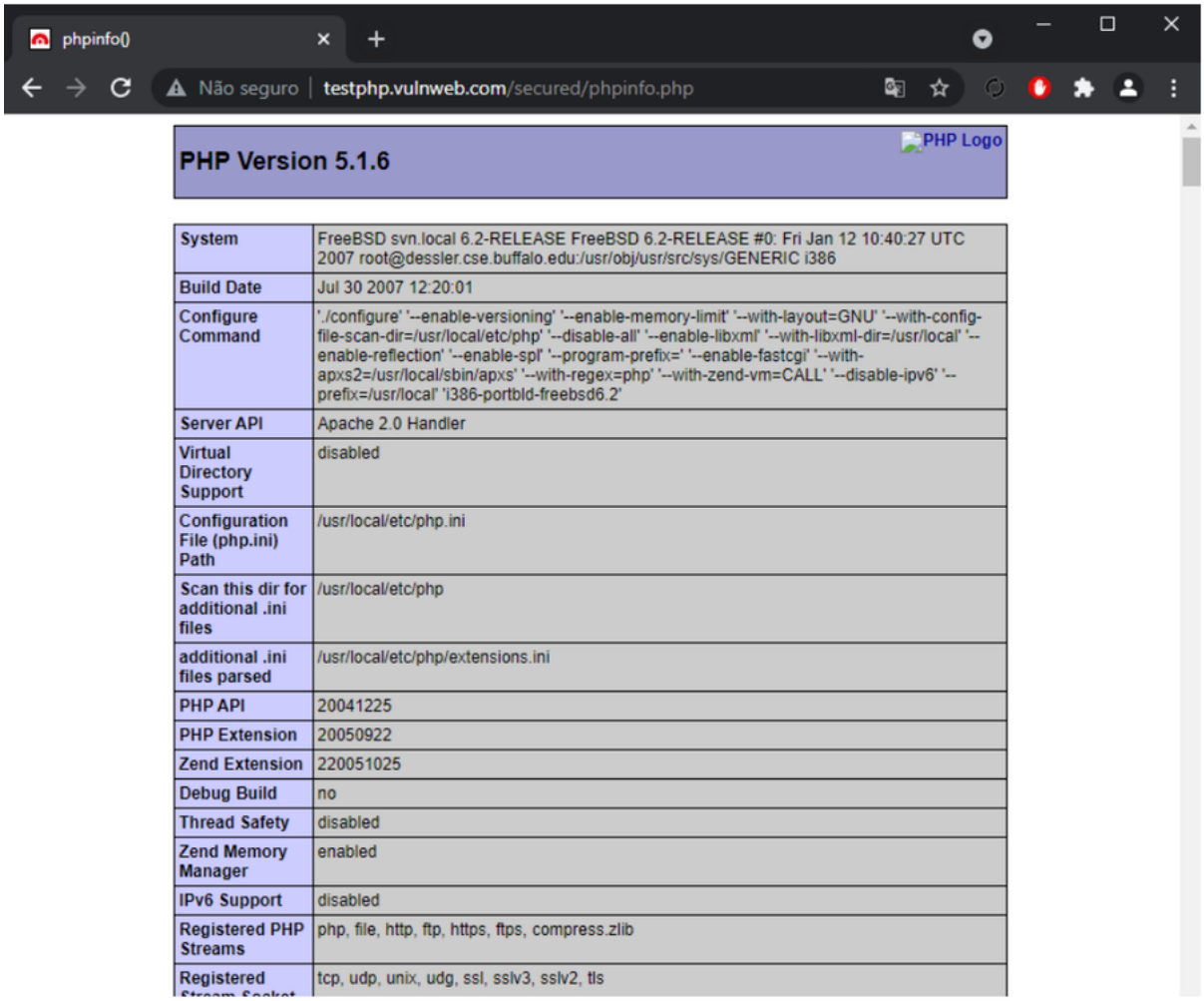
### *6.3.1 Demonstração*

Utilizando o mesmo ambiente do laboratório anterior, dessa vez foi selecionado uma ferramenta para fazer um ataque de força bruta em busca de arquivos e diretórios padrões na URL da aplicação, com o intuito de encontrar arquivos sensíveis.

A ferramenta utilizada chama-se 'ffuf', ela automatiza o processo de buscas recursivas por nomes de arquivos a partir de uma lista de palavras definida. Foi utilizada uma lista comum para testes como esses, disponível publicamente. Abaixo segue o comando utilizado para executar o processo, sendo "common.txt" a lista que contém nomes de arquivos e diretórios padrões que serão testados.

```
Comando: $ ffuf -w common.txt -u http://testphp.vulnweb.com/FUZZ --recursion
```

**Figura 4 – Laboratório Exposição de dados sensíveis**




| | |
|---|---|
| System | FreeBSD svn.local 6.2-RELEASE FreeBSD 6.2-RELEASE #0: Fri Jan 12 10:40:27 UTC 2007 root@dessler.cse.buffalo.edu:/usr/obj/usr/src/sys/GENERIC i386 |
| Build Date | Jul 30 2007 12:20:01 |
| Configure Command | './configure' '--enable-versioning' '--enable-memory-limit' '--with-layout=GNU' '--with-config-file-scan-dir=/usr/local/etc/php' '--disable-all' '--enable-libxml' '--with-libxml-dir=/usr/local' '--enable-reflection' '--enable-spl' '--program-prefix=' '--enable-fastcgi' '--with-apxs2=/usr/local/sbin/apxs' '--with-regex=php' '--with-zend-vm=CALL' '--disable-ipv6' '--prefix=/usr/local' 'i386-portbld-freebsd6.2' |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | disabled |
| Configuration File (php.ini) Path | /usr/local/etc/php.ini |
| Scan this dir for additional .ini files | /usr/local/etc/php |
| additional .ini files parsed | /usr/local/etc/php/extensions.ini |
| PHP API | 20041225 |
| PHP Extension | 20050922 |
| Zend Extension | 220051025 |
| Debug Build | no |
| Thread Safety | disabled |
| Zend Memory Manager | enabled |
| IPv6 Support | disabled |
| Registered PHP Streams | php, file, http, ftp, https, ftps, compress.zlib |
| Registered | tcp, udp, unix, udg, ssl, sslv3, sslv2, tls |

Fonte: Autor

Ao fim do processo, o arquivo no caminho */secured/phpinfo.php* fora encontrado como mostrado na figura 4, trata-se de uma página executando a função *phpinfo();* da linguagem PHP. Este é um arquivo muito sensível para qualquer aplicação PHP, visto que exibe informações sobre o ambiente, podendo citar como exemplo os seguintes itens:

- A versão atual do PHP que o site está rodando.
- A informação e o ambiente do servidor.
- O ambiente PHP.
- Informações sobre a versão do sistema operacional (SO).
- Caminhos, incluindo a localização do php.ini.
- Valores padrões e locais para as opções de configuração do PHP.
- Cabeçalhos HTTP.

- Licença PHP.
- Módulos e extensões atualmente em uso.

Todos esses dados constituem informações sensíveis, que para um atacante seria de grande auxílio para direcionar outros possíveis ataques ao ambiente já conhecido.

### *6.3.2 Mitigação*

Revisar todos os arquivos que são disponibilizados publicamente no diretório da aplicação, de modo a evitar possíveis vazamentos acidentais de conteúdos sensíveis, que possam prejudicar a infraestrutura do sistema.

## 6.4 A4 - XXE - Entidade Externa do XML

A injeção da entidade externa do XML (XXE) é uma vulnerabilidade que permite que um invasor interfira no processamento de dados XML de um aplicativo. Frequentemente, permite que um invasor visualize arquivos no servidor da aplicação e interaja com qualquer sistema *back-end* ou externo que o próprio aplicativo possa acessar.

Em algumas situações, um invasor pode escalar um ataque XXE para comprometer o servidor subjacente ou outra infraestrutura de *back-end*, aproveitando a vulnerabilidade XXE para executar outros ataques como falsificação de solicitação do lado do servidor (SSRF).

### *6.4.1 Como surgem as vulnerabilidades XXE?*

Alguns aplicativos utilizam o formato XML para transmitir dados entre o navegador e o servidor. Aplicativos que fazem isso quase sempre utilizam uma biblioteca padrão ou API de plataforma para processar os dados XML no servidor. As vulnerabilidades XXE surgem porque o XML contém vários recursos potencialmente perigosos e as plataformas padrões oferecem suporte a esses recursos, mesmo se eles não forem normalmente usados pelo aplicativo.

Entidades externas XML são um tipo de entidade XML customizada, cujos valores definidos são carregados fora do DTD (Document type definition) em que são declarados. As entidades externas são particularmente interessantes do ponto de vista da segurança, pois permitem que uma entidade seja definida com base no conteúdo de um caminho de arquivo ou URL.

### *6.4.2 Quais são os tipos de ataques XXE*

- **Para ler arquivos do servidor**
  - Quando uma entidade externa é definida contendo o caminho de um arquivo interno e o conteúdo desse arquivo é retornado na resposta da requisição.

- **Para realizar ataques SSRF (Server-side request forgery)**
  - Quando uma entidade externa é definida com base em uma (URL) para um sistema *back-end*.
- **Para recuperar dados através de mensagens de erro**
  - Quando o invasor pode acionar uma mensagem de erro que exponha dados confidenciais.

### *6.4.3 Explorando XXE para ler arquivos internos*

Para realizar um ataque de XXE que lê um arquivo do sistema de arquivos do servidor, é necessário modificar o XML enviado de duas maneiras:

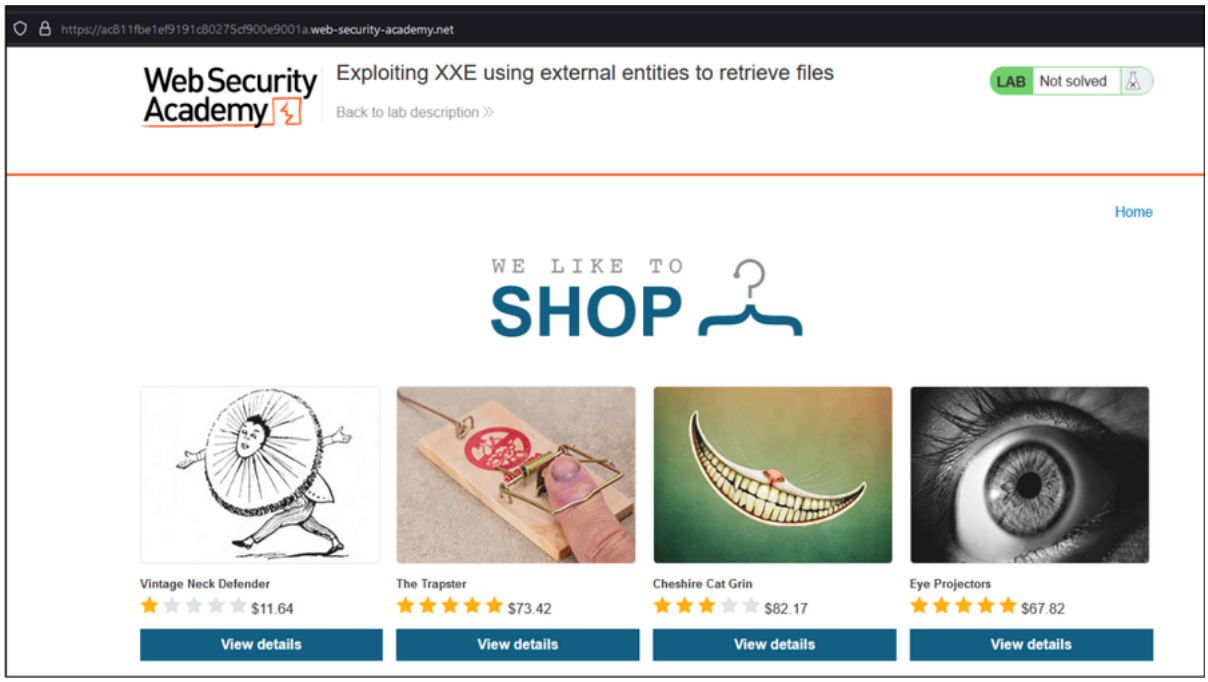
- Apresente (ou edite) um elemento DOCTYPE que define uma entidade externa que contém o caminho para o arquivo.
- Edite um valor de dados no XML que é retornado na resposta do aplicativo, para fazer uso da entidade externa definida.

### *6.4.4 Demonstração*

Nesse exemplo foi utilizado o laboratório de XXE da *PortSwigger*, basicamente consiste em um site de uma loja que faz consultas de dados com a utilização de XML.

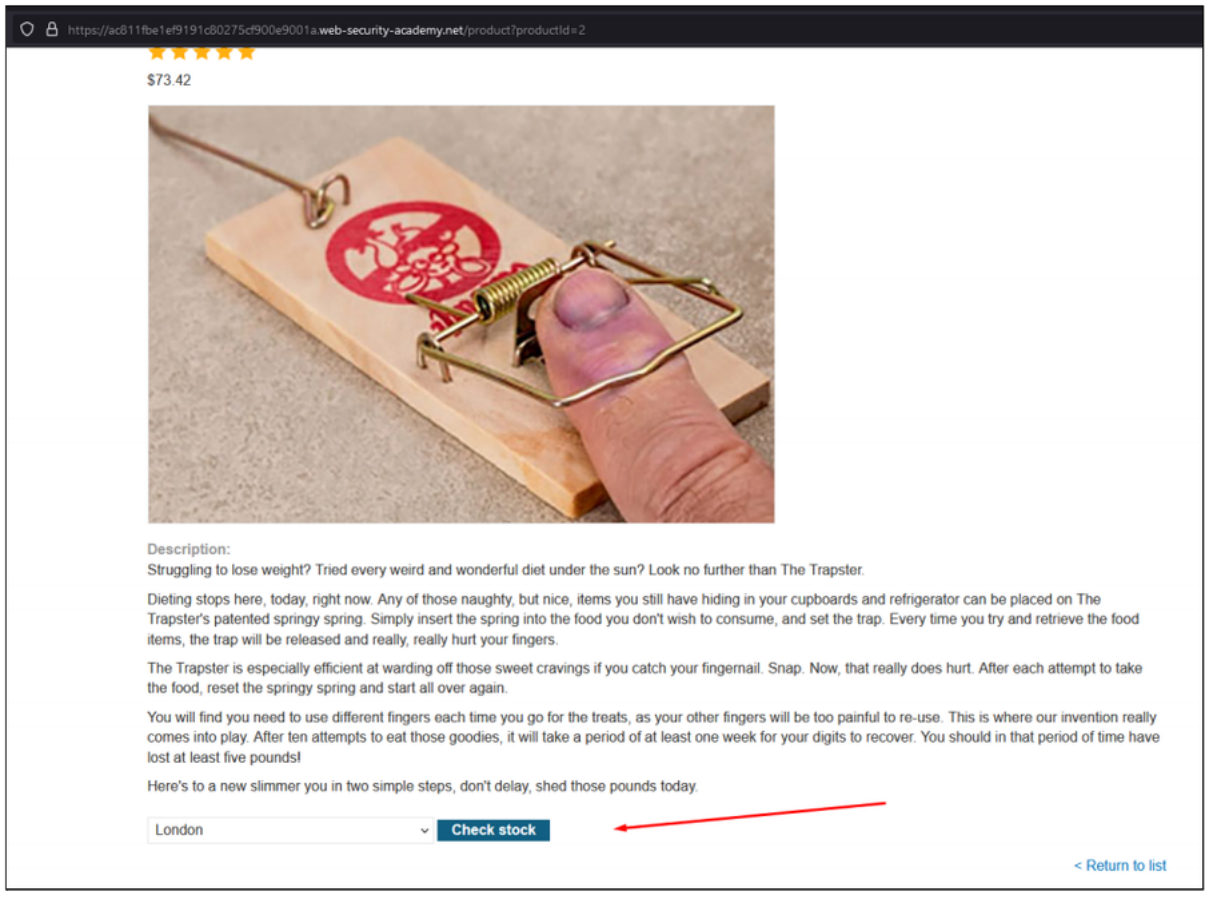
Um atacante então com o auxílio de um *proxy* entre a aplicação e a sua máquina, consegue alterar a requisição de checagem de estoque de um produto, para o caminho interno de um arquivo. E, na resposta da requisição o conteúdo do arquivo é carregado, permitindo ao atacante visualizá-lo. Abaixo (figuras 5 a 11) temos o passo-a-passo da exploração visando ler o arquivo interno '/etc/passwd' do servidor.

**Figura 5 – Laboratório XXE**




Fonte: Autor

**Figura 6 – Checagem de estoque de um produto**

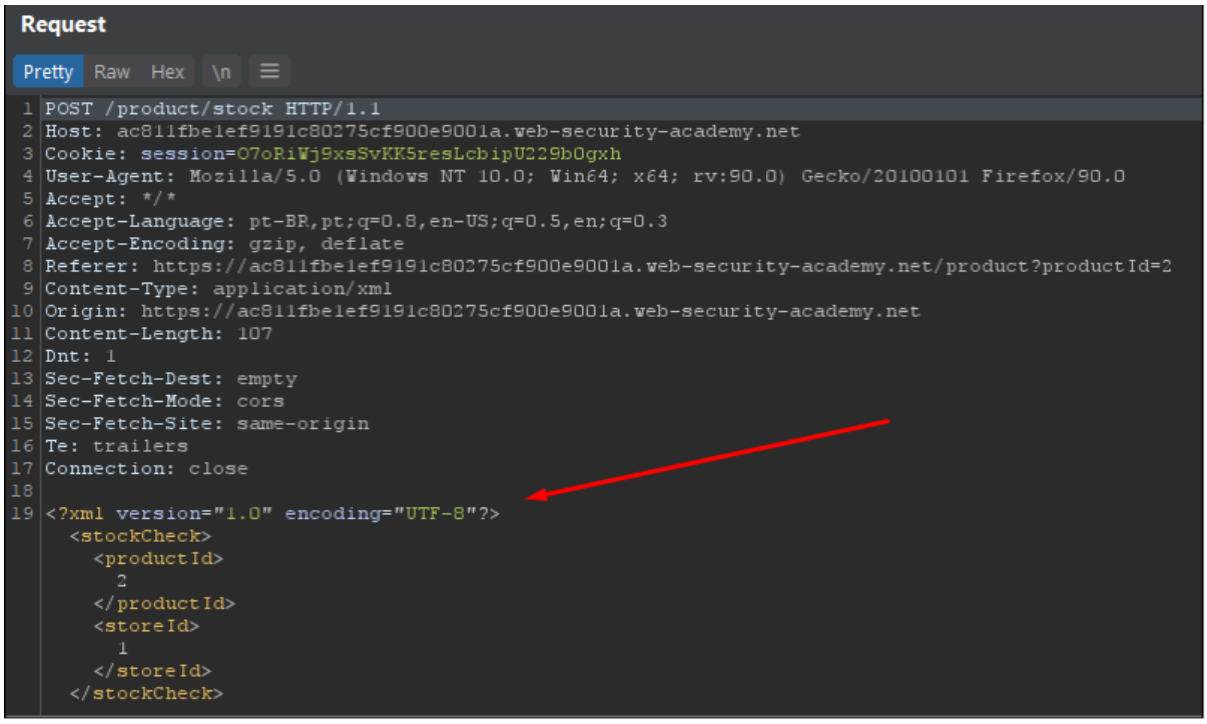



Fonte: Autor

**Figura 7 – Resposta da checagem**

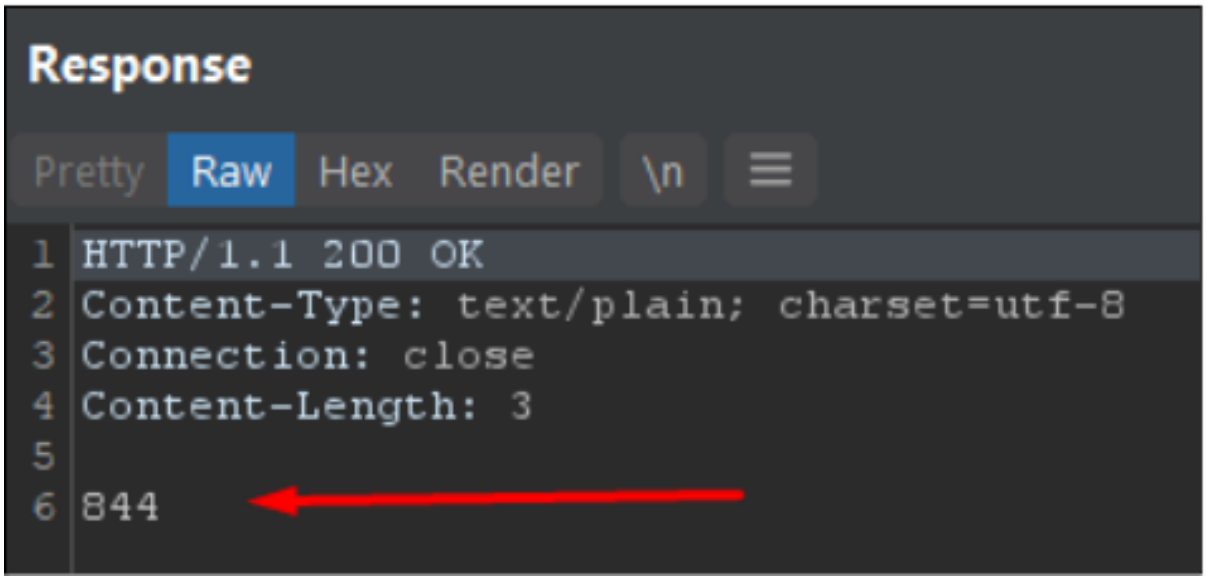

Fonte: Autor

**Figura 8 – Requisição da checagem interceptada**



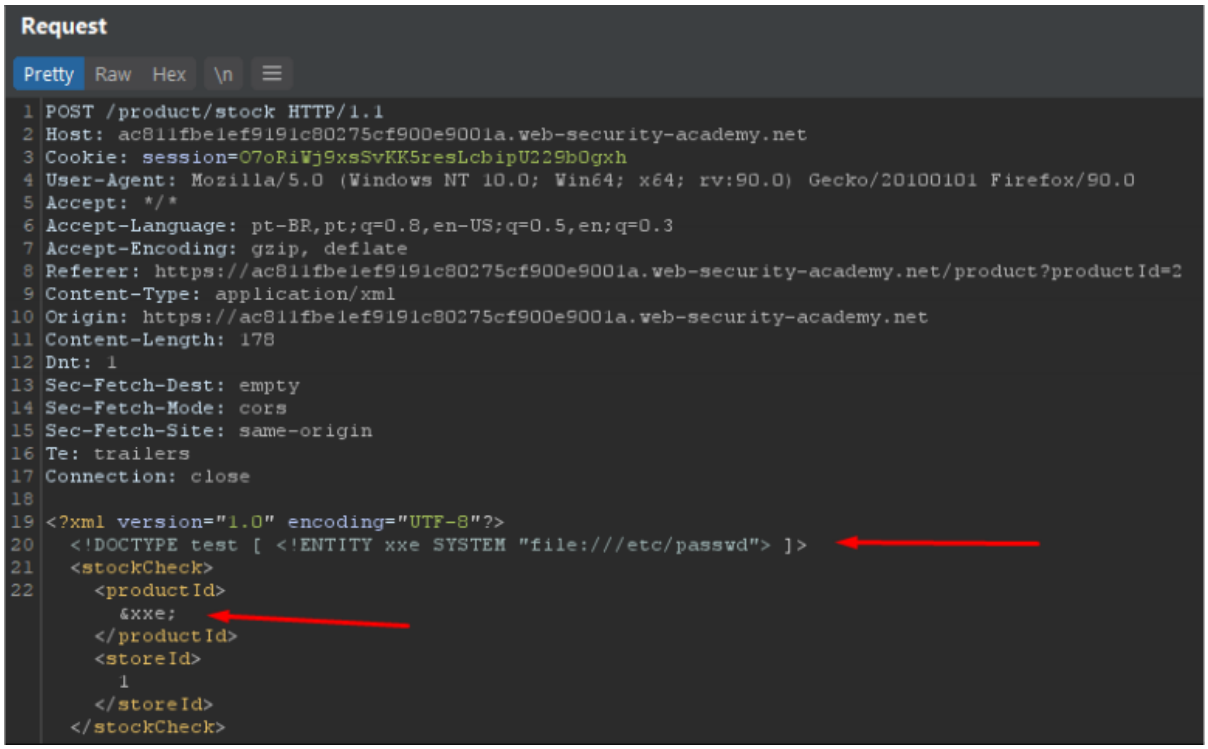

Fonte: Autor

**Figura 9 – Resposta da requisição**



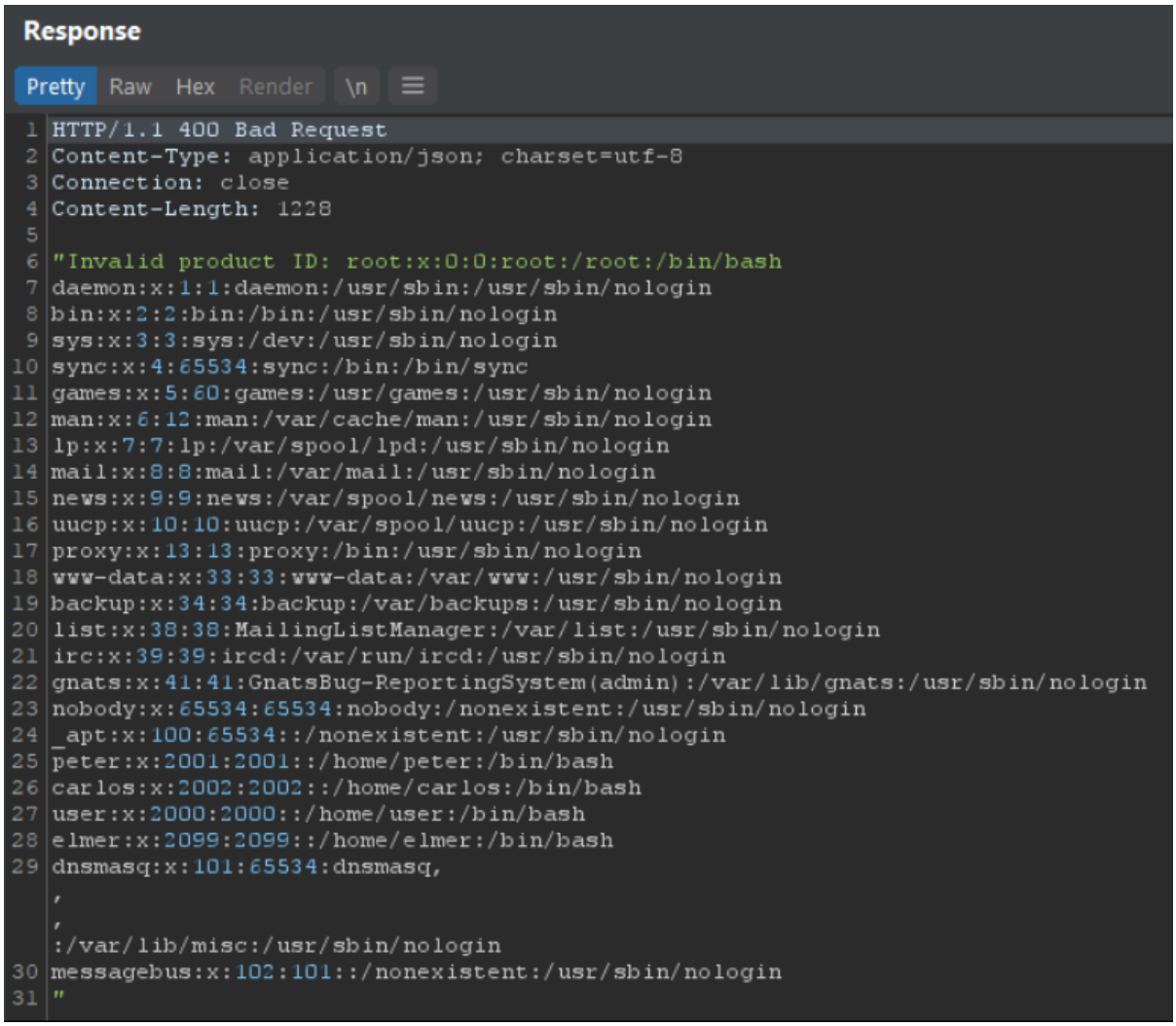

Fonte: Autor

**Figura 10 – Payload de exploração XXE inserido**




Fonte: Autor

**Figura 11 – Resultado da execução do payload XXE**




Fonte: Autor

### *6.4.5 Mitigação*

Praticamente todas as vulnerabilidades XXE surgem pois a biblioteca de análise XML da aplicação, oferece suporte a recursos XML potencialmente perigosos, que o aplicativo não precisa ou não pretende usar. A maneira mais fácil e eficaz de prevenir ataques XXE é desabilitar esses recursos.

Geralmente, é suficiente desabilitar a resolução de entidades externas e desabilitar o suporte para *XInclude*. Isso geralmente pode ser feito através de opções de configuração ou substituindo no código o comportamento padrão. É importante consultar a documentação da biblioteca de análise de XML ou API utilizada para obter detalhes sobre como desabilitar recursos desnecessários.

## 6.5 A5 - Quebra de Controle de Acesso

Os sites possuem páginas protegidas de visitantes regulares, por exemplo, apenas o usuário do administrador deve ser capaz de acessar uma página para gerenciar outros usuários. Se um visitante puder acessar páginas protegidas ao qual ele não está autorizado para visualizar, tem-se uma quebra de controle de acesso.

Um visitante regular que consegue acessar páginas protegidas, pode causas alguns problemas de segurança:

- Ser capaz de visualizar informações sensíveis
- Acessar funcionalidades não autorizadas

Se um usuário não autenticado puder acessar qualquer página, é uma falha. Se um usuário não administrativo pode acessar a página administrativa, também é uma falha.

A quebra de controle de acesso possibilita que invasores ignorem o controle de autorização, permitindo visualizar dados confidenciais ou executar tarefas como se fossem um usuário privilegiado.

### *6.5.1 Exemplo*

IDOR (*Insecure Direct Object Reference*), é o ato de explorar uma configuração incorreta na forma como a entrada do usuário é tratada, para acessar recursos que um usuário comum normalmente não conseguiria acessar. IDOR é um tipo de vulnerabilidade de controle de acesso.

Por exemplo, digamos que um usuário esteja fazendo login em sua conta bancária e, depois de autenticar corretamente, a aplicação redireciona a um endereço como este: '*https://example.com/banco?numero_conta=1234'*. Nessa página, pode-se ver todos os dados bancários da conta logada.

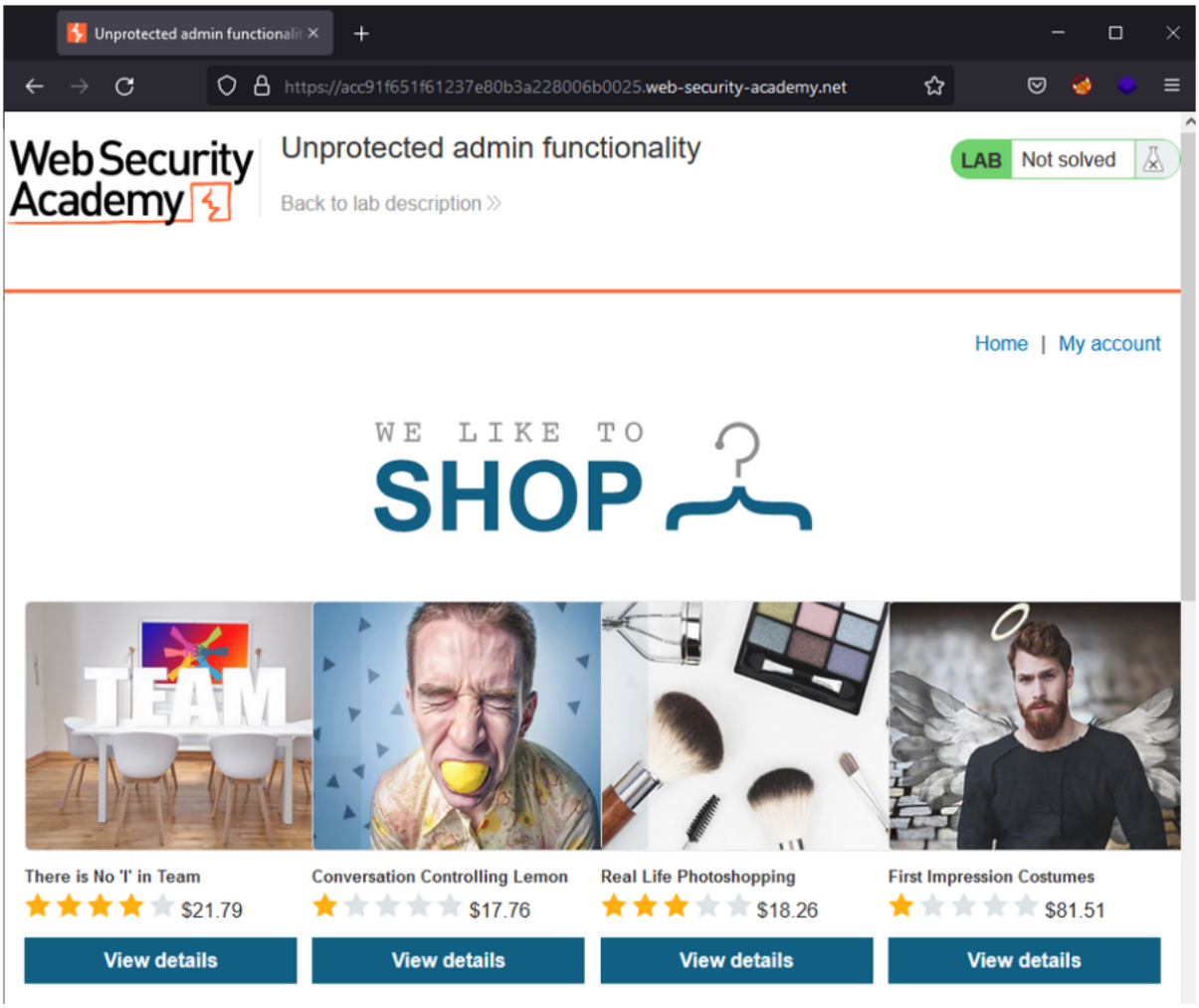
No entanto, pode haver um problema potencialmente enorme, um invasor pode tentar alterar o valor do parâmetro 'numero_conta' para algo como '12345' e, se o site estiver configurado incorretamente, ele teria acesso às informações bancárias de outro usuário ao qual a conta '12345' pertence.

### *6.5.2 Demonstração*

Nesse exemplo foi explorada uma falha de quebra de controle de acesso, através da exposição da página administrativa da aplicação, por meio do arquivo *robots.txt*, que tem como função indicar aos mecanismos de buscas quais páginas podem ou não podem ser indexadas em suas pesquisas. Nesse caso o administrador desabilitou a indexação da página */administrator-panel/,* que é sua página de gerenciamento do sistema.

Um usuário comum não encontraria essa página, mas um atacante ao saber da possibilidade da presença do arquivo *robots.txt*, sempre vai checar sua existência. E ao acessar a pagina administrativa encontrada no arquivo robots.txt (figura 13), o atacante percebe que não possui nenhum outro mecanismo de proteção para o sistema de gerenciamento do site. E consegue acessar a funcionalidade de administrador, tendo a possibilidade de criar e deletar contas de outros usuários cadastrados como demonstrado na figura 14. Abaixo segue o passo-a-passo para demonstração desse ataque.

**Figura 12 – Laboratório Quebra de Controle de Acesso**

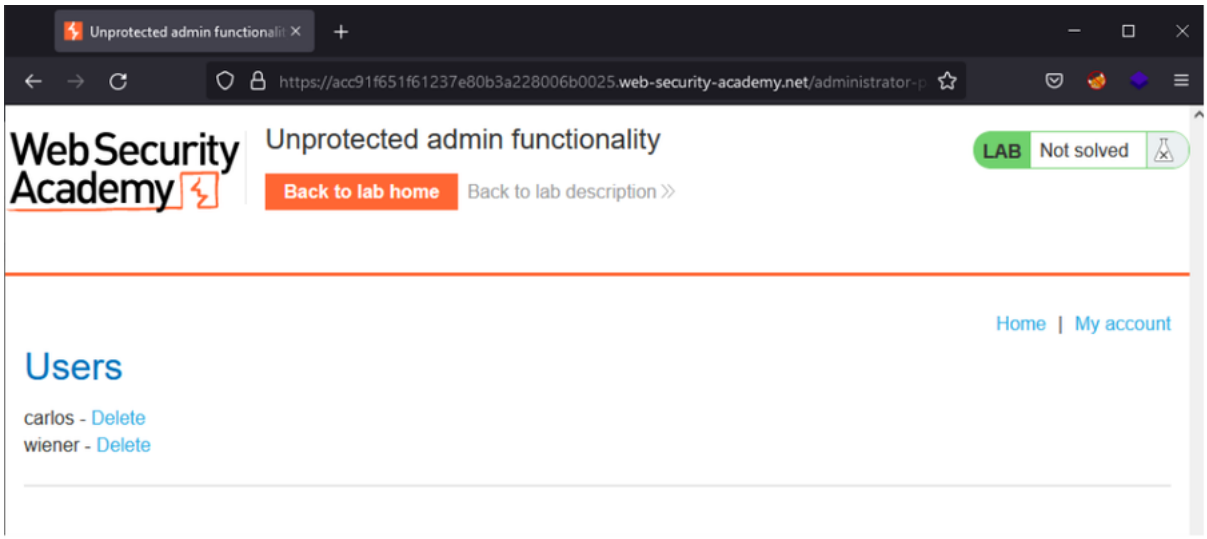



Fonte: Autor

**Figura 13 – Checando arquivo robots.txt**


Fonte: Autor

**Figura 14 – URL administrativa acessada**




Fonte: Autor

### *6.5.3 Mitigação*

As vulnerabilidades de quebra de controle de acesso geralmente podem ser evitadas adotando uma abordagem de defesa em profundidade e aplicando os seguintes princípios:

- Nunca confiar apenas na ofuscação para controle de acesso.
- A menos que um recurso deva ser acessível publicamente, o acesso deve ser negado por padrão.
- Sempre que possível, utilizar um único mecanismo em todo o aplicativo para impor controles de acesso.
- No nível do código, tornar obrigatório para os desenvolvedores declarar o acesso permitido para cada recurso e negar o acesso por padrão.
- Efetuar uma auditoria completa onde os controles de acesso sejam testados, para garantir que estejam funcionando conforme projetados.

## 6.6 A6 - Configurações de Segurança Incorretas

Este tópico é diferente das outros, porque ocorre quando a segurança poderia ter sido configurada corretamente, mas não foi.

As configurações incorretas de segurança incluem:

- Permissões mal configuradas em serviços em nuvem, como *Buckets S3*.
- Manter recursos desnecessários ativados, como serviços, páginas, contas ou privilégios.
- Contas padrão com senhas inalteradas.
- Mensagens de erro que são excessivamente detalhadas e permitem que um invasor descubra mais sobre o sistema.
- Não usar cabeçalhos de segurança ou revelar muitos detalhes do servidor nas respostas de solicitações HTTP.

Essa vulnerabilidade pode frequentemente levar a mais vulnerabilidades, como credenciais padrão que fornecem acesso à dados confidenciais, XXE ou injeção de comando em páginas administrativas.

### *6.6.1 Demonstração - Senhas padrão*

Este é um exemplo específico de configuração incorreta de segurança. Deve-se alterar qualquer senha padrão, mas as pessoas geralmente não o fazem. É muito comum em dispositivos embarcados e de IoT (Internet of things), na maioria das vezes os proprietários não alteram essas senhas.

É fácil imaginar o risco de credenciais padrões do ponto de vista de um invasor. Ser capaz de obter acesso à painéis de administração, serviços projetados para administradores de sistema e fabricantes, ou até mesmo a infraestrutura de rede pode ser muito útil para atacar uma empresa. Da exposição de dados a injeção de comandos, os efeitos de credenciais padrões podem ser graves.

## 6.7 A7 - XSS - Cross Site Scripting

*Cross-site scripting*, também conhecido como XSS é um tipo de ataque que pode permitir que um invasor execute *scripts* maliciosos, e faça com que também sejam executados na máquina da vítima.

Uma aplicação *web* é vulnerável a XSS se não sanitizar entradas de usuários. XSS é possível em *Javascript, VBScript, Flash e* CSS. Existem dois tipos principais de ataques XSS:

- XSS Armazenado

  - O tipo mais perigoso de XSS. Quando um *payload* malicioso é executado tendo como origem o banco de dados do site. Isso geralmente acontece quando um site permite que uma entrada não sanitizada de usuário seja armazenada no banco de dados. Atacantes aproveitam disso para armazenar *scripts* maliciosos no sistema, afetando outros usuários que requisitem o conteúdo onde o *script* foi inserido, como páginas de comentários.

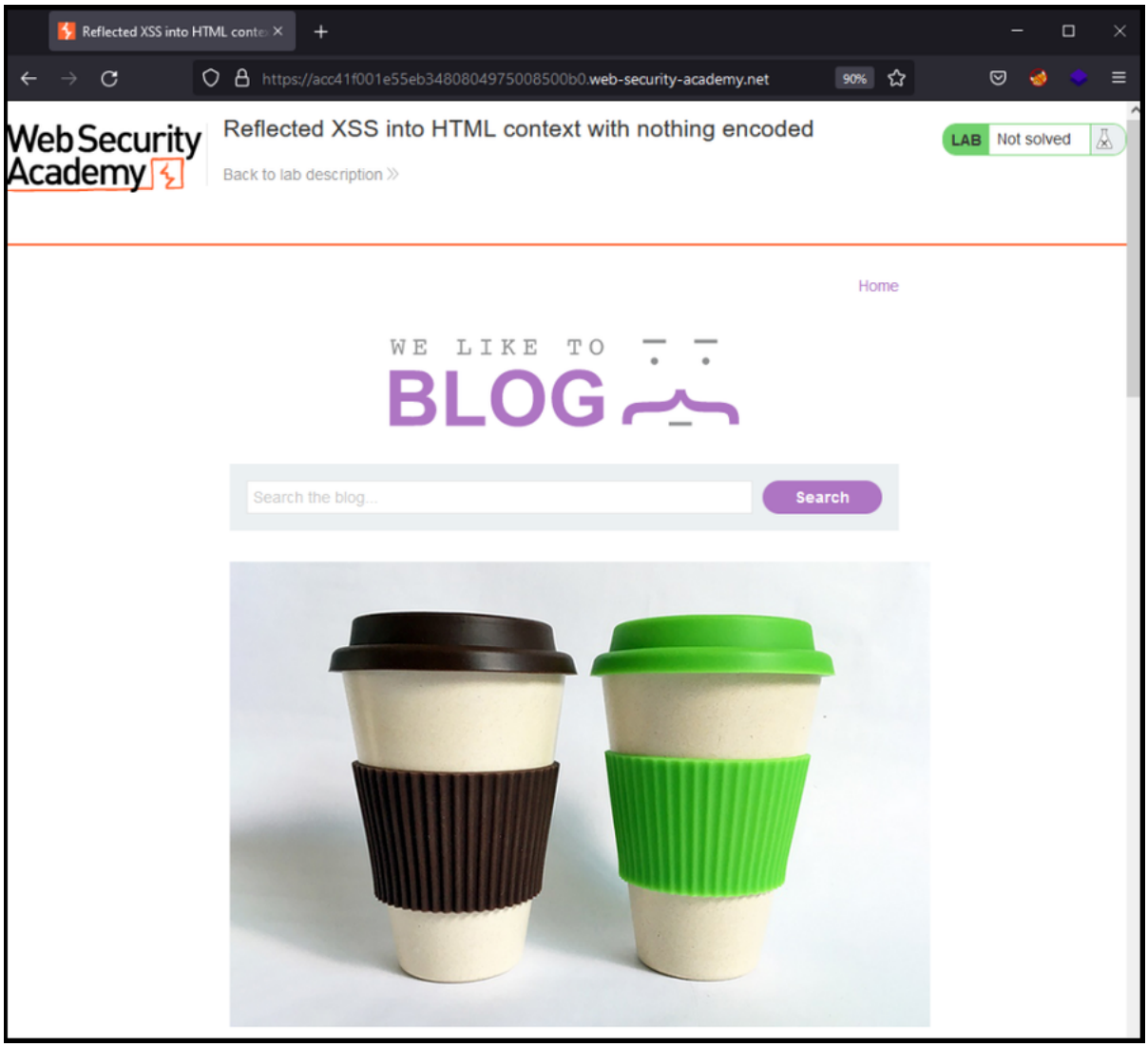
- XSS Refletido

  - O *payload* faz parte da URL de solicitação das vítimas ao site. O site então inclui esse *payload* na resposta ao usuário. Ou seja, um invasor precisa enganar a vítima para que ela clique em uma URL modificada contendo o *script* malicioso, para que seja executado.

### *6.7.1 Demonstração*

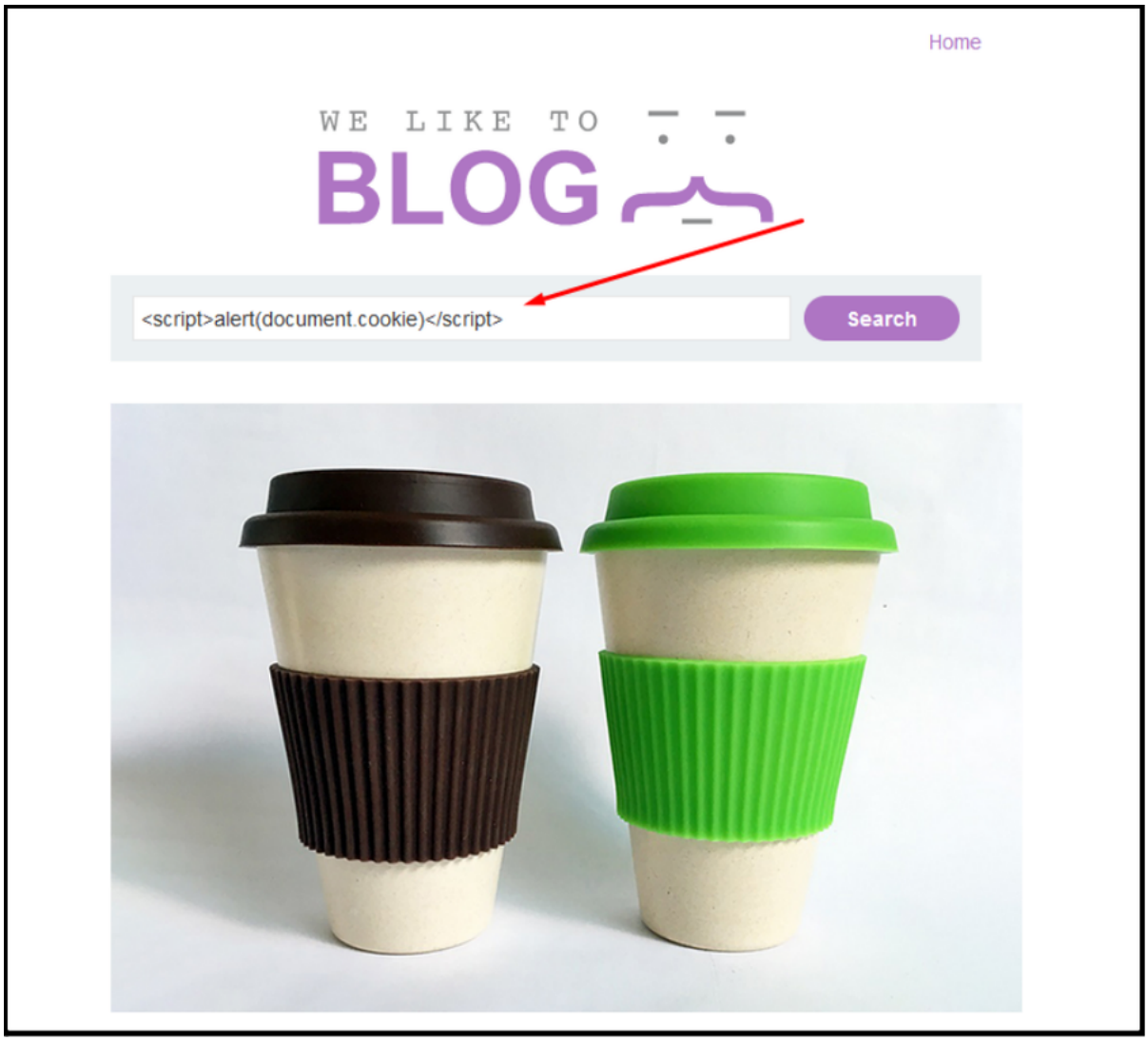
Nesse exemplo tem-se um site com um campo de buscas (15), que permite entrada de usuários, como o campo não sanitiza os dados passados, um atacante pode inserir *payloads* XSS que são enviados através de parâmetros GET na URL, dessa forma tem-se um XSS Refletido, onde o atacante precisa informar a URL modificada para que o *script* seja executado no alvo escolhido. Na demonstração foi utilizado um *script* para exibir o valor atual do *cookie* do usuário do site como visto na figura 17. Porém, existem diversos outros tipos de *payloads* como de redirecionamento e de escrita de HTML.

**Figura 15 – Laboratório XSS**



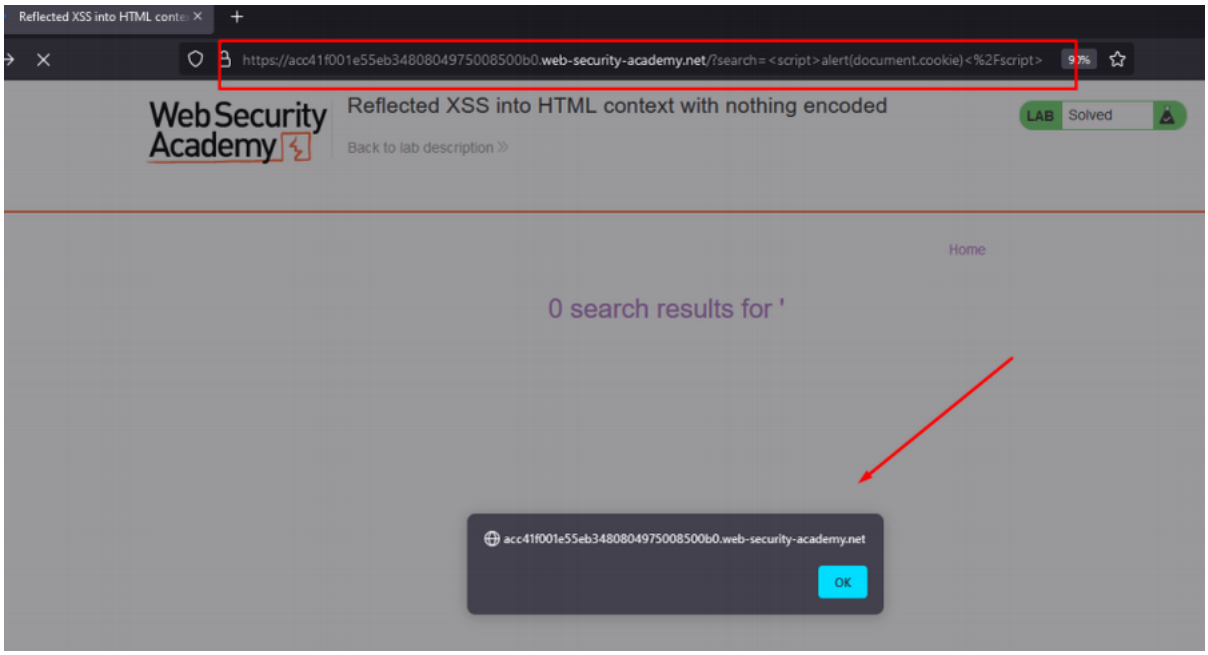

Fonte: Autor

**Figura 16 – Payload XSS inserido no campo de busca**




Fonte: Autor

**Figura 17 – Script executado, exibe o cookie do usuário**




Fonte: Autor

Exemplo de payload de redirecionamento: '*https://acc41f001e55eb348080497500850 0b0.web-security-academy.net/?search=<**script**>**document.location="http://ufc.br"**</**script**>*'

### *6.7.2 Mitigação*

A prevenção de XSS é trivial em alguns casos, mas pode ser muito mais difícil dependendo da complexidade da aplicação e da maneira como a entrada do usuário é tratada.

Em geral, a prevenção eficaz de vulnerabilidades XSS pode envolver uma combinação das seguintes medidas:

- Filtros de entrada na chegada. No ponto em que a entrada do usuário é recebida, deve-se filtrar o mais estritamente possível com base em entradas permitidas.
- Codificar os dados na saída. No ponto em que os dados controláveis pelo usuário são produzidos nas respostas HTTP, codificar a saída para evitar que seja interpretada como conteúdo ativo. Dependendo do contexto de saída, isso pode exigir a aplicação de combinações de codificação HTML, URL, JavaScript e CSS.
- Utilização de cabeçalhos de resposta apropriados. Para evitar XSS em respostas HTTP que não pretendem conter nenhum HTML ou *JavaScript*, pode-se utilizar cabeçalhos *Content-Type* e *X-Content-Type-Options* para garantir que os navegadores interpretem as respostas da maneira desejada.

## 6.8 A8 - Desserialização Insegura

"A desserialização insegura é uma vulnerabilidade que ocorre quando dados não confiáveis são usados para abusar da lógica de um aplicativo" (ACUNETIX, 2017)

Essa definição ainda é bastante ampla, para dizer o mínimo. Simplesmente, a desserialização insegura está substituindo os dados processados de um aplicativo por um código malicioso, permitindo qualquer coisa, desde DoS (Negação de Serviço) a RCE (Remote Command Execution), que o invasor pode usar para se firmar em um cenário de ataque.

Especificamente, esse código malicioso aproveita o processo legítimo de serialização e desserialização utilizado por aplicações *web*.

A OWASP classifica esta vulnerabilidade como 8 em 10 devido aos seguintes motivos:

- Baixa explorabilidade. Essa vulnerabilidade geralmente ocorre caso a caso - não existe uma ferramenta/estrutura confiável para ela. Por causa de sua natureza, os invasores precisam ter uma boa compreensão do funcionamento interno da aplicação.
- A exploração é tão perigosa quanto a habilidade do invasor permite, ou a criticidade do que pode ser afetado. Por exemplo, alguém que só pode causar um DoS tornará o aplicativo indisponível. O impacto disso nos negócios varia na infraestrutura. Algumas organizações se recuperam bem, outras não.

### *6.8.1 Objetos*

Um elemento recorrente da programação orientada a objetos (POO), os objetos são compostos de duas coisas:

- Estado
- Comportamento

Os objetos permitem a criação de linhas de código semelhantes sem ter que fazer o trabalho de escrever as mesmas linhas de código novamente.

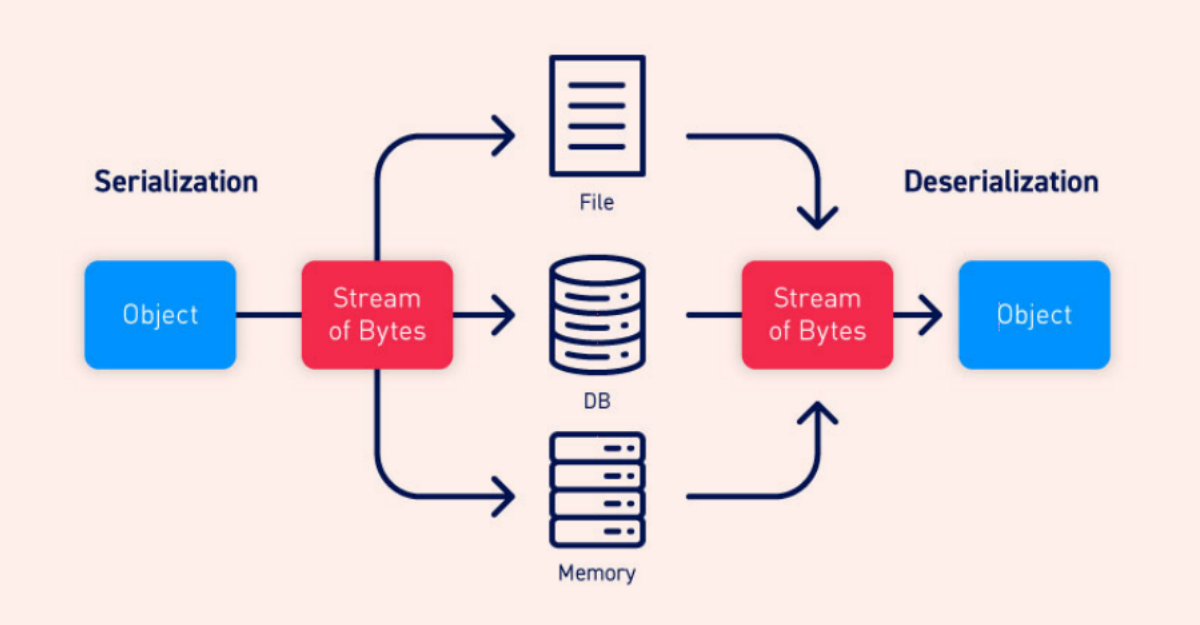
Por exemplo, uma lâmpada seria um bom objeto. As lâmpadas podem ter diferentes tipos de lâmpadas, este seria o seu estado, além de estarem ligadas/desligadas, seu comportamento.

Em vez de ter que acomodar todos os tipos de lâmpadas e se essa lâmpada específica está ligada ou desligada, é possível utilizar métodos para simplesmente alterar o estado e o comportamento da lâmpada.

Serialização é o processo de conversão de objetos usados na programação em uma formatação mais simples e compatível, para transmissão entre sistemas ou redes para posterior processamento, ou armazenamento como demonstrado na figura 18.

Alternativamente, a desserialização é o reverso disso, converter informações serializadas para sua forma legível por aplicativos.

**Figura 18 – Serialização e desserialização**

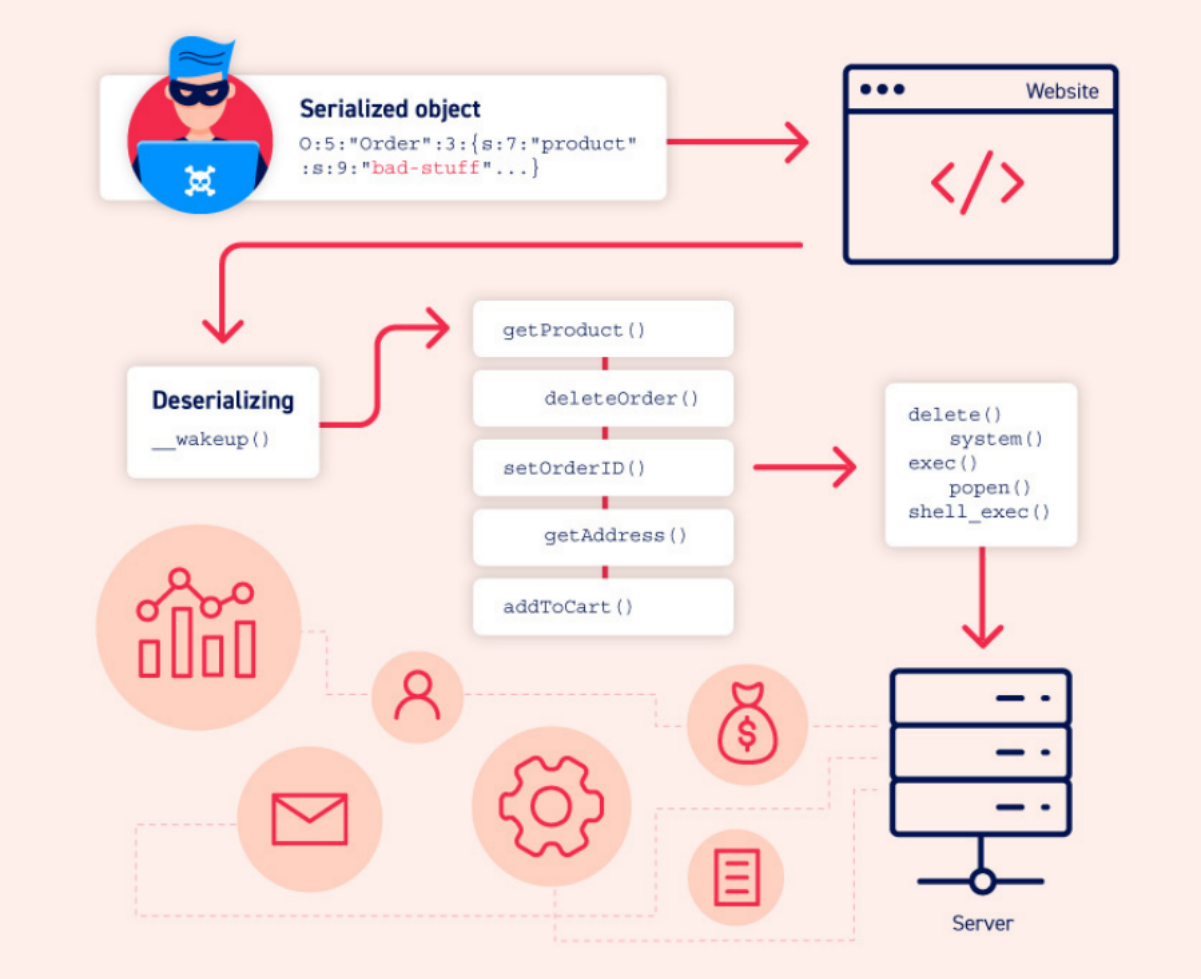



Fonte: PortSwigger

Como exemplo, pode-se citar que existe uma senha “senha123” de um programa que precisa ser armazenado em um banco de dados em outro sistema. Para viajar através de uma rede, esta *string* precisa ser convertida em binário. Uma vez que chega ao banco de dados, é convertida ou desserializada de volta em “senha123” para que possa ser armazenada.

A desserialização insegura ocorre quando os dados de uma parte não confiável (invasor) são executados porque não há filtragem ou sanitização da entrada, o sistema assume que os dados são confiáveis e os executa sem restrições.

**Figura 19 – Desserialização Insegura**

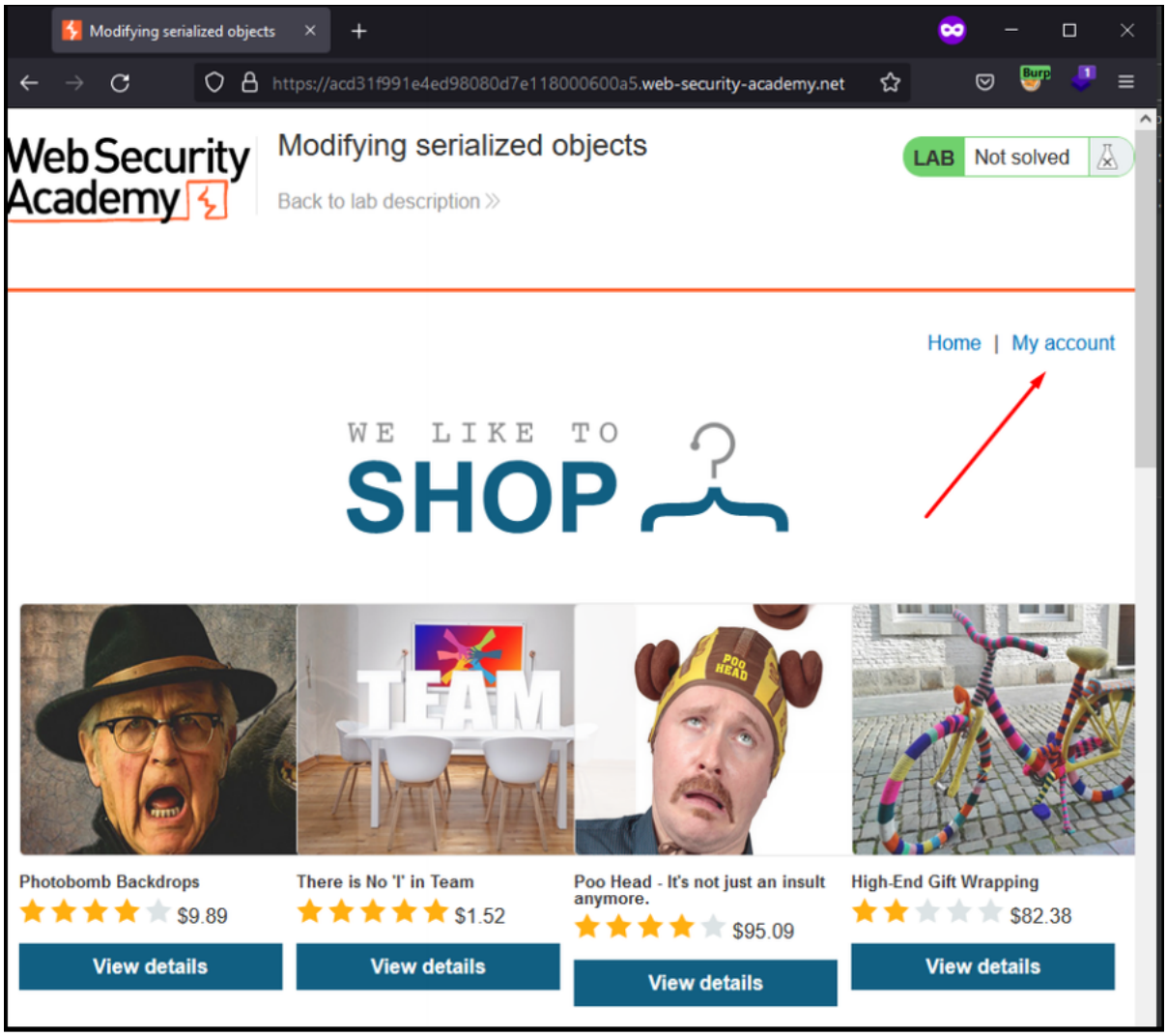



Fonte: PortSwigger

### *6.8.2 Demonstração*

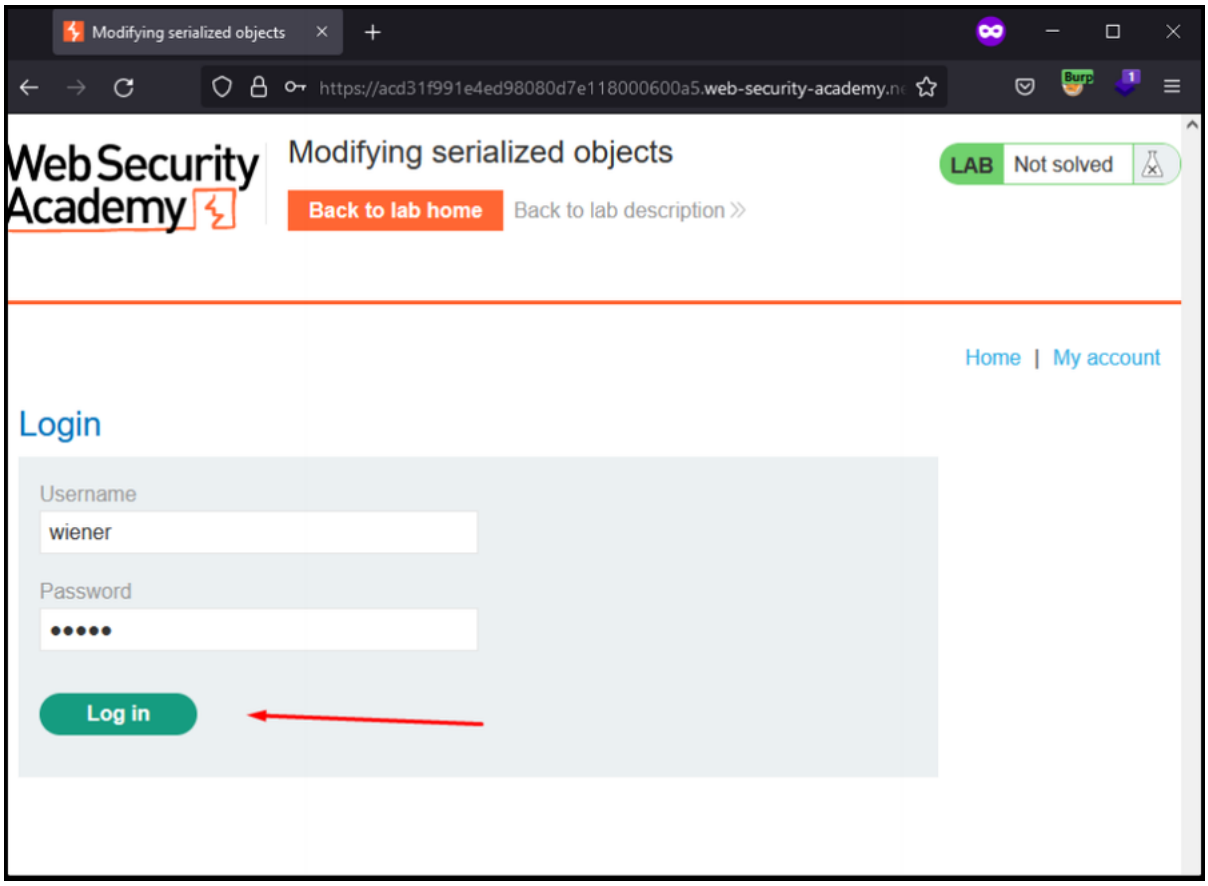
Nesse exemplo será demonstrado um caso de desserialização insegura através da manipulação de *cookies* de sessão.

**Figura 20 – Laboratório Desserialização Insegura**



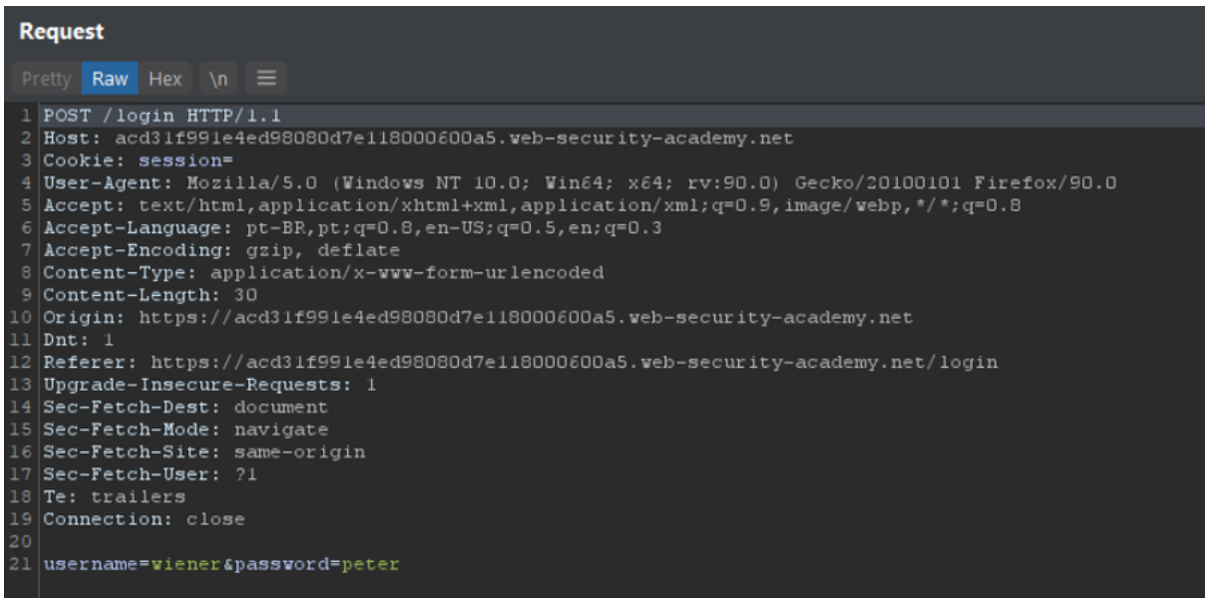

Fonte: Autor

**Figura 21 – Página de login**




Fonte: Autor

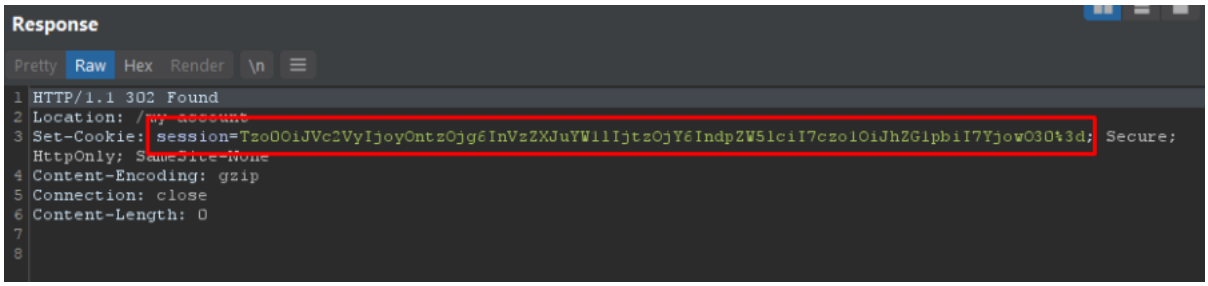
**Figura 22 – Requisição de Login**




Fonte: Autor

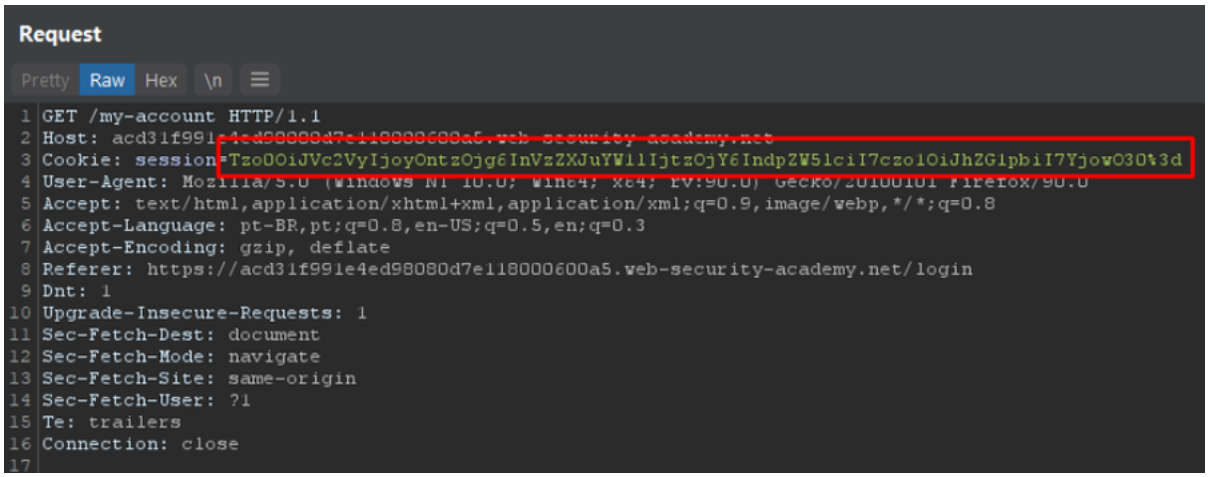
Após a requisição de login ser enviada (figura 22), a aplicação então retorna um *cookie* de sessão codificado (serializado) (figura 23).

**Figura 23 – Cookie recebido após login**




Fonte: Autor

**Figura 24 – Cookie é automaticamente reenviado em futuras solicitações**




Fonte: Autor

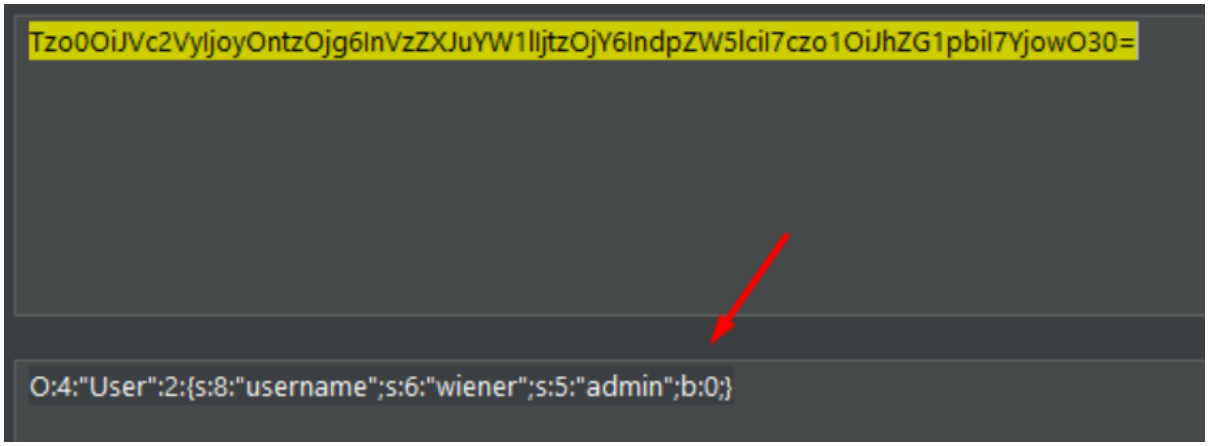
Após autenticar, é apresentada uma página simples em que apenas é possível alterar o endereço de e-mail do cadastro.

**Figura 25 – Pagina autenticada**


Fonte: Autor

O *cookie* recebido estava codificado em base64, e ao decodificá-lo é possível ler seus parâmetros, sendo que um deles é um denominado 'admin' que tem um valor b (booleano) igual a zero como visto na figura 26.

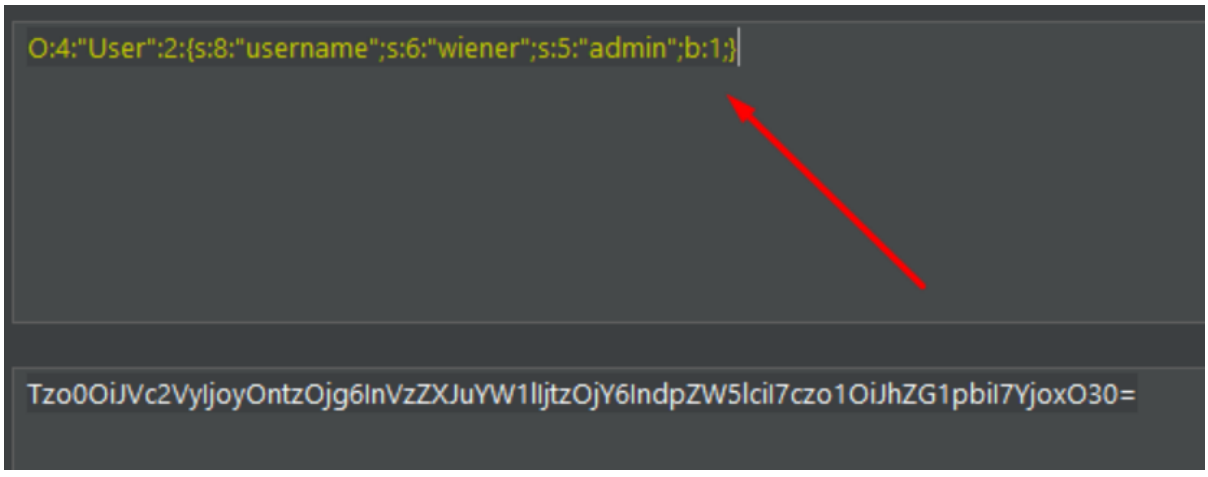
**Figura 26 – Cookie desserializado**




Fonte: Autor

Simulando um atacante, alteramos o valor *booleano* do parâmetro admin de 0 para 1 e codificamos novamente para base64 (figura 27).

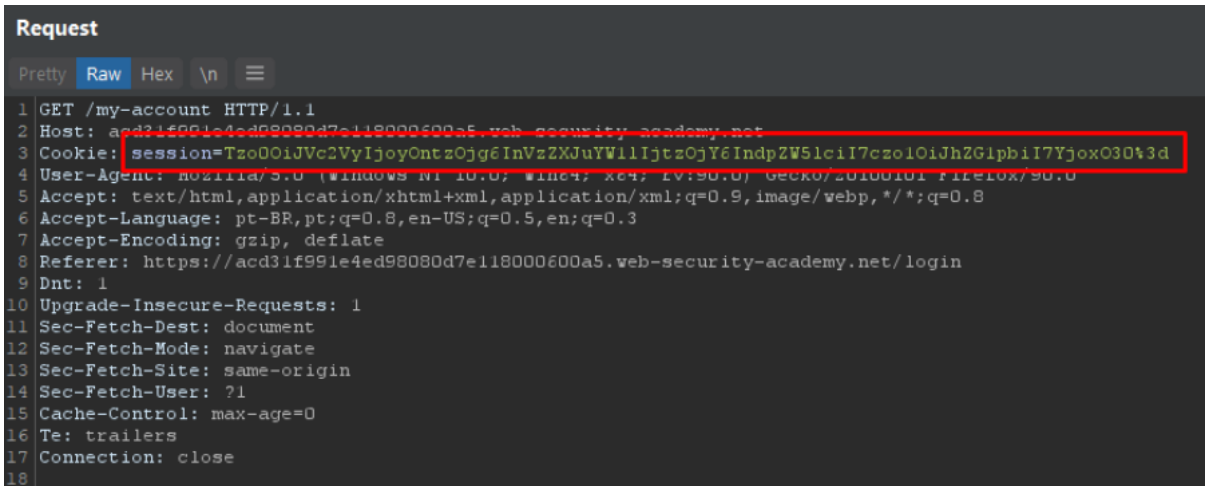
**Figura 27 – Alteração no cookie desserializado**




Fonte: Autor

Após codificar o *cookie* alterado, então o reenviamos na requisição referente ao painel autenticado.

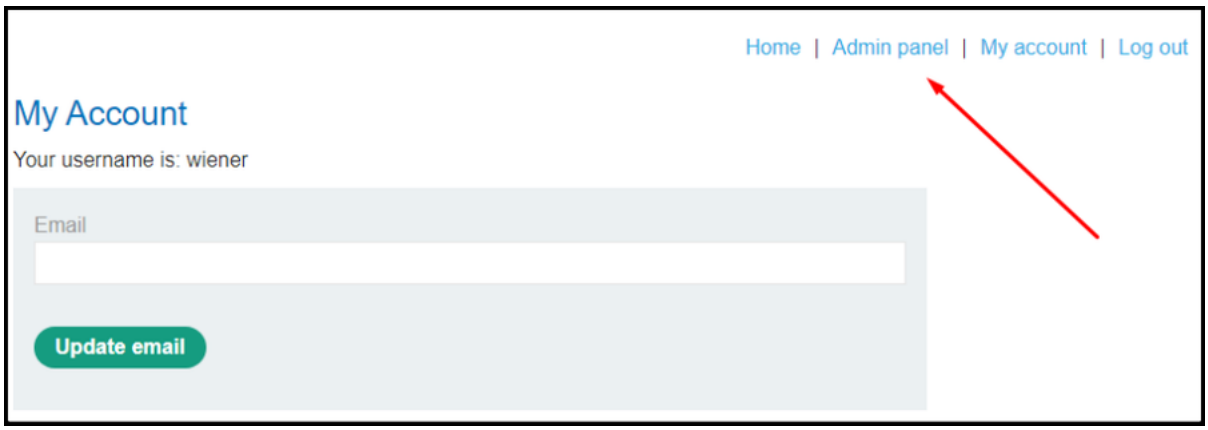
**Figura 28 – Cookie modificado reenviado**




Fonte: Autor

Ao carregar novamente a página com o *cookie* modificado, é possível notar que uma nova opção foi inserida ao menu “Admin Panel” como demonstra a figura 29. Isso ocorreu porque ao desserializarmos o *cookie* e alterar seu parâmetro admin de 0 para 1 e reenviar a aplicação, a mesma assumiu que essa sessão seria uma sessão administrativa e então liberou novas funcionalidades ao painel.

**Figura 29 – Funcionalidade administrativa disponível**




Fonte: Autor

Ao clicar na nova opção liberada, agora um atacante tem permissão de deletar outros usuários já existentes.

**Figura 30 – Funcionalidade administrativa acessada**


Fonte: Autor

### *6.8.3 Mitigação*

De modo geral, a desserialização da entrada do usuário deve ser evitada, a menos que seja absolutamente necessário. A alta gravidade das explorações que ele potencialmente permite e a dificuldade de proteção contra elas superam os benefícios em muitos casos.

Se for necessário desserializar dados de fontes não confiáveis, precisa-se incorporar medidas robustas para garantir que os dados não sejam adulterados.

Se possível, deve ser evitado o uso de recursos genéricos de desserialização. Os dados serializados desses métodos contêm todos os atributos do objeto original, incluindo campos privados que potencialmente contêm informações confidenciais. Em vez disso, pode-se criar métodos próprios de serialização de classes específicas, para que possa controlar quais campos são expostos.

## 6.9 A9 - Utilização de Componentes Vulneráveis

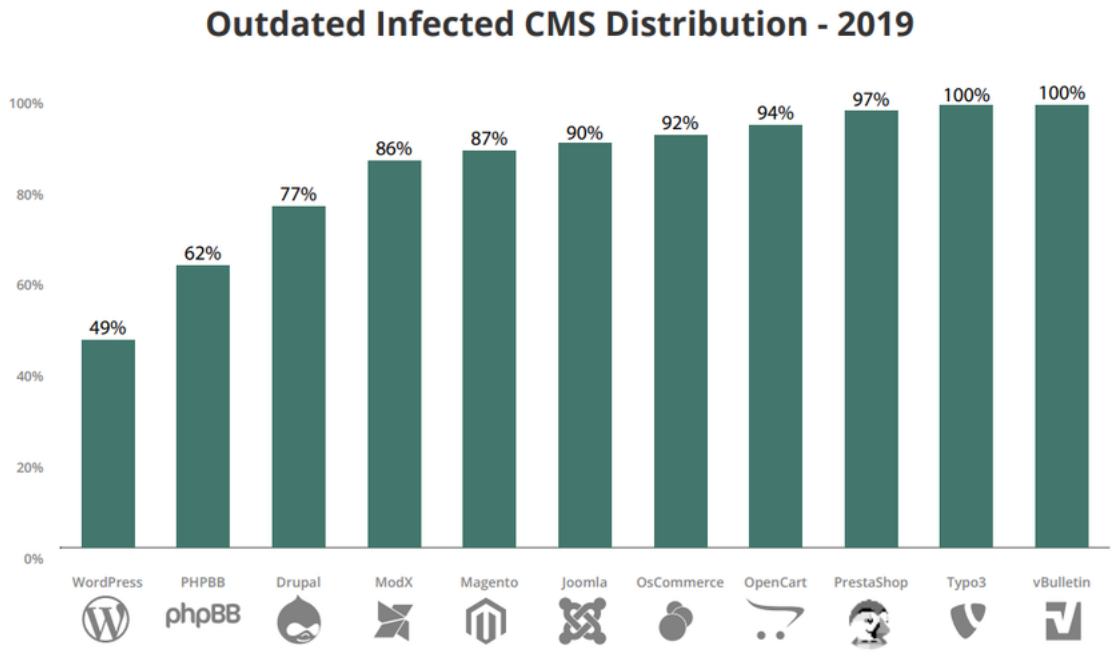
Esse tópico busca abordar a utilização de *softwares* desatualizados que possuem vulnerabilidades conhecidas. Grandes organizações possuem diversos *softwares* e é complexo manter todos atualizados o mais breve possível.

Quando uma vulnerabilidade é reportada e torna-se pública, *paths* de segurança são disponibilizados pelo fabricante, porém nem todas as corporações irão efetuar a atualização imediatamente, e os atacantes sabem disso, dessa forma vasculham a internet em busca de *softwares* com vulnerabilidades recém-descobertas, para poder explorar e comprometer os sistemas internos de grandes organizações.

Não somente grandes corporações são afetados pela utilização de componentes vulneráveis, blogs pessoais e pequenos sites que utilizam CMS (Content Management System) também são bastante afetados por falta de atualizações.

Um relatório de 2019 (figura 31) da empresa de segurança digital Sucuri apontou que 56% de todos as aplicações com CMS encontravam-se com componentes desatualizados e vulneráveis a ataques conhecidos (SUCURI, 2019).

**Figura 31 – 2019 Website Threat Research Report**




Fonte: Sucuri.net

### *6.9.1 Demonstração*

Nesse exemplo será utilizado um *plugin* vulnerável do CMS *WordPress*, que possui uma falha que ao explorada, permite ao atacante a execução de códigos no servidor da aplicação.

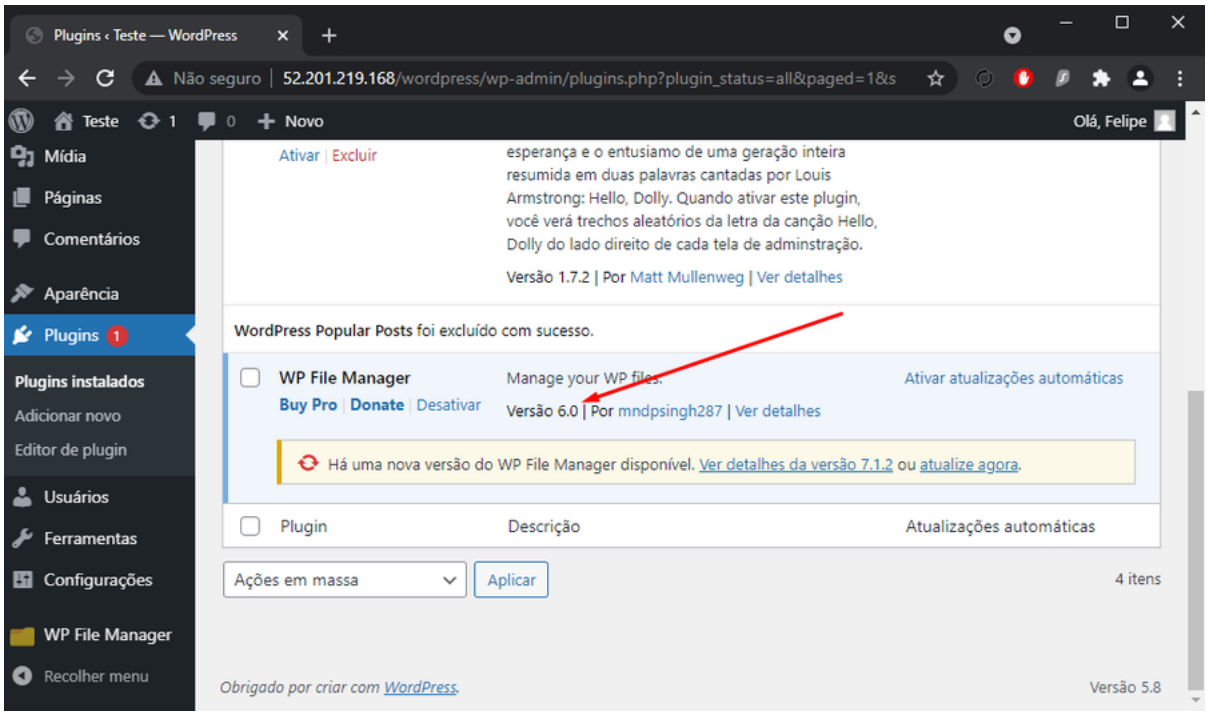
### *6.9.2 File Manager Plugin 6.0-6.9 RCE*

"O *plugi*n *File Manager* (wp-file-manager) anterior a 6.9 para *WordPress* permite que atacantes remotos carreguem e executem código PHP arbitrário pois renomeia um arquivo exemplo de um conector inseguro para ter a extensão .php. Isso, por exemplo, permite que invasores executem o comando *elFinder upload* (ou *mkfile* e *put*) para escrever código PHP no diretório *wp-content/plugins/wp-file-manager/lib/files/* Essa falha foi descoberta entre agosto e setembro de 2020". (MITRE, 2020).

#### *6.9.2.1 Exploração*

Para demonstrar esse ataque, foi necessário uma instalação do CMS *WordPress*, como também a instalação da versão vulnerável do *plugin* citado.

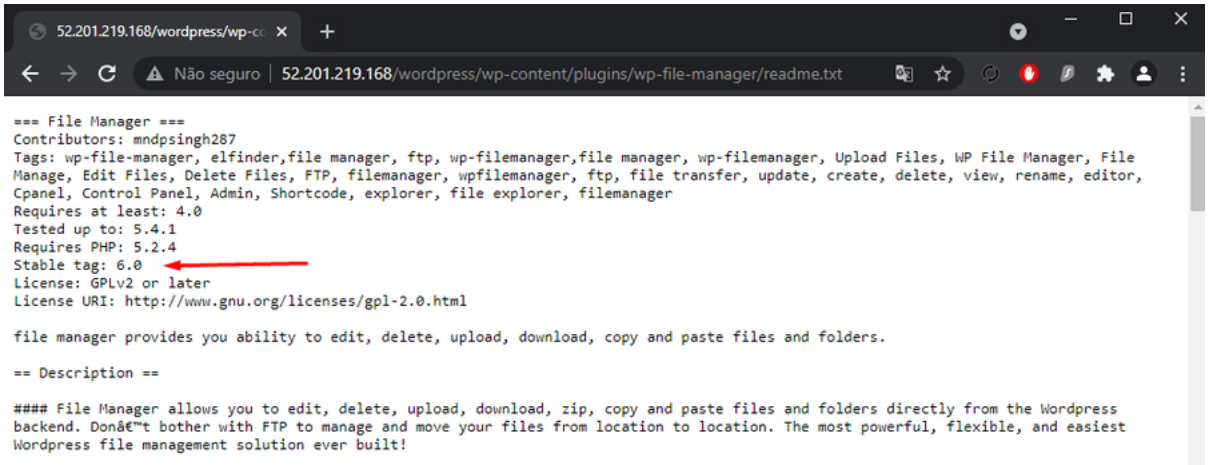
**Figura 32 – Plugin Vulnerável Instalado**




Fonte: Autor

Ao ter conhecimento dessa falha, um atacante buscaria pelos diretórios padrões de uma aplicação *WordPress*, como demonstrado abaixo (figura 33), o atacante facilmente encontraria o caminho desse *plugin*, como também o arquivo que descreve sua versão, visto que os caminhos de instalação são padronizados e conhecidos.

**Figura 33 – Plugin exposto publicamente**




Fonte: Autor

Wpscan.com (figura 34) contém o maior catálogo de falhas descobertas exclusivamente no CMS *WordPress*, o mesmo contém descrições e provas de conceito de cada nova vulnerabilidade descoberta. Abaixo temos a página referente a falha aqui discutida do *plugin File Manager*.

**Figura 34 – Prova de conceito pública**


Fonte: Wpscan.com

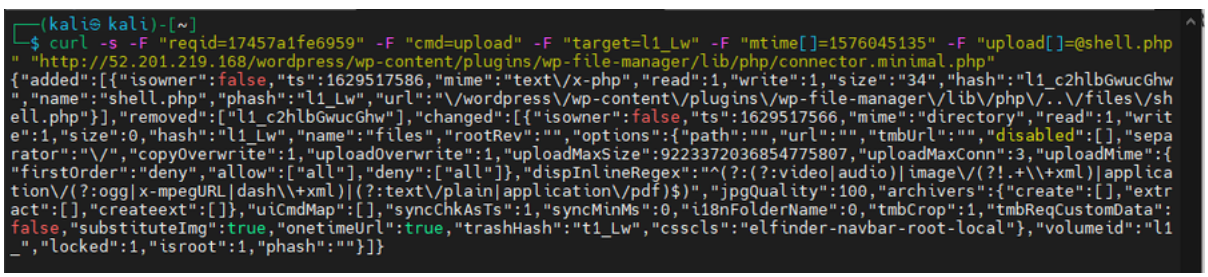
Como visto acima, a exploração ocorre através de uma simples requisição a um arquivo php, com dados de upload para o *plugin File Manager* vulnerável.

**Código 6.2 – Exploração**

```
curl -s -F "reqid=17457a1fe6959" -F "cmd=upload" -F "target=
    l1_Lw" -F "mtime[]=1576045135" -F "upload[]=@shell.php" "
    http://52.201.219.168/wordpress/wp-content/plugins/wp-file
    -manager/lib/php/connector.minimal.php"
```

A requisição maliciosa foi executada utilizando a ferramenta Curl (figura 35), pois com ela não é necessário criar um formulário HTML para submeter o *upload* ao *plugin.*

**Figura 35 – Exploração**

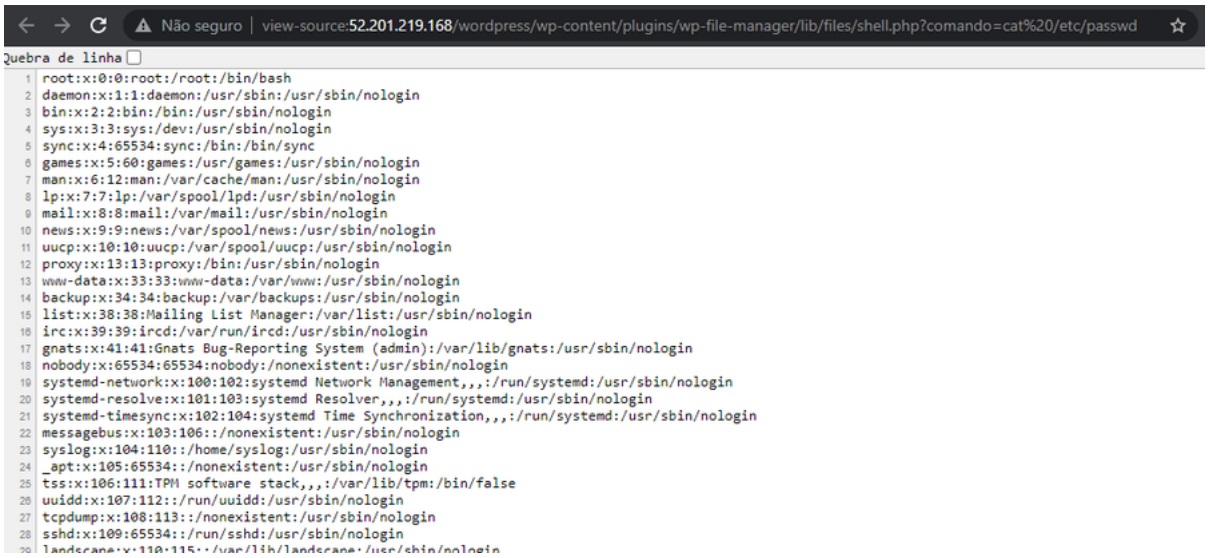



Fonte: Autor

Após executada a requisição maliciosa que efetua o *upload* de um *backdoor php* (shell.php) para a aplicação, basta que o atacante visite o diretório */wp-content/plugins/wp-*

*file-manager/lib/files/* com o nome do arquivo submetido ao final. Nesse caso foi enviado uma *web* shell *php*, que permite ao atacante a execução de comandos no servidor da aplicação (figura 36).

**Figura 36 – Execução de comandos no backdoor php inserido**




Fonte: Autor

### *6.9.3 Mitigação*

Para evitar vulnerabilidades desse tópico, é necessário sempre manter todos os *softwares* atualizados, sejam eles CMS, *plugins* ou qualquer outro componente que faça parte da aplicação *web* em execução.

## 6.10 A10 - Registro e monitorização insuficiente

Quando aplicações são configuradas, todas as ações executadas pelos usuários devem ser registradas. O registro é importante pois, no caso de um incidente, as ações dos invasores podem ser rastreadas. Uma vez que suas ações são rastreadas, seu risco e impacto podem ser determinados. Sem o registro, não haveria como saber quais ações um invasor executou e se obteve acesso à determinados sistemas. Os maiores impactos incluem:

- Danos regulatórios: Se um invasor obteve acesso à dados pessoais de usuários e não há registro disso, não apenas os usuários da aplicação são afetados, mas os proprietários podem estar sujeitos a multas ou ações mais severas, dependendo dos regulamentos.
- Risco de novos ataques: Sem registro, é difícil de detectar uma invasão. Isso pode permitir que um invasor efetue novos ataques contra a mesma aplicação, roubando credenciais, atacando a infraestrutura entre outros.

As informações armazenadas nos logs devem incluir:

- Códigos de status HTTP
- *Timestamps*
- Nomes de usuário
- Endpoints de API/diretórios
- Endereços IP

Esses registros contêm algumas informações confidenciais, portanto, é importante garantir que sejam armazenados com segurança e que várias cópias sejam armazenadas em locais diferentes

O registro é mais importante após a ocorrência de uma violação ou incidente. O caso ideal é ter um monitoramento para detectar qualquer atividade suspeita. O objetivo de detectar essa atividade suspeita é parar o invasor completamente ou reduzir o impacto causado, caso o ataque tenha sido detectado não imediatamente. Exemplos comuns de atividades suspeitas incluem:

- Várias tentativas não autorizadas para uma ação específica (geralmente tentativas de autenticação inválidas ou acesso à recursos não autorizados, como páginas de administrativas)
- Requisições de endereços IP ou locais incomuns: Embora isso possa indicar que outra pessoa está tentando acessar a conta de um usuário específico, também pode apresentar uma taxa de falsos positivos.
- Uso de ferramentas automatizadas: Determinadas ferramentas automatizadas podem ser facilmente identificáveis, exemplo: utilizando o determinado *User Agent* ou a velocidade que requisições são recebidas. Isso pode indicar que um invasor está utilizando ferramentas automatizadas.
- *Payloads* comuns: Em aplicações *web*, é comum que os invasores utilizem *payloads* XSS. Detectar esses usos pode indicar a presença de alguém conduzindo testes não autorizados/maliciosos no ambiente.

Apenas detectar atividades suspeitas não ajuda. Essa atividade suspeita deve ser avaliada de acordo com o nível de impacto. Por exemplo, certas ações terão maior impacto do que outras. Essas ações de maior impacto precisam ser respondidas mais cedo, portanto, elas devem incluir um alarme que desperte a atenção da parte responsável.

# 7 BOAS PRÁTICAS

A partir da análise prática de todas as vulnerabilidades de segurança em aplicações *web* do último top 10 OWASP, foi possível obter uma visão ampla dos principais riscos e cuidados ao manipular qualquer tipo de site, seja ele governamental, institucional ou até mesmo *blogs* pessoais. Nenhuma aplicação está totalmente segura, milhares de falhas surgem todos os dias, e as falhas aqui demonstradas formam os pilares para a maior parte dos problemas de segurança em qualquer aplicação de acordo com a metodologia OWASP.

Dentro disso, com base em todas as recomendações da OWASP das principais vulnerabilidades existentes, será descrito abaixo um guia de boas práticas e cuidados para manter uma aplicação *web* mais segura, de modo a auxiliar desenvolvedores a tornar a internet um lugar mais seguro, tentando evitar que seus sistemas sejam comprometidos e dados confidenciais sejam acessados por Hackers.

- Entradas de usuários devem ser sanitizadas com base em termos ou caracteres permitidos, dados não permitidos devem ser removidos antes de processados.
- Utilizar políticas de senha forte para usuários e administradores.
- Bloquear automaticamente um IP após certo número de tentativas de login inválidas.
- Implementar autenticação multifator.
- Desabilitar resolução de entidades externas e desabilitar suporte ao *XInclude*, em aplicações que façam utilização de XML.
- Negar o acesso a todos os recursos da aplicação não disponíveis publicamente.
- Padronizar mecanismos de controle de acesso em toda a aplicação.
- Implementar cabeçalhos *Content-Type* e *X-Content-Type* para evitar ataques XSS.
- Incorporar medidas robustas de desserialização para dados não confiáveis, como entradas de usuários.
- Evitar utilização de recursos genéricos para desserialização.
- Sempre manter atualizados os *softwares*, CMS, *plugins* e todos os componentes que compõem a aplicação.
- Manter de forma segura o máximo de registros de logs possíveis, sobre o funcionamento da aplicação e sua interação com usuários e administradores.

# 8 CONCLUSÃO

Este trabalho apresenta um estudo prático que se propõe definir e demonstrar vulnerabilidades em aplicações *web*, de forma que possa ser útil para analistas de segurança e desenvolvedores como referência sobre ataques relacionados a metodologia da OWASP. O resultado do estudo é o entendimento por trás do processo de cada falha abordada, e a criação de uma lista de boas práticas que devem ser adotadas para tornar uma aplicação *web* mais segura.

O guia proposto foi baseado na lista OWASP top 10 web 2017 e busca uma abordagem prática para a análise de aplicativos *web* em algumas etapas como: descrição, demonstração e mitigação. Cada um desses passos foi explanado em detalhes para a compreensão do leitor, a respeito da vulnerabilidade abordada em cada tópico.

A partir do estudo proposto, foi feita uma descrição sobre cada vulnerabilidade, como também uma demonstração prática, com o auxílio de laboratórios específicos para cada caso. Por fim, foram discutidos os possíveis passos para a mitigação do problema causado pela falha existente.

No geral, o estudo se mostrou eficaz para seu objetivo por conseguir demonstrar de forma prática ao leitor, o processo por trás de cada vulnerabilidade existente, e espera-se que seja uma contribuição não só para a comunidade acadêmica, mas para a comunidade de segurança de informação em geral, principalmente desenvolvedores e *pentesters*.

## 8.1 Trabalhos Futuros

Como possível trabalho futuro pode-se sugerir uma análise do mais recente Top 10 Web OWASP publicado em setembro de 2021, que conta com novas vulnerabilidades na lista, e também traz reformulações como agrupamento de algumas falhas em novos conjuntos. Seria um trabalho para atualizar ainda mais o meio acadêmico e técnico em relação à segurança de aplicações *web*, e auxiliar no desenvolvimento mais seguro das aplicações com base no que os invasores mais buscam explorar recentemente.

# REFERÊNCIAS

ACUNETIX. **What is Insecure Deserialization?** 2017. Disponível em: https://www.acunetix.com/blog/articles/what-is-insecure-deserialization/. Acesso em: 06 ago. 2021.

BACH-NUTMAN, M. **Understanding The Top 10 OWASP Vulnerabilities**. Bournemouth, United Kingdom, 2020. Disponível em: https://arxiv.org/abs/2012.09960. Acesso em: 06 ago. 2021.

CABRAL NETO, J. da C.; BRANDÃO, R. da S. **Mitigando Ataque Aplicando Boas Práticcas No Desenvolvimento Seguro de Aplicações Web**. 2019. 39 p. — Centro Universitário de João Pessoa. Disponível em: https://bdtcc.unipe.edu.br/wp-content/uploads/2019/06/TCC-FINAL.pdf. Acesso em: 28 jul. 2021.

FERREIRA, F. N. F.; ARAÚJO, M. T. de. **Políticas de Segurança da Informação - Guia Prático para elaboração e implementação**. Rio de Janeiro: Ciência Moderna, 2008.

GOMES, J. F. **Um guia para análise de segurança de aplicativos na plataforma Android**. 2017. 51 p. Monografia (Sistemas de Informação) — Universidade Federal do Ceará. Disponível em: http://www.repositorio.ufc.br/bitstream/riufc/29519/1/2017_tcc_jfgomes.pdf. Acesso em: 15 mai. 2021.

LINS, B. F. E. A evolução da Internet: uma perspectiva histórica. **Cadernos Aslegis**, n. 48, p. 11 – 46, 01 2013. Disponível em: https://bd.camara.leg.br/bd/bitstream/handle/bdcamara/33179/evolucao_internet_lins.pdf. Acesso em: 22 mai. 2021.

MITRE. **CVE-2020-25213**. 2020. Disponível em: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-25213. Acesso em: 06 ago. 2021.

NEGRI MORENO, D. H. **Pentest em aplicações web**. São Paulo: Novatec, 2017. Disponível em: https://novatec.com.br/livros/pentest-em-aplicacoes-web/. Acesso em: 03 mai. 2021.

NIC.BR. **CERT.br aponta aumento de notificações de ataques a servidores Web**. 2016. Disponível em: https://www.nic.br/noticia/releases/cert-br-aponta-aumento-de-notificacoes-de-ataques-a-servidores-web/. Acesso em: 25 jul. 2021.

NOVASKI NETO, D. **Análise de Vulnerabilidades Web e de Ferramentas de Código Aberto Para Exploração**. 2019. 92 p. Dissertação (Mestrado em Informática) — Universidade Federal do Paraná. Disponível em: https://www.acervodigital.ufpr.br/bitstream/handle/1884/65693/R-D-DAVIDNOVASKINETO.pdf. Acesso em: 09 ago. 2021.

OWASP. **About the OWASP Foundation**. 2017a. Disponível em: https://owasp.org/about/. Acesso em: 10/08/2021.

OWASP. **OWASP Top 10 - 2017 Os Dez Riscos de Segurança Mais Críticos em Aplicações Web**. 2017b. Disponível em: https://owasp.org/www-pdf-archive/OWASP_Top_10-2017-pt_pt.pdf. Acesso em: 06 mai. 2021.

PORTSWIGGER. **WebSecurity Academy**. Disponível em: https://portswigger.net/web-security. Acesso em: 01 ago. 2021.

QUINN, L. S. **Comparing Online vs. Traditional Software**. 2010. Disponível em: https://www.techsoup.org/support/articles-and-how-tos/comparing-online-vs-traditional-software. Acesso em: 15 mai. 2021.

SÊMOLA, M. **Gestão da Segurança da Informação - Uma Visão Executiva**. 2. ed. [S.l.]: Elsevier, 2014.

SØHOEL, H. M. **OWASP top ten - What is the state of practice among start-ups?** 2018. 94 p. Dissertação (Master of Science in Communication Technology) — Norwegian University of Science and Technology. Disponível em: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2487106. Acesso em: 03 ago. 2021.

SUCURI. **Website Threat Research Report**. 2019. Disponível em: https://sucuri.net/wp-content/uploads/2020/01/20-sucuri-2019-hacked-report-1.pdf. Acesso em: 15 ago. 2021.