ANÁLISE DAS VULNERABILIDADES *TOP 10 OWASP* 2021

Vitor Bonfá Domingos; Prof. Me. Orientador: José Aparecido de Aguiar Viana

e-mail:
vitorddomingos@gmail.com; jose.viana@fatec.sp.gov.br

Resumo: Com o crescente número de ataques cibernéticos, o assunto de segurança no meio digital se torna cada vez mais relevante. O tipo de aplicação que se torna cada vez mais presente no meio digital é a chamada "aplicação *web*", sendo caracterizada por ser um *software* que é executado em um navegador *web*. Dito isso, o presente trabalho faz uma análise prática das vulnerabilidades mais comuns em aplicações *web*, as quais foram demarcadas no *OWASP Top* 10 2021, uma das principais documentações que delimitam as vulnerabilidades mais comuns para desenvolvedores que atuam no meio *web*. A análise será feita por meio de análise de casos reais e replicações por meio de códigos *Ruby*, mostrando passos de mitigação e cenários prováveis que podem acontecer caso as vulnerabilidades não sejam devidamente tratadas.

Palavras-chave: *OWASP* 2021. Segurança *Web*. Vulnerabilidades *Web*.

Abstract: *With the growing number of cyber-attacks, the subject of security in the digital environment is becoming increasingly relevant. The type of application that is becoming increasingly present in the digital environment is the so-called "web application", which is characterized by being software that runs in a web browser. Having said that, this paper makes a practical analysis of the most common vulnerabilities in web applications, which have been ranked in the OWASP Top 10 2021, one of the main documentations that delimit the most common vulnerabilities for developers working in the web environment. The analysis will be carried out by analyzing real cases and replications using Ruby code, showing mitigation steps and probable scenarios that could occur if the vulnerabilities are not properly addressed.*

Keywords: *OWASP 2021. Web Security. Web Vulnerabilities.*

1 Introdução

Nos últimos anos, a crescente dependência da sociedade moderna em tecnologias digitais tem sido acompanhada por uma preocupação igualmente crescente com a segurança cibernética. Com a proliferação de aplicativos da *web* e sistemas *online*, a vulnerabilidade desses sistemas a ataques maliciosos se tornou uma questão central. Entre as muitas fontes de orientação e padrões de segurança disponíveis, o *Top* 10 do *OWASP* (*Open Web Application Security Project*) se destaca como um guia fundamental para identificar e mitigar as principais ameaças à segurança em aplicações *web*.

Essa pesquisa propõe uma análise prática e aprofundada do *Top 10 OWASP*, examinando cada uma das vulnerabilidades listadas e investigando sua relevância e impacto no contexto das aplicações *web* atuais. A abordagem adotada neste estudo visa fornecer uma compreensão holística das ameaças à segurança cibernética enfrentadas pelas organizações e

desenvolvedores, além de oferecer *insights* valiosos sobre as melhores práticas de proteção contra essas ameaças.

Ao longo deste trabalho, serão realizados estudos de caso e demonstrações práticas para ilustrar os riscos associados a cada uma das vulnerabilidades do *ranking*, bem como as estratégias e ferramentas disponíveis para mitigar tais riscos. Além disso, serão discutidas as implicações mais amplas dessas vulnerabilidades em termos de confidencialidade, integridade e disponibilidade dos dados, bem como as consequências financeiras e reputacionais para as organizações afetadas.

Por meio desta análise prática, este trabalho busca contribuir para o avanço do conhecimento e das práticas de segurança cibernética, capacitando desenvolvedores, profissionais de segurança e tomadores de decisão a protegerem efetivamente suas aplicações *web* contra as ameaças emergentes e persistentes. Ao entender e enfrentar as vulnerabilidades destacadas, as organizações podem fortalecer suas defesas cibernéticas e garantir a segurança de seus sistemas e dados em um ambiente digital cada vez mais hostil e complexo.

2 Justificativa

A complexidade do ambiente digital contemporâneo exige uma abordagem abrangente e prática para garantir a segurança das aplicações *web*. Nesse contexto, o *Top 10 OWASP* se destaca como um recurso fundamental para identificar e priorizar as principais vulnerabilidades que ameaçam a integridade e a segurança das aplicações *web*. No entanto, a implementação efetiva das diretrizes do *OWASP* muitas vezes enfrenta obstáculos práticos e técnicos, exigindo uma análise aprofundada e prática para oferecer soluções tangíveis.

Esta pesquisa se justifica pela necessidade de preencher essa lacuna entre teoria e prática no campo da segurança cibernética, fornecendo uma análise detalhada e empiricamente fundamentada das vulnerabilidades abordadas. Ao compreender não apenas os conceitos teóricos por trás das ameaças cibernéticas, mas também suas implicações práticas e estratégias eficazes de mitigação.

Dada a natureza dinâmica do cenário de segurança cibernética, é essencial manter uma abordagem atualizada e pragmática para garantir a eficácia contínua das defesas cibernéticas. Oferecendo *insights* acionáveis e práticos, o presente trabalho busca não apenas aumentar a conscientização sobre as ameaças à segurança cibernética, mas também capacitar as organizações e indivíduos a protegerem suas aplicações *web* contra essas ameaças em constante evolução.

Dito isso, ao realizar uma análise prática e baseada em evidências do *Top 10 OWASP*, este trabalho visa fornecer contribuições significativas para a comunidade de segurança cibernética, promovendo a adoção de práticas eficazes de segurança e fortalecendo as defesas contra ameaças cibernéticas em um mundo digital cada vez mais interconectado e vulnerável.

3 Objetivos

Este trabalho tem como objetivo principal realizar uma análise aprofundada das vulnerabilidades mais críticas em aplicações *web*, com base no *OWASP Top* 10 de 2021, buscando não apenas catalogar essas falhas, mas também examinar suas causas raiz, vetores de exploração e impactos em diferentes cenários reais. O estudo visa oferecer uma visão holística sobre como essas vulnerabilidades comprometem a segurança de sistemas, desde vazamentos de dados até comprometimento total de infraestruturas, analisando casos concretos que demonstram a gravidade dessas ameaças.

Além da análise técnica, este trabalho tem como propósito desenvolver estratégias práticas de mitigação, adaptáveis a diversos contextos de desenvolvimento. Também pretende-se fornecer recomendações pontuais e práticas, respaldadas em recomendações do *OWASP*, e orientações para implementação de controles preventivos e detectivos ao longo do ciclo de vida do *software*.

Como objetivos complementares, busca-se:

- Educar desenvolvedores e profissionais de segurança através da criação de um material de referência que detalhe as vulnerabilidades mais recorrentes, seus métodos de exploração e técnicas de prevenção, servindo como guia prático para integração de boas práticas de segurança desde as fases iniciais de design e codificação. O intuito é reduzir a ocorrência dessas falhas por meio do conhecimento aplicado, demonstrando como vulnerabilidades como *injection flaws* ou *broken access control* podem ser evitadas com abordagens como *input validation* e princípio do menor privilégio;
- Promover a conscientização sobre segurança no ecossistema tecnológico, destacando a importância da segurança cibernética não apenas como uma preocupação técnica, mas como um requisito estratégico para organizações de todos os portes. O trabalho visa sensibilizar *stakeholders* sobre os riscos financeiros,

legais e reputacionais associados a brechas de segurança, ilustrando com exemplos reais de violações de dados e seus impactos devastadores;

- Contribuir para a comunidade de segurança ao compartilhar descobertas, análises e recomendações que possam enriquecer o conhecimento coletivo no campo da segurança de aplicações *web*. A divulgação dos resultados deste estudo em fóruns técnicos, artigos e *workshops* busca fomentar discussões e colaborações que impulsionem inovações em defesa cibernética, fortalecendo a resiliência da comunidade contra ameaças emergentes.

Em síntese, este trabalho almeja ser uma ponte entre a teoria e a prática, transformando conhecimento sobre vulnerabilidades em ações concretas que elevem o patamar de segurança em aplicações *web*, beneficiando desenvolvedores, empresas e usuários finais.

4 Fundamentação Teórica

A fundamentação teórica deste trabalho parte dos princípios essenciais da segurança cibernética e das metodologias de desenvolvimento seguro de *software*, tendo como base principal as diretrizes estabelecidas pelo *OWASP*. Reconhecida globalmente como uma das principais autoridades no tema, a *OWASP* destaca-se por sua abordagem aberta e colaborativa, oferecendo recursos técnicos, ferramentas e documentação que servem como referência para profissionais de segurança e desenvolvedores (*OWASP*, 2021). Desde sua criação em 2001, a organização tem desempenhado um papel crucial na evolução das práticas de segurança aplicadas ao desenvolvimento *web*, com o objetivo de combater vulnerabilidades críticas por meio de conhecimento acessível e atualizado.

Dentre as diversas iniciativas mantidas pela *OWASP*, o *Top* 10 se consolida como um dos projetos mais relevantes, sintetizando as ameaças mais críticas e frequentes em aplicações *web*. Atualizado periodicamente, o relatório não apenas cataloga essas vulnerabilidades, mas também contextualiza seu impacto, vetores de ataque e tendências emergentes, refletindo a dinâmica evolutiva do cenário de ameaças. A edição de 2021, que serve como referência para este estudo, incorpora lições aprendidas com incidentes recentes, ajustando suas prioridades para incluir riscos modernos como falhas em componentes de *software* e con

ções inseguras, além de manter alertas sobre problemas persistentes, como injeção de *SQL* (*Structured Query Language*) e quebras de autenticação (*OWASP*, 2021).

A relevância do *Top 10 OWASP* é ainda mais evidente quando considerado a história da segurança em aplicações *web*, marcada por brechas recorrentes que exploram deficiências tanto técnicas quanto conceituais. Vulnerabilidades como *Cross-Site Scripting* (*XSS*) e injeção de comandos, por exemplo, remontam aos primórdios da *web*, mas continuam presentes em sistemas contemporâneos devido à negligência em práticas básicas de codificação segura. Nesse contexto, o *Top* 10 surge não apenas como um documento de alerta, mas como um guia prático para a adoção de controles preventivos, ajudando organizações a priorizar esforços de segurança de forma eficiente.

Este trabalho não apenas se beneficia de uma estrutura metodológica validada pela comunidade internacional de segurança, mas também se conecta a um acervo de casos reais e soluções testadas, essenciais para uma análise crítica das vulnerabilidades e suas contramedidas. A fundamentação aqui apresentada reforça a importância de integrar segurança desde as etapas iniciais do desenvolvimento, alinhando-se ao conceito de "*Security by Design*", também sublinhado pela *OWASP* (2021) e por estudos como os de Soldera (2023) que evidenciam o papel da arquitetura segura para mitigar riscos antes mesmo da fase de *deploy.*

5 Trabalhos Similares

5.1 Uma análise prática das principais vulnerabilidades em aplicações *web* baseado no *Top 10 OWASP*

O *TCC* (Trabalho de Conclusão de Curso) feito por Sampaio (2021) tem um objetivo similar ao presente trabalho, proporcionando uma análise prática das principais vulnerabilidades em aplicações *web*, baseado no *Top 10 OWASP*. O atual trabalho visa expandir e atualizar o conhecimento propagado por Sampaio, já que em seu trabalho foi utilizado o *Top 10 OWASP* 2017, lista que foi substituída pelo *ranking* mais recente de 2021, o qual analisa as vulnerabilidades de 2017 a 2021.

Em seu estudo, foi feito uma descrição sobre cada vulnerabilidade, como também, uma demonstração prática por meio de laboratórios específicos. Posteriormente, foram discutidos os possíveis passos para a mitigação do problema causado pela falha existente. As vulnerabilidades presentes no *ranking* de 2017, e consequentemente, abordadas por Sampaio (2021) em seu *TCC*, foram:

- *Injection* (Injeção): Essa vulnerabilidade permite que invasores insiram códigos maliciosos em aplicativos, explorando falhas na validação de entrada. Com ataques

como *SQL injection*, *LDAP* (*Lightweight Directory Access Protocol*) *injection* ou *XML* (*eXtensible Markup Language*) *injection*, os invasores podem manipular consultas de banco de dados, obter acesso não autorizado a sistemas e até mesmo executar comandos de forma remota;

- *Broken Authentication* (Autenticação Quebrada): Quando a autenticação não é implementada corretamente, invasores podem explorar falhas como autenticação fraca, gestão inadequada de sessões ou falta de proteção adequada de credenciais para assumir a identidade de usuários legítimos, obtendo acesso não autorizado a sistemas e dados;
- *Sensitive Data Exposure* (Exposição de Dados Sensíveis): Esta vulnerabilidade ocorre quando informações sensíveis, como senhas, números de cartões de crédito ou dados de saúde, não são devidamente protegidas. Isso pode resultar em acesso não autorizado a dados confidenciais, expondo-os a roubo de identidade, fraudes ou vazamento de informações;
- *XML External Entities* (*XXE*): Ataques de *XXE* exploram falhas na manipulação de documentos *XML* para acessar recursos arbitrários do servidor. Ao injetar entidades externas maliciosas em documentos *XML*, os invasores podem ler arquivos do sistema, executar varreduras de portas ou realizar ataques de negação de serviço;
- *Broken Access Control* (Controle de Acesso Quebrado): Quando as restrições de acesso não são implementadas corretamente, os invasores podem acessar recursos não autorizados ou realizar ações além de seus privilégios autorizados. Isso pode incluir acesso a dados confidenciais, funcionalidades administrativas ou alteração indevida de informações;
- *Security Misconfiguration* (Configuração de Segurança Incorreta): Erros na configuração do ambiente de aplicação podem expor inadvertidamente sistemas a ameaças. Configurações padrão não seguras, informações de *debug* expostas ou permissões excessivas podem facilitar ataques, permitindo que invasores explorem vulnerabilidades;
- *XSS*: Permite que invasores injetem *scripts* maliciosos no lado do cliente, comprometendo a integridade e segurança das páginas *web*. Com *XSS*, os invasores podem roubar *cookies* de sessão, redirecionar usuários para *sites* maliciosos ou roubar informações confidenciais;

- *Insecure Deserialization* (Desserialização Insegura): Ao manipular objetos durante a desserialização, os invasores podem executar ataques como execução remota de código, negação de serviço ou escalonamento de privilégios. Isso ocorre quando os aplicativos confiam cegamente nos dados desserializados, sem verificar sua integridade ou origem;
- *Using Components with Known Vulnerabilities* (Uso de Componentes com Vulnerabilidades Conhecidas): A incorporação de componentes de *software* com falhas de segurança conhecidas pode expor aplicativos a ataques. Sem atualizações regulares ou monitoramento de vulnerabilidades, os invasores podem explorar essas falhas para comprometer sistemas e dados;
- *Insufficient Logging and Monitoring* (*Log* e Monitoramento Insuficientes): A falta de registro adequado de atividades de segurança e monitoramento ativo de sistemas dificulta a detecção e resposta a incidentes. Sem registros detalhados ou alertas de segurança, os invasores podem realizar ataques sem serem detectados, prolongando a exposição a ameaças.

5.2 *Understanding the Top 10 OWASP*

Bach-Nutman (2020) realizou um estudo mais superficial e teórico sobre as vulnerabilidades que constam no *OWASP*, explicando as vulnerabilidades e os possíveis impactos que essas podem ter em usuários e empresas, reforçando que se deve analisar cada vulnerabilidade identificada no contexto único daquela organização. O objetivo do artigo é aumentar a compreensão sobre essas vulnerabilidades, discutir os métodos de mitigação e enfatizar a importância da segurança no desenvolvimento de *software*.

O estudo de Bach-Nutman (2020) também abordou todas as vulnerabilidades presentes no *ranking*, oferecendo um conhecimento básico do funcionamento de todas elas, para que mesmo uma pessoa sem muito conhecimento técnico tenha pelo menos uma compreensão geral das vulnerabilidades.

5.3 Análise de vulnerabilidades em autenticação

Soldera (2023), em seu trabalho também baseado no *OWASP* 2021, fez uma análise focada nas vulnerabilidades que envolvem autenticação. As vulnerabilidades analisadas foram:

- *SQL Injection* (Injeção de *SQL*): Vulnerabilidade que permite que um invasor injete comandos *SQL* maliciosos em uma aplicação *web*, explorando falhas na validação de entrada. Isso pode levar à execução não autorizada de consultas *SQL* no banco de dados, comprometendo a integridade e a segurança dos dados armazenados;
- *XSS* (*Script* entre *sites*): Esta vulnerabilidade permite que um invasor injete *scripts* maliciosos em páginas *web*, que são então executados nos navegadores dos usuários legítimos. Isso pode resultar em roubo de sessão, redirecionamento para *sites* maliciosos ou roubo de informações confidenciais do usuário;
- *Brute Force Attack* (Ataque de Força Bruta): Consiste em tentativas repetidas de adivinhar credenciais de usuário, como nomes de usuário e senhas, até encontrar combinações válidas. É uma abordagem simples, mas eficaz, para contornar a autenticação e obter acesso não autorizado a sistemas ou contas de usuários;
- *Broken Access Control* (Controle de Acesso Quebrado): Refere-se a falhas na implementação de restrições de acesso adequadas, permitindo que usuários acessem recursos ou funcionalidades não autorizadas. Isso pode resultar em exposição de informações confidenciais, comprometendo a integridade e a segurança do sistema de autenticação.

Todas essas vulnerabilidades têm um impacto direto na autenticação, que é o processo de verificar a identidade de um usuário. O *SQL Injection* pode comprometer a integridade do sistema de autenticação ao permitir a injeção de comandos *SQL* maliciosos. O *XSS* pode roubar *cookies* de sessão, que são usados para autenticar usuários. Os ataques de força bruta visam diretamente as credenciais de autenticação dos usuários. E o controle de acesso quebrado pode permitir que usuários não autenticados ou não autorizados acessem recursos ou funcionalidades que deveriam estar protegidos.

6 Metodologia

Para atingir os objetivos propostos, foi utilizada uma combinação de pesquisa bibliográfica, análise de casos reais, e aplicação prática de ferramentas de segurança. A seguir, será detalhado os materiais e métodos que foram utilizados.

6.1 Pesquisa Bibliográfica

A pesquisa bibliográfica constitui a base teórica necessária para a compreensão das vulnerabilidades de segurança descritas no *OWASP Top* 10. Foram revisados livros, artigos

científicos, relatórios técnicos e publicações da própria *OWASP*. Esta revisão permite a fundamentação teórica sólida e atualizada sobre os principais conceitos e práticas em segurança cibernética e desenvolvimento seguro de *software*.

6.2 Análise de Casos Reais

Para ilustrar a aplicação prática das vulnerabilidades e suas implicações, foram analisados casos reais de ataques cibernéticos reportados na literatura e na mídia. Esta análise ajudará a contextualizar a importância das práticas de segurança e a eficácia das medidas de mitigação propostas pela *OWASP*.

6.3 Ferramentas e Recursos Utilizados

A análise das vulnerabilidades será realizada utilizando uma série de ferramentas e recursos amplamente reconhecidos na área de segurança da informação. As principais ferramentas e métodos incluem:

- *VSCode* com Extensões *Ruby*: Ambiente completo para desenvolvimento seguro em Ruby, incluindo análise de código, *debug* e testes de vulnerabilidades diretamente no editor. Configurado com *plugins* essenciais como *RuboCop* para análise estática, *Solargraph* para *autocomplete* inteligente e o terminal integrado para execução rápida de *scripts* de teste;
- Firefox Developer Edition: Navegador escolhido para pesquisa de vulnerabilidades, equipado com ferramentas integradas de inspeção como o *Network Monitor* para análise de requisições *HTTP/HTTPS* (*Hypertext Transfer Protocol/Hypertext Transfer Protocol Secure*) e o *debugger* para *JavaScript*;
- Microsoft *Word*: *Software* principal para documentação formal do projeto, com formatação customizável para padrões da *ABNT* (Associação Brasileira de Normas Técnicas);
- Windows 11 Pro: Sistema operacional escolhido para realizar todas as etapas do desenvolvimento do projeto e dos exemplos de código, incluindo pesquisa bibliográfica e formatação do artigo.

7 Desenvolvimento

O desenvolvimento deste trabalho envolve a realização de várias etapas, cada uma essencial para alcançar os objetivos propostos. A seguir, está detalhado cada uma das

vulnerabilidades do *OWASP Top* 10, mostrando: uma breve introdução de cada vulnerabilidade, formas comuns de exploração, casos reais, exemplo da vulnerabilidade por um trecho em *Ruby,* recomendações de mitigação e possíveis impactos da exploração de cada vulnerabilidade.

7.1 *A01:2021 – Broken Access Control*

O Controle de Acesso garante que usuários só possam atuar dentro de suas permissões. Quando esse controle falha, usuários não-autorizados podem expor, modificar ou destruir dados protegidos e executar funções restritas (*OWASP*, 2021). Em outras palavras, falhas em autorização permitem escalonamento de privilégios (horizontal ou vertical) e acesso indevido.

Falhas pertencentes a essa vulnerabilidade têm impacto elevado, fato respaldado por dados do *OWASP*: 318 mil aplicações do *dataset* utilizado apresentaram algum tipo de falha nessa categoria, que lidera o *Top* 10 2021.

7.1.1 Formas Comuns de Exploração

Engloba falhas como *IDOR* (*Insecure Direct Object References*), em que identificadores de objetos (IDs) são manipuláveis por usuários não autorizados; escalonamento de privilégio horizontal (um usuário comum acessa dados de outro) e vertical (usuário comum obtém privilégios de administrador); *bypasses* de autenticação (ex.: requisições manipuladas no servidor ou cabeçalhos adulterados); e *CORS* mal configurado, que permite origens não confiáveis acessarem *APIs* internas. Outros subtipos incluem acesso impróprio por *Uniform Resource Location (URL)* não protegida e manipulação de *tokens*/*JWTs* sem validação de assinatura, garantindo que usuários obtenham capacidades além das previstas.

Exemplos clássicos incluem: ignorar verificações mudando parâmetros de *URL*, referências diretas inseguras a objetos (*ID* em rota), elevação de privilégio (atuar como *admin* sem permissão), *tokens JWT* manipulados, ou configuração incorreta de *CORS*. Em um cenário real, um atacante pode alterar o parâmetro de *ID* em uma *URL REST* para visualizar ou editar dados de outro usuário sem autorização.

7.1.2 Casos Reais

Caso *USPS* (2018): O Serviço Postal dos *EUA* (*USPS*) expôs cerca de 60 milhões de contas de usuários devido a uma falha de controle de acesso em uma *API* que estava atrelado a uma iniciativa postal chamada "*Informed Visibility*". Qualquer usuário autenticado no portal do *USPS* podia visualizar dados de outros clientes (e-mail, endereço, telefone etc.) apenas alterando parâmetros de requisição. Trata-se de um clássico erro de "*IDOR*" (Referência de Objeto Direto não Seguro), onde o serviço retorna informações sensíveis sem verificar privilégios. A falha foi reportada anonimamente em 2017 e só corrigida após divulgação em 2018 (ASERTO, 2023).

Não houve evidência de uso malicioso além da exposição em si, mas o incidente ressaltou a importância de implementar autorização robusta: o *USPS* só corrigiu o componente afetado semanas após a denúncia pública, evitando multas imediatas, mas levando a críticas na mídia sobre gestão de segurança.

Caso *First American Financial* (2019): A gigante de seguros imobiliários *First American* deixou disponível, sem autenticação, um repositório online contendo aproximadamente 885 milhões de documentos de transações imobiliárias. Um desenvolvedor notou que bastava alterar um dígito no identificador do documento para acessar registros alheios. Os documentos incluíam informações financeiras e pessoais (números de conta bancária, *SSNs*, recibos de transferência, imagens de *CNH* etc.). Ao ser notificada via imprensa, a empresa desativou o portal em horas (KREBON SECURITY, 2019). Em seguida, foi alvo de investigação da SEC e reguladores estaduais por suposta violação de obrigações de segurança, além de processo coletivo alegando falta de "medidas mínimas de segurança" (KREBON SECURITY, 2019). Embora não tenha havido pronunciamento público de perdas financeiras diretas, estima-se que os custos internos com perícia forense, monitoramento de crédito e eventuais sanções regulatórias ultrapassaram dezenas de milhões de dólares. Este caso ilustra que mesmo grandes organizações podem falhar em implementar controles de acesso nos sistemas legados, levando a repercussões legais e revisões de políticas internas, para que, apenas assim, autenticações e autorizações adequadas em *APIs* e serviços *web* sejam verdadeiramente levadas a sério.

7.1.3 Exemplo da Vulnerabilidade

Utilizando como exemplo o caso real da *First American*, no qual não havia qualquer exigência de autenticação para acessar registros confidenciais, é possível ilustrar como essa

vulnerabilidade poderia estar presente na base de código da aplicação. Na figura 1, é apresentado um exemplo em *Ruby on Rails* que simula um trecho vulnerável, no qual não há verificação de autorização antes do acesso aos dados sensíveis:

Figura 1 Código vulnerável A01

```ruby
class DocumentsController < ApplicationController
  def show
    @document = Document.find(params[:id])     # Vulnerável
    render json: @document
  end
end
```

Fonte: Autores

No código representado pela figura 1 não há nenhum filtro de autorização (como a utilização de um *before_action*) ou verificação de propriedade do usuário. Isso significa que qualquer usuário, mesmo sem privilégios, pode acessar qualquer documento desde que conheça seu *ID*. Em um cenário real, isso permitiria vazamento de dados confidenciais (como contas bancárias e *SSNs* no caso da *First American*).

7.1.4 Recomendações de Mitigação

Utilizando o código apresentado como exemplo, uma possível correção seria aplicar um controle de acesso explícito. Como por exemplo:

Figura 2 Código corrigido A01

```ruby
class DocumentsController < ApplicationController
  before_action :authenticate_user!
  def show
    @document = current_user.documents.find(params[:id])
    render json: @document
  end
end
```

Fonte: Autores

Com a utilização de um controle de acesso explícito, como exemplificado pela restrição de busca por documentos apenas do usuário atual, a vulnerabilidade apresentada anteriormente foi completamente mitigada, já que o usuário não conseguirá mais visualizar um documento que não pertence a ele.

As recomendações usuais, são:

- Implementar verificações de autorização (ex.: *before_action :authorize_user*) para garantir que o usuário logado só possa operar seus próprios recursos;
- Usar *gems/frameworks* de autorização (Pundit, CanCanCan) definindo políticas claras;

- Não confiar em parâmetros do usuário para controle de acesso. Sempre validar *server-sid*e que o recurso pertence ao usuário logado;
- Princípio do menor privilégio: dê ao usuário só o acesso mínimo necessário;
- Auditoria: registrar tentativas de acesso negadas para monitoramento e análise.

7.1.5 Possíveis Impactos da Exploração

A exploração dessa vulnerabilidade compromete diretamente o princípio do menor privilégio, permitindo que usuários acessem recursos ou executem ações que deveriam estar fora do seu alcance. Isso pode resultar em vazamento de informações confidenciais, como dados pessoais, financeiros ou corporativos, além de possibilitar o roubo de identidades e credenciais sensíveis. Em casos mais graves, um invasor pode modificar ou excluir registros de outros usuários, assumir permissões administrativas (elevação de privilégio) ou acionar funcionalidades críticas do sistema que deveriam estar restritas a perfis autorizados. Esses cenários geram sérias consequências, incluindo a perda de integridade e confidencialidade dos dados, interrupção de serviços por uso indevido ou malicioso, e exposição da organização a riscos legais e financeiros significativos.

Conforme destacado pela *OWASP*, falhas de controle de acesso frequentemente resultam em divulgação não autorizada de informações, alterações indevidas ou destruição de dados, além da execução de operações fora dos limites definidos para o usuário, configurando uma violação grave da segurança da aplicação (*OWASP*, 2021).

7.2 *A02:2021 – Cryptographic Failures*

Anteriormente conhecido como Exposição de Dados Sensíveis, que é mais um sintoma amplo do que uma causa raiz, a falha criptográfica aborda a utilização de criptografias falhas ou até mesmo a completa ausência de uma criptografia, o que frequentemente leva à exposição de dados sensíveis.

Essa categoria ganhou ainda mais relevância com a promulgação de legislações como a Lei Geral de Proteção de Dados (*LGPD*) no Brasil e o Regulamento Geral sobre a Proteção de Dados (*GDPR*) na União Europeia, que exigem das organizações a adoção de medidas técnicas para proteger dados pessoais em trânsito e em repouso. A falha em aplicar criptografia adequada pode não apenas resultar em comprometimento de dados, mas também em sanções legais e perda de confiança por parte dos usuários.

7.2.1 Formas Comuns de Exploração

As falhas desta categoria abrangem diversos cenários técnicos. Um dos mais comuns é o uso de algoritmos criptográficos considerados inseguros ou obsoletos, como *MD5* e *SHA-1*, que podem ser facilmente quebrados por ataques de força bruta ou dicionário. Outro caso recorrente é o uso do modo de operação *ECB* com o algoritmo *AES*, que, por não introduzir aleatoriedade nos blocos cifrados, expõe padrões dos dados originais, comprometendo a confidencialidade. Além disso, é comum encontrar dados trafegando sem qualquer criptografia, como em conexões *HTTP* simples, ou armazenados em texto claro em bancos de dados, principalmente senhas e *tokens* de autenticação.

A má gestão de chaves também é uma causa frequente de falhas criptográficas. Isso inclui a reutilização de chaves, sua exposição em repositórios públicos ou o armazenamento em código-fonte, o que facilita o acesso por invasores. Um fato que corrobora essa afirmação é o crescente número de credenciais que são expostas anualmente em repositórios públicos do GitHub, os quais chegaram a 12,778,559 *tokens* expostos (GitGuardian, 2024).

Em muitos sistemas, senhas de usuários são armazenadas com funções de hash inadequadas, como *MD5* ou *SHA-1*, frequentemente sem o uso de um sal exclusivo ou com número insuficiente de iterações. Há ainda aplicações que implementam criptografia própria sem qualquer revisão formal ou validação por especialistas, o que tende a resultar em sistemas frágeis e facilmente quebráveis. Outro ponto crítico envolve a aceitação de certificados *TLS* inválidos ou a falta de verificação adequada da cadeia de confiança, abrindo espaço para ataques *man-in-the-middle*.

7.2.2 Casos Reais

Caso Yahoo (2013-2016): Em 2016/2017 a Yahoo revelou que sofreu duas invasões massivas, expondo ≈3 bilhões de contas de usuários (*CSO*, 2017). Entre os dados roubados estavam diversas senhas de conta, as quais foram armazenadas usando o *hash MD5* (sem *salt*), considerado fraco e suscetível a ataques, como o ataque de colisão (STOCKLEY, 2016). A descoberta só veio à tona anos depois, quando a empresa já estava em processo de venda. Como os dados foram armazenados usando um algoritmo extremamente fraco, os *hashes MD5* foram quebrados rapidamente pelos invasores. As consequências financeiras foram graves: o fechamento da aquisição da Yahoo pela Verizon sofreu abatimentos no valor, e a empresa detentora da marca, então conhecida como Altaba, foi multada em US$ 35 milhões pela *SEC* por omissão de informações sobre o vazamento (SEC, 2018).

Além disso, acordos judiciais ultrapassaram US$ 117 milhões em processos coletivos. O caso reforça lições como usar *hashes* de senha robustos (*bcrypt*, Argon2 etc.) com *salt*, criptografar dados sensíveis adequadamente e manter protocolos de resposta a incidentes, como notificar usuários e autoridades.

7.2.3 Exemplo de Vulnerabilidade

O exemplo a seguir, representado pela figura 3, apresenta duas falhas criptográficas: a utilização de *MD5* para armazenamento de senhas e o uso incorreto do *AES* no modo *ECB* para criptografar os dados do cartão. Ambas as práticas são altamente inseguras.

Figura 3 Código vulnerável A02

```ruby
class User < ApplicationRecord
  def store_sensitive_data(password, credit_card)
    self.encrypted_password = Digest::MD5.hexdigest(password)

    cipher = OpenSSL::Cipher::AES.new(128, :ECB)
    cipher.encrypt
    cipher.key = 'minhachavesecreta' # chave fixa e exposta
    encrypted = cipher.update(credit_card) + cipher.final

    self.encrypted_credit_card = Base64.encode64(encrypted)
  end
end
```

Fonte: Autores

Neste código, o uso do MD5 permite que as senhas sejam quebradas facilmente com ferramentas automatizadas ou *rainbow tables*. Além disso, o modo *ECB* do *AES* não utiliza vetor de inicialização, o que faz com que blocos idênticos de entrada produzam blocos idênticos de saída, revelando padrões dos dados originais e comprometendo a confidencialidade. A chave criptográfica também está *hardcoded* no código-fonte, outro erro grave que facilita o acesso ao conteúdo criptografado por qualquer indivíduo com acesso ao repositório da aplicação. Essas falhas, combinadas, comprometem severamente a segurança do sistema.

7.2.4 Recomendações de Mitigação

Um exemplo de correção seria a seguinte implementação:

Figura 4 Código vulnerável A02

```ruby
class User < ApplicationRecord
  has_secure_password

  def store_sensitive_data(password, credit_card)
    self.password = password

    cipher = OpenSSL::Cipher::AES256.new(:GCM)
    cipher.encrypt
    cipher.key = Rails.application.credentials[:encryption_key]
    iv = cipher.random_iv
    cipher.auth_data = "auth"

    encrypted = cipher.update(credit_card) + cipher.final

    self.encrypted_credit_card = Base64.strict_encode64({
      iv: iv, data: encrypted, tag: cipher.auth_tag
    }.to_json)
  end
end
```

Fonte: Autores

A correção proposta aumenta significativamente a segurança do armazenamento de senhas, já que utiliza uma criptografia múltiplas vezes mais forte, pelo uso de *has_secure_password,* o qual utiliza *bcrypt*, e remove o uso da chave fixa exposta no código para autenticação, optando por usar um parâmetro previamente configurado em um arquivo de credenciais.

7.2.5 Possíveis Impactos da Exploração

A exploração leva diretamente à perda de confidencialidade e integridade dos dados. Sem criptografia efetiva, invasores conseguem ler informações privadas, como dados pessoais, financeiros e credenciais, falsificar conteúdo ou interceptar. Isso facilita roubo de identidade, fraude financeira e exposição de propriedade intelectual.

No caso estudado do *Yahoo*, a criptografia ineficiente permitiu vazamentos em massa, gerando enormes prejuízos financeiros e perda de confiança pública. Falhas criptográficas também podem permitir que invasores criem dados forjados (assinaturas inválidas) ou executem código arbitrário se explorarem integrações fracas em protocolos seguros.

7.3 *A03:2021 – Injection*

As falhas de Injeção ocorrem quando dados não confiáveis são enviados a intérpretes (*SQL*, *NoSQL*, comandos do sistema etc.) como parte de consultas ou comandos, permitindo execução maliciosa (CODACY, 2024).

Essas falhas decorrem de *validação insuficiente de entrada* ou uso de construção dinâmica de consultas. O impacto é severo: vazamento de dados, corrupção de banco de dados ou execução de código arbitrário no servidor se tornam possíveis. Por exemplo, concatenar diretamente o *input* de usuário em uma *query SQL* pode permitir um ataque de injeção *SQL*, onde um parâmetro malicioso pode causar exposição de toda a tabela.

7.3.1 Formas Comuns de Exploração

Injeção *SQL* (em bancos de dados relacionais) é a mais difundida, onde comandos *SQL* arbitrários são inseridos em queries que não foram corretamente sanitizadas. Injeções *NoSQL*, por sua vez, são idênticas a injeções *SQL* à nível de impacto e resultado, porém tem como objetivo banco de dados não-relacionais.

Existem injeções que não são direcionados a bancos de dados, como por exemplo as injeções de comando para sistemas operacionais, que ocorrem quando entradas do usuário são passadas a comandos do sistema operacional, ou injeções de *cookies*, os quais ficam armazenados no navegador e corriqueiramente utilizados em diversas partes de uma aplicação *web.*

7.3.2 Casos Reais

Caso "*ResumeLooters*" (2023): Um grupo de invasores conhecido como *ResumeLooters* usou injeção *SQL* e *XSS* para comprometer 65 *sites* de empresas de recrutamento e varejo entre novembro e dezembro de 2022 (MONTALBANO, 2024). Ao explorar falhas de validação de parâmetros em consultas ao banco de dados, roubaram aproximadamente 2,08 milhões de registros únicos, o que incluía e-mails, nomes, telefones e histórico profissional de candidatos. O ataque ocorreu com ferramentas convencionais de *pentest*, apontando deficiências em filtragem de entradas. A resposta pública incluiu alertas de segurança e correção imediata dos pontos de injeção. Não houve publicação de valores de prejuízo financeiro, mas empresas afetadas recomendam revisão de códigos *SQL* e uso de *prepared statements* como forma de mitigação.

Caso *MOVEit Transfer* (2023): O *ransomware* Cl0p explorou uma vulnerabilidade de injeção *SQL*, no *software* de transferência de arquivos *MOVEit Transfer*. O bug já havia sido utilizado por *hackers* para roubar dados de mais de 2.700 organizações e expor cerca de 95,8 milhões de registros de funcionários e clientes (SIMAS, 2023). Entre as vítimas estão empresas como *Amazon*, *BBC*, *Boots* e *British Airways*; por exemplo, vazaram 2,8 milhões de

dados do quadro de funcionários da *Amazon* (SCROXTON, 2024). Esse incidente, que foi causado por um vetor de injeção em *software* terceirizado, causou prejuízos enormes, motivou cooperação internacional e acelerou a implantação de monitoramento de intrusões e políticas de *patch*. A mitigação incluiu aplicação imediata do *patch* de segurança em todos os servidores *MOVEit*, rotação de credenciais (para impedir uso de *backdoors*) e notificações a clientes sobre possíveis acessos não autorizados. As investigações e divulgações públicas levaram muitas organizações a reforçarem testes de intrusão e validar entradas em aplicações legadas.

7.3.3 Exemplo de Vulnerabilidade

O seguinte exemplo mostra uma clássica vulnerabilidade de injeção *SQL*, onde um parâmetro é usado diretamente dentro de uma consulta sem qualquer tipo de sanitização:

Figura 5 Código Vulnerável A03

```
1  class SearchController < ApplicationController
2    def search
3      query = params[:q]
4      @results = Article.where("title LIKE '%#{query}%'")
5    end
6  end
```

Fonte: Autores

Caso o usuário enviar algum parâmetro malicioso, como uma série de comandos seguidos por um símbolo de porcentagem, a consulta torna-se uma requisição que pode retornar todos os registros do banco de dados, expondo uma quantidade massiva de dados.

7.3.4 Recomendações de Mitigação

Existem diversas maneiras de corrigir e sanitizar entradas para que seja evitado tais vulnerabilidades. Uma correção extremamente simples do código vulnerável é apresentado na figura 6.

Figura 6 Código Corrigido A03

```
1  class SearchController < ApplicationController
2    def search
3      query = params[:q]
4      @results = Article.where("title LIKE ?", "%#{query}%")
5    end
6  end
```

Fonte: Autores

A utilização do método *where* com *placeholders* é uma prática essencial de segurança em *Rails*, pois garante o escape adequado dos *inputs*. O *ActiveRecord* realiza automaticamente a sanitização dos valores antes de incluí-los como parâmetros na consulta *SQL*, prevenindo que dados inseridos pelos usuários sejam interpretados como código malicioso. Essa abordagem protege contra-ataques de *SQL Injection*, onde um invasor poderia manipular a *query* para acessar dados não autorizados ou executar operações indevidas no banco de dados.

Além da sanitização automática, outras camadas de proteção podem ser implementadas para reforçar a segurança:

- Validação Positiva de Entrada (*Allowlist*): Consiste em definir explicitamente quais caracteres ou padrões são aceitáveis, rejeitando todo o resto. Por exemplo, para uma busca por título de artigos, o ideal permitir apenas letras, números, espaços e alguns caracteres acentuados, bloqueando símbolos especiais que não façam sentido no contexto;
- Limitação de Comprimento: Estabelecer um tamanho máximo razoável para o termo de busca previne ataques que tentam sobrecarregar o sistema com entradas extremamente longas;
- Normalização de Dados: Remover espaços extras, unificar caracteres similares (como diferentes tipos de aspas) e converter para um formato padrão antes do processamento;
- Controle de Taxa de Requisições (*Rate Limiting*): Limitar quantas buscas podem ser feitas em um determinado período por usuário, prevenindo abuso do sistema.

7.3.5 Possíveis Impactos da Exploração

A exploração de falhas de injeção representa uma das ameaças mais críticas à segurança de aplicações, uma vez que permite que um invasor interfira diretamente na lógica de execução de comandos da aplicação, geralmente através da manipulação de entradas fornecidas pelo usuário. Quando mal implementadas, essas entradas são interpretadas como comandos legítimos em contextos como consultas *SQL*, comandos de sistemas operacionais, expressões *LDAP*, instruções *XPath*, entre outros.

O impacto pode ser catastrófico: desde o acesso não autorizado a dados sensíveis, como credenciais e informações financeiras, até a alteração ou destruição de registros no

banco de. Em casos mais extremos, a falha pode ser explorada para execução remota de código (*RCE*), comprometendo completamente o sistema alvo.

Além disso, há o risco de escalonamento de privilégios, se comandos injetados permitirem a elevação de permissões dentro da aplicação ou infraestrutura. Tais vulnerabilidades afetam diretamente os pilares da segurança da informação e estão frequentemente associadas a incidentes de alta severidade, que podem resultar em multas regulatórias, danos reputacionais significativos, interrupção de serviços críticos e custos elevados com resposta a incidentes e remediação. A *OWASP* ressalta que, embora tradicional, a injeção continua altamente prevalente, especialmente em sistemas legados ou aplicações com validação de entrada insuficiente, sendo uma das causas principais de ataques complexos e automatizados na *web* moderna.

7.4 *A04:2021 Insecure Design*

O *Design* Inseguro refere-se a falhas em nível de arquitetura ou projeto que deixam lacunas de segurança não previstas. Ao contrário de erros de codificação pontuais, trata-se de controles necessários que nunca foram concebidos. Segundo a *OWASP*, esta nova categoria foca em "*risks related to design and architectural flaws*" (falhas de *design* e arquitetura) e recomenda maior uso de modelagem de ameaças e padrões seguros desde o início do desenvolvimento (*OWASP*, 2021).

Um mesmo sistema pode ser seguro por *design,* mas ainda assim ter bugs de implementação, porém um *design* inseguro não pode ser consertado apenas consertando código: faltam controles fundamentais que deveriam ter sido planejados anteriormente. Por exemplo, se não for feito *threat modeling*, requisitos de segurança podem ser subestimados e recursos críticos (credenciais, fluxo de negócio etc.) podem ficar expostos.

7.4.1 Formas Comuns de Exploração

Ataques sobre designs inseguros exploram diretamente omissões lógicas e fluxos de negócio mal planejados. Entre os vetores comuns estão:

- Ataques automatizados: A falta de defesas contra abuso automatizado (como *rate limiting* ou detecção de *bots*) permite ataques de força bruta e agendamento de massa. Por exemplo, se um *e-commerce* não implementar mecanismos adequados, *scalpers* podem automatizar compras de itens em estoque limitado. Essa ausência de controle *anti-automation* é típica de *design* inseguro (*OWASP*, 2021);

- Controles de autenticação fracos: Fluxos de *login* ou recuperação de conta mal arquitetados (perguntas-senha no *password recovery*, uso de *HTTP* sem *HTTPS*, falta de MFA) podem ser explorados para invasão de contas. O *OWASP* cita que esquemas de "perguntas e respostas" em recuperação de credenciais são proibidos justamente porque dois usuários podem conhecer as mesmas respostas e comprometer facilmente a conta. Isso representa uma exploração direta de *design* inseguro no fluxo de autenticação;
- Exposição de informações sensíveis: Mensagens de erro muito detalhadas ou dados sensíveis retornados pela *API* expõem sistema e baseiam outros ataques. Por exemplo, mostrar a *stacktrace* completa em caso de erro *SQL* pode revelar a estrutura do banco de dados e facilitar injeções.

7.4.2 Casos Reais

Botnet Mirai (2016): Um dos maiores incidentes envolvendo *design* inseguro ocorreu com a *botnet* Mirai, que explorou falhas de projeto em dispositivos *IoT*. Seus criadores publicaram o código-fonte em 2016, e o *malware* varreu a internet atrás de câmeras, roteadores e *DVRs* que ainda usavam usuário e senha padrão de fábrica. Se a combinação padrão não havia sido alterada, Mirai fazia *login* automaticamente e infectava o dispositivo (*CLOUDFLARE*, 2025). Com isso, aproximadamente cem mil dispositivos *IoT* foram sequestrados para realizar ataques de *DDoS* massivos. O evento culminou no ataque ao provedor *DNS* Dyn, derrubando grandes *sites* na América do Norte e Europa. O prejuízo foi enorme: indisponibilidade geral de serviços e desgaste da confiança em dispositivos *IoT*. Em resposta, os responsáveis foram identificados e detidos, mas muitas variantes continuaram a circular. A lição aprendida foi clara: nenhum dispositivo *IoT* deve deixar credenciais padrão, e arquiteturas de *IoT* precisam integrar padrões de segurança desde o *design*. Este episódio ilustra efeitos de disponibilidade (queda de serviços) e confidencialidade (*IoT* infectadas), reforçando que falhas de *design* podem ser exploradas em larga escala.

Exchange ProxyShell (2021): Outro exemplo foi a sequência de falhas conhecida como *ProxyShell* no Microsoft *Exchange*. Pesquisadores encadearam três *CVEs* para obter execução remota de código em servidores *Exchange* não atualizados. O diferencial é que essas falhas não eram simples bugs de implementação, mas "fraquezas fundamentais de *design*" no gerenciamento de autenticação e autorização do *Exchange* (SAID, 2025). Mesmo após aplicação de *patches*, o *design* permissivo permitia escalada de privilégios e execução

maliciosa de comandos no servidor. A Microsoft respondeu emergencialmente com updates de segurança em maio/julho de 2021, porém o incidente ensinou que a arquitetura de sistemas críticos deve prever múltiplas camadas de proteção (ex.: validação de *tokens*, *hardening* de serviços expostos) para evitar que vulnerabilidades lógicas provoquem ataques graves.

7.4.3 Exemplo de Vulnerabilidade

Um exemplo simples, ilustrado na figura 7, de *design* inseguro pode ser representado pelo seguinte código em *Ruby*, onde um sistema de desconto não verifica se o cupom já foi usado ou se ele está válido de alguma forma.

Figura 7 Código Vulnerável A04

```ruby
class PaymentsController < ApplicationController
  def apply_discount
    coupon = Coupon.find_by(code: params[:code])
    if coupon
      current_user.update(discount: coupon.value)
      redirect_to checkout_path, notice: "Desconto aplicado!"
    end
  end
end
```

Fonte: Autores

Tal fluxo pode fazer com que um cupom mal pensado seja usado por diversos usuários, infinitas vezes, causando um comportamento inesperado no valor da compra. Essa vulnerabilidade não se trata de um erro de código, e sim de um *design* fundamentalmente mal pensado em sua implementação.

7.4.4 Recomendações de Mitigação

Em código, figura 8, a correção do exemplo anterior pode ser uma simples verificação em relação ao uso prévio do cupom.

Figura 8 Código Corrigido A04

```ruby
class PaymentsController < ApplicationController
  def apply_discount
    coupon = Coupon.find_by(code: params[:code], used: false)
    if coupon
      coupon.update(used: true, user: current_user)
      current_user.update(discount: coupon.value)
      redirect_to checkout_path, notice: "Desconto aplicado!"
    end
  end
end
```

Fonte: Autores

Apesar de não ser uma correção ideal, já que seria necessário uma reanalise de todo o fluxo de cupom, verificando se os cupons não deveriam ter uma data de validade ou quantidade máxima de usos, a solução serve como uma correção pontual para que um cupom não seja usado infinitas vezes pelo mesmo usuário.

7.4.5 Possíveis Impactos da Exploração

A exploração de um *design* inseguro compromete diretamente os princípios básicos da segurança da informação. Quando um sistema é mal concebido desde sua arquitetura, a confidencialidade pode ser violada através de vazamentos de dados sensíveis causados por mecanismos de autenticação frágeis ou armazenamento impróprio de informações críticas. A integridade dos dados fica ameaçada quando falhas na lógica de negócio permitem que atacantes manipulem processos ou alterem registros além de suas permissões. Já a disponibilidade é colocada em risco quando a falta de controles contra abusos possibilita ataques que sobrecarregam os recursos do sistema, tornando serviços inacessíveis.

Além dos impactos técnicos, as consequências organizacionais são severas. Empresas enfrentam não apenas custos diretos com reparos e multas regulatórias, mas também sofrem danos reputacionais difíceis de recuperar. A perda de confiança dos clientes, somada a possíveis ações judiciais, transforma falhas de *design* em ameaças estratégicas. Um sistema mal projetado não cria apenas vulnerabilidades pontuais, ele estabelece brechas sistêmicas que podem ser exploradas de múltiplas formas. A segurança precisa ser incorporada desde as primeiras etapas de *design*, pois corrigir falhas estruturais posteriormente é sempre mais complexo e custoso do que as prevenir durante a concepção do sistema.

7.5 *A05:2021 Security Misconfiguration*

A configuração incorreta de segurança ocorre quando um sistema ou aplicação não está ajustado adequadamente para o ambiente de produção, deixando aberturas exploráveis). Atacantes exploram essas falhas escaneando portas abertas, acessando interfaces de administração não protegidas ou diretórios públicos, e colhendo informações de *debug* ou mensagens de erro detalhadas (*OWASP*, 2021). Por exemplo, a listagem de diretórios habilitada pode expor classes compiladas para engenharia reversa, ou mensagens de erro detalhadas podem revelar versões de componentes vulneráveis, como observou a *OWASP* ao mencionar *stack traces* sensíveis retornadas a usuários (*OWASP*, 2021).

Essa vulnerabilidade é frequentemente encontrada em nuvens públicas, onde permissões padrão excessivamente abertas em *buckets* ou máquinas virtuais permitem que dados sensíveis sejam acessados externamente com esforço diminuto.

7.5.1 Formas Comuns de Exploração

- Escaneamento de configuração pública: atacantes usam ferramentas, como Shodan, para encontrar instâncias com credenciais padrão ou sem autenticação. Um exemplo comum seria o acesso ao console de administração Jenkins sem *login*;
- Listagem de diretórios e arquivos expostos: servidores com *directory listing* ativado permitem o *download* de código fonte ou arquivos de configuração embarcados, facilitando descoberta de falhas internas;
- Mensagens de erro detalhadas: erros da aplicação excessivamente detalhados, quando exibidos para o usuário, podem revelar versões de bibliotecas ou chaves secretas, facilitando ataques subsequentes;
- Serviços e portas desnecessárias: deixar serviços de *debug*, administração ou perfis de gerenciamento ativos abre caminhos para exploração lateral;
- Recursos em nuvem públicos: *buckets* S3 abertos ou *VMs* com regras liberais permitem acesso a dados ou inserção de código malicioso.

7.5.2 Casos Reais

Caso Twilio (2020): Em julho de 2020, um *bucket AWS S3* desprotegido da Twilio permitiu que invasores injetassem código malicioso no *SDK JavaScript* da empresa. O incidente foi detectado pelo *UpGuard* e comunicado em 23 de julho de 2020 (SHERIDAN, 2020), ilustrando como configurações erradas de armazenamento em nuvem permitem alterações indesejadas no *software*.

Caso *NASA*/JIRA (2018): em 2018 descobriu-se que uma instância mal configurada do Atlassian JIRA utilizada pela *NASA* expôs dados internos (nomes de projetos, e-mails, senhas em *hash*). De acordo com pesquisador de segurança Avinash Jain, tal vulnerabilidade aconteceu graças a uma interpretação errônea do administrador de sistemas em relação a configuração de acesso "*all users*" e "*everyone*" (AFIFI-SABET, 2019). Segundo relato técnico, os dados foram acessíveis por semanas antes da correção em setembro de 2018.

7.5.3 Exemplo de Vulnerabilidade

No código *Ruby* da figura 9, é possível exemplificar um caso em que o erro é enviado por completo para o usuário final.

Figura 9 Código Vulnerável A05

```ruby
class UsersController < ApplicationController
  def show
    @user = User.find(params[:id])
    # ... processamento ...
  rescue StandardError => e
    render plain:
      "Erro no processamento:
        #{e.class} - #{e.message}\n#{e.backtrace.join("\n")}"
  end
end
```

Fonte: Autores

Nesse código Ruby, se ocorrer exceção, o *rescue* retorna diretamente detalhes internos (classe de erro, mensagem e *backtrace*) ao usuário, expondo informação sensível e versões de componentes. Essa falha de configuração (modo *debug*) é tipicamente usada por atacantes para identificar pontos fracos internos.

7.5.4 Recomendações de Mitigação

No trecho de código a seguir, mostrado por meio da figura 10, é mostrado uma possível maneira de resolver a vulnerabilidade previamente apresentada.

Figura 10 Código Corrigido A05

```ruby
class UsersController < ApplicationController
  def show
    @user = User.find(params[:id])
    # ... processamento ...
  rescue StandardError => e
    logger.error
      "Erro no processamento do usuário #{params[:id]}: #{e.message}"
    render status: 500, plain:
      "Ocorreu um erro ao processar sua solicitação."
  end
end
```

Fonte: Autores

A correção mostrada mitiga de forma prática a exibição de informações sensíveis no *logging* de erro, fazendo com que seja retornado um status de erro e um texto genérico de erro para o usuário, fazendo com que as informações importantes sejam logadas externamente.

As ações de segurança mais recomendadas para mitigação, são: desativar o modo de *debug* em produção (*config.consider_all_requests_local = false* no *Rails*); não expor informações de versão ou configuração em respostas *HTTP*; usar cabeçalhos de segurança

adequados (*X-Content-Type-Options, X-Frame-Options*); restringir acesso de rede com firewall (princípio de menor privilégio); sempre remover ou proteger credenciais padrão e realizar auditoria periódica de configurações (usando benchmarks *NIST*/*CIS*).

7.5.5 Possíveis Impactos da Exploração

A exploração de configurações incorretas de segurança pode levar a graves violações de dados e comprometimento de sistemas. Quando portas desnecessárias estão abertas ou interfaces administrativas não são protegidas, invasores ganham acesso não autorizado a informações sensíveis, como credenciais de banco de dados ou chaves de *API*.

Em ambientes de nuvem, permissões excessivamente permissivas em *buckets* de armazenamento ou instâncias virtuais frequentemente resultam em vazamentos em larga escala. Dados internos, backups não criptografados ou até mesmo código-fonte podem ser acessados publicamente, servindo como ponto de entrada para ataques mais avançados. Além disso, a listagem de diretórios ativada expõe estruturas de arquivos críticos, possibilitando engenharia reversa e descoberta de vulnerabilidades adicionais.

As consequências vão além do técnico: organizações enfrentam multas regulatórias por descumprimento de normas como *GDPR* ou *LGPD*, além de danos irreparáveis à reputação. Casos como vazamentos em *buckets* públicos da *AWS* demonstram como configurações negligentes podem custar milhões em prejuízos financeiros e judiciais. A ausência de *hardening* adequado transforma falhas simples em brechas catastróficas, exigindo não apenas correções reativas, mas a adoção de políticas contínuas de revisão e proteção de ambientes em produção.

7.6 *A06:2021 Vulnerable and Outdated Components*

Esta categoria trata do uso de bibliotecas, *frameworks*, *plugins* ou sistemas operacionais com vulnerabilidades conhecidas que não foram corrigidas. Componentes legados ou de terceiros (módulos Node.js, *gems Ruby*, *containers* etc.) em versões antigas podem conter falhas graves, permitindo exploração mesmo que o código em volta esteja correto. Atacantes identificam componentes desatualizados explorando CVEs publicados utilizando scanners automáticos, como *Dependency Check* e *Gemnasium*, ou plataformas de pesquisa como Shodan para localizar serviços vulneráveis.

Por exemplo, falhas conhecidas em bibliotecas populares (Apache Struts CVE-2017-5638, Drupal CVE-2018-7600, Apache *Log*4j CVE-2021-44228) são alvo de *scans* públicos, pois permitem execução remota de código em servidores não corrigidos

7.6.1 Formas Comuns de Exploração

Dentro de todo o *ranking OWASP*, essa vulnerabilidade é a mais facilmente explorada, já que existem diversas ferramentas automatizadas para o *scan* dessas brechas. Alguns exemplos, são:

- Exploração de vulnerabilidades conhecidas: uso de *exploits* ou *payloads* públicos para CVEs de componentes populares, como *scripts* que exploram vulnerabilidades do Apache Struts ou Log4j;
- Ferramentas de busca automática: atacantes empregam Shodan e outras plataformas para mapear versões vulneráveis em larga escala, encontrando serviços acessíveis na internet que não receberam *patches*, como dispositivos *IoT* ainda expostos ao *Heartbleed* de 2014;
- *Supply chain injection*: uso de componentes desatualizados pode incluir *backdoors* embutidos (intencionais ou não); por exemplo, um atacante compromete uma biblioteca popular para distribuir código malicioso em atualizações subsequentes;
- Dependências aninhadas: vulnerabilidades em dependências de dependências (que o desenvolvedor pode não conhecer) também são exploráveis, por isso a falta de inventário completo expõe o sistema.

7.6.2 Casos Reais

Caso Equifax (2017): A agência de crédito Equifax sofreu em 2017 uma das maiores brechas da história por falha em manter seus componentes atualizados. Os invasores exploraram uma vulnerabilidade crítica do Apache Struts (CVE-2017-5638), que já havia patch disponível, ganhando execução remota e roubando informações pessoais de ≈147 milhões de consumidores (EPIC, 2021). Isso resultou em custos superiores a US$ 1,4 bilhão em remediação e um acordo global de US$ 575 milhões com autoridades federais e estaduais (FTC, 2024). O caso ilustrava exatamente o risco de dependências desatualizadas: embora a infraestrutura de rede da Equifax estivesse em parte moderna, foi um componente legado não atualizado que abriu o caminho para a invasão. As lições incluíram obrigatoriedade de escanear automaticamente bibliotecas conhecidas, e atualizá-las imediatamente, e de práticas

rígidas de gerenciamento de ciclo de vida de software, além de planos de resposta para substituição urgente de componentes críticos.

Caso *Log4Shell* (2021): Outro exemplo de cadeia é a vulnerabilidade *Log4Shell* (Apache *Log*4j, CVE-2021-44228), descoberta em dezembro de 2021. Trata-se de uma falha *zero-day* em uma biblioteca de *logging* amplamente usada. Apesar de não estar ligada a um único vazamento público, o incidente levou centenas de organizações (de bancos a agências governamentais) a atualizarem emergencialmente seus sistemas. O potencial de exploração de *Log*4Shell (uso de JNDI para execução remota) demonstrou que depender de componentes desatualizados ou não auditados pode colocar em risco toda a cadeia de *software*. Como resposta, empresas reforçaram a prática de manter *Software Bill of Materials* (SBOM) e políticas rígidas de *patch* em bibliotecas de terceiros.

7.6.3 Exemplo de Vulnerabilidade

Como visto na figura 11, essa vulnerabilidade se trata de dependências desatualizadas, a melhor maneira de demonstrar é mostrando como uma *gem* pode estar sendo utilizada em uma versão insegura.

Figura 11 Código Vulnerável A06

```
1   gem 'nokogiri', '1.7.0'
2   require 'nokogiri'
3
4   html_input = params[:html]
5   doc = Nokogiri::HTML(html_input)
```

Fonte: Autores

Neste trecho, o código inclui explicitamente uma versão antiga da *gem* Nokogiri, uma biblioteca utilizada para analisar arquivos HTML e XML em *Ruby*, a qual é vulnerável, em vez de usar a última disponível. Esse componente desatualizado pode conter falhas (por exemplo, CVEs de *parsing*) que não existem nas versões correntes, permitindo que a simples manipulação de entradas HTML levante riscos extras ao processar.

7.6.4 Recomendações de Mitigação

No trecho de código a seguir, representado por meio da figura 12, é mostrado uma possível maneira de resolver a vulnerabilidade previamente apresentada:

Figura 12 Código Corrigido A06

```
1   gem 'nokogiri', '>= 1.11.0'
2   require 'nokogiri'
3
4   html_input = params[:html]
5   doc = Nokogiri::HTML.parse(html_input)
```

Fonte: Autores

A simples ação de garantir que a biblioteca utilizada esteja acima de uma versão possivelmente vulnerável já reduz consideravelmente os riscos de ser afetado por uma vulnerabilidade documentada. Muitas vezes isso não acontece por exigir refatorações em componentes onde essa biblioteca está sendo utilizada, gerando um trabalho constante de atualização de dependências.

Boas práticas incluem: manter inventário automático de dependências (Bundler, npm *audit*, *OWASP Dependency Check*) e atualizar regularmente todos os componentes para versões suportadas. Use repositórios oficiais ou *mirrors* internos para garantir integridade (preferindo pacotes assinados). Testar atualizações em ambiente controlado antes de *deploy*, e eliminar código ou *plugins* obsoletos. Se atualização não for imediata, considere *virtual patching* (WAF ou filtros de aplicação) para mitigar vulnerabilidades conhecidas.

7.6.5 Possíveis Impactos da Exploração

Componentes vulneráveis e desatualizados podem ser um vetor de ataque para qualquer outra vulnerabilidade. Por isso, o impacto pode variar muito dependendo da vulnerabilidade que está presente nesse componente desatualizado. Como por exemplo:

- Execução remota de código: componentes com *RCE* conhecido permitem que invasores executem comandos arbitrários no servidor, como no caso do Struts usado na Equifax (EQUIFAX, 2017);
- Comprometimento em larga escala: um componente common (usado por muitos sistemas) vulnerável pode afetar múltiplos serviços simultaneamente (ex.: *IoT* desatualizados expostos globalmente, como dispositivos com Heartbleed);
- Negação de serviço: vulnerabilidades em servidores *web* ou servidores de aplicação, como por exemplo *buffer overflows* já documentados, podem levar à queda dos serviços.

7.7 *A07:2021 Identification and Authentication Failures*

Falhas de identificação e autenticação ocorrem quando os mecanismos que verificam identidade e protegem sessões são insuficientes. Isso inclui ausência de *MFA*, políticas de senha fracas, gerenciamento de sessão inadequado (*tokens* previsíveis ou expiração incorreta). Atacantes exploram essas vulnerabilidades usando ataques automatizados: técnicas comuns são credential stuffing (testar listas de senhas vazadas em massa), força bruta (tentar combinações repetidas) e *phishing*/*SMS-swap* para obter segundas-fatores. Conforme descrito pelo *OWASP*, permitir senhas fracas, padrões conhecidos ("*admin*/*admin*") ou não limitar tentativas de *login* configurações abre espaço para invasores

7.7.1 Formas Comuns de Exploração

As explorações que envolvem falhas de autenticação e identificação envolvem outras falhas, já que normalmente se tornam uma vertente para explorações mais críticas. Formas comuns envolvem:

- *Credential stuffing*: reutilização de credenciais válidas obtidas em outro serviço. Por exemplo, se um serviço sofreria um vazamento externo, seus dados podem ser testados contra outras aplicações;
- Ataques de força bruta: *scripts* que repetidamente tentam *logins* combinando usuário/senha. Se não houver limitação de tentativas, senhas curtas ou sem bloqueio tornam isso trivial;
- *Bypass* de autenticação: falta de verificação de sessão/pulso (ex.: fixação de sessão, *URL* de *token* ou ausência de *MFA*) permite que atacantes reutilizem *tokens* roubados;
- Uso de credenciais padrão/rotas de recuperação fracas: caso ainda existam contas de administrador com senha padrão, ou sistemas que liberam resets via informações de fácil resposta, invasores podem obter acesso sem muita resistência;
- Ataques à confirmação de identidade: interceptação de códigos *OTP* via *SIM swap* ou falha em revogar *tokens* antigos permite uso indevido de sessões ativas.

7.7.2 Casos Reais

Caso *Change Healthcare* (2024): Em fevereiro de 2024, a subsidiária da *UnitedHealth Group*, *Change Healthcare*, sofreu ataques de *ransomware BlackCat*/*ALPHV* que paralisou o setor de saúde dos *EUA*. Investigadores concluíram que invasores

aproveitaram credenciais comprometidas em um aplicativo de acesso remoto que não exigia autenticação multifator (LANGLEY, 2024). A ausência de *MFA* permitiu mover-se lateralmente pela rede da empresa; dados críticos (PHI e PII) foram exfiltrados de centros de dados, levando a atrasos em tratamentos e abastecimento de medicamentos. O grupo de resgate divulgou que recebera US$22 milhões em bitcoin após o incidente. Não há números públicos exatos de registros afetados, mas sabe-se que a operação resultou em investigações federais (*HHS*) e debates sobre padronização de segurança na saúde. A lição principal foi reforçar que *MFA* e monitoramento contínuo em todos os pontos de autenticação, especialmente em fornecedores críticos, são imprescindíveis para evitar que um único *login* violado comprometa todo o sistema

Caso *Capital One* (2019): Em julho de 2019, a *Capital One* reportou que invasores obtiveram dados de 106 milhões de clientes, 100 milhões de pessoas nos *EUA* e 6 milhões no Canadá (SCHROEDER, 2020). O ataque explorou uma configuração errada de *firewall WAF* na infraestrutura *AWS* da empresa, permitindo um tipo de *Server-Side Request Forgery* (*SSRF*)/*SSRF-like* que expôs metadados de instâncias de nuvem. Embora sem mau uso direto de credenciais (não foi um *phishing* clássico), o episódio destacou falha de gestão de credenciais na nuvem e falta de controles de acesso avançados. Na resposta, a *Capital One* bloqueou imediatamente o ponto de entrada, rotacionou credenciais *AWS* e cooperou com o *FBI*. Consequências financeiras: multa de US$ 80 milhões pelo órgão regulador bancário *OCC* e um acordo coletivo de US$ 190 milhões com vítimas em 2022 (AIJAZ, 2025). Isso ilustra que falhas na autenticação/autorização (mesmo em componentes de terceiros) podem gerar enormes litígios e exigir investimentos pesados em segurança de identidade (senhas fortes, *MFA* e revisão de configurações).

7.7.3 Exemplo de Vulnerabilidade

No código a seguir, representado pela figura 13, é mostrado um código extremamente vulnerável que pode ser facilmente explorado utilizando técnicas diversas, como de força bruta:

Figura 13 Código Vulnerável A07

```ruby
class SessionsController < ApplicationController
  def create
    user = User.find_by(email: params[:email])
    if user && user.authenticate(params[:password])
      session[:user_id] = user.id
      redirect_to dashboard_path
    else
      render plain: "Falha de login"
    end
  end
end
```

Fonte: Autores

Nesse código, qualquer usuário que adivinhe (ou saiba) uma combinação válida de e-mail e senha é autenticado sem verificação adicional. Não há contador de tentativas nem exigência de *token* extra, expondo o sistema a ataques automatizados e uso indevido de senhas vazadas.

7.7.4 Recomendações de Mitigação

O código a seguir, representado pela figura 14, mostra uma possível correção para o código vulnerável mostrado anteriormente:

Figura 14 Código Corrigido A07

```ruby
class SessionsController < ApplicationController
  def create
    user = User.find_by(email: params[:email])
    if user
      if user.failed_attempts >= 5
        render plain: "Conta bloqueada por muitas tentativas"
        return
      end
      if user.authenticate(params[:password])
        user.update(failed_attempts: 0)
        if user.admin?
          if params[:otp_code].present?
            && verify_totp(user, params[:otp_code])
            session[:user_id] = user.id
          else
            render plain: "Segundo fator inválido" and return
          end
        else
          session[:user_id] = user.id
        end
      else
        user.increment!(:failed_attempts)
        render plain: "Credenciais inválidas"
      end
    else
      render plain: "Credenciais inválidas"
    end
  end
```

Fonte: Autores

No código corrigido, foi criado um contador para que as tentativas de *logins* sejam contabilizadas, e caso chegue a um máximo de 5 tentativas, a conta é automaticamente bloqueada, evitando ataques de *brute force.* Outra correção relevante foi a pseudoinplementação de autenticação multifator por meio de um OTP.

Boas práticas incluem: habilitar lockout progressivo ou CAPTCHA após repetidas falhas de *login*, e exigir MFA para contas críticas. Exigir senhas fortes e verificar listas de senhas fracas. Não expor IDs de sessão em URLs e incentivar o uso de geradores de sessão criptograficamente seguros. Registrar todas as tentativas de autenticação e alertar administradores de padrões anômalos.

7.7.5 Possíveis Impactos da Exploração

A falha em mecanismos de identificação e autenticação permite que invasores comprometam contas de usuários e administradores, obtendo acesso não apenas a dados pessoais, mas também a informações privilegiadas que deveriam estar restritas. Uma vez dentro do sistema, os atacantes podem manipular configurações críticas, executar comandos internos ou até mesmo instalar *malware*, comprometendo a integridade de toda a infraestrutura. Em serviços financeiros, esse tipo de brecha abre caminho para fraudes em larga escala, como transferências ilegítimas ou aprovação de transações fraudulentas. Além disso, a falta de monitoramento contínuo permite que atividades maliciosas passem despercebidas por longos períodos, ampliando significativamente os danos causados.

Os casos reais que foram expostos mostram como falhas de autenticação não apenas violam a segurança imediata, mas também facilitam ataques persistentes e de alto impacto, afetando operações críticas e resultando em prejuízos financeiros e reputacionais substanciais.

7.8 *A08:2021 Software and Data Integrity Failures*

Essa categoria abrange falhas na integridade do *software* e dos dados, principalmente em cadeias de suprimentos de *software*. Envolve confiar em atualizações, pacotes ou dados sem verificar sua proveniência ou integridade. Isso inclui falhas como não validar assinaturas de atualização, permitir inserção de código em pipelines de CI/CD, ou aplicações que buscam recursos externos (*plugins*, firmas) sem proteção. Ataques típicos incluem injeção de código malicioso em processos de atualização e exploração de bibliotecas modificadas (*typosquatting*, ou interceptação de downloads).

7.8.1 Formas Comuns de Exploração

- Atualizações não assinadas: atualizar *software* sem checar assinatura digital permite que atacantes coloquem versões maliciosas (roteadores domésticos, firmware de *IoT*);
- Injeção na cadeia de suprimentos: invasores comprometem repositórios ou *pipelines* (*GitHub*, *npm*, *PyPI*), inserindo código malicioso em componentes amplamente usados;
- Deserialização insegura: confiando em dados serializados de fontes não confiáveis (ex.: deserializar *JSON*/*XML* sem verificação) pode levar à execução de código arbitrário;
- Ferramentas de automação sem verificações: *scripts* internos que baixam e executam código (via *curl* | *bash* sem validação) dão poder ao atacante de controlar o fluxo de código no servidor.

7.8.2 Casos Reais

Caso SolarWinds (2020): O ataque *SolarWinds* é o exemplo mais emblemático de falha de integridade de *software*. Hackers infiltraram um *backdoor* ("*Sunburst*") numa atualização legítima do produto Orion da *SolarWinds*, utilizado por milhares de organizações. Mais de 18.000 clientes baixaram o update malicioso (FORTINET, 2023), incluindo agências federais (*FBI*, *CISA*, tesouro etc.) e empresas como Microsoft. Com acesso privilegiado via Orion, os atacantes conseguiram espionagem e roubo de dados de redes internas. O incidente foi descoberto em dezembro de 2020, desencadeando resposta global. Na sequência, a *SEC*, em outubro de 2023, acusou a *SolarWinds* de fraude ao superestimar suas práticas de segurança (SEC, 2023) e obrigou mudanças executivas. No setor, adotou-se controle de integridade via assinaturas digitais (ganchos de build seguros) e análise automática de código de fornecedores, reforçando o conceito de cadeia de confiança. Financeiramente, a *SolarWinds* sofreu forte queda de valor em bolsa (≈25% em dois dias) e está sob litígios civis/financeiros, evidenciando que ataques à cadeia de suprimentos podem ser mais devastadores que brechas diretas.

Caso Codecov (2021): Em abril de 2021 a *Codecov* divulgou que invasores modificaram seu "*bash uploader*" (usado em *pipelines* de *CI*) para roubar chaves de ambiente (*IAM*, *tokens*, segredos) de sistemas que executavam o *script*. A adulteração mostrou como uma ferramenta genuína pode se tornar um vetor de roubo em massa de credenciais.

7.8.3 Exemplo de Vulnerabilidade

No código a seguir, representado pela figura 15, é mostrado um exemplo onde não há preocupação com a integridade dos dados utilizados:

Figura 15 Código Vulnerável A08

```ruby
require 'net/http'
uri = URI("http://atualizacoes.exemplo.com/upgrade.rb")
res = Net::HTTP.get(uri)
eval(res)
```

Fonte: Autores

Nesse código, o aplicativo baixa dinamicamente um *script Ruby* pela internet e o executa sem verificação. Se o servidor remoto for comprometido, qualquer código malicioso hospedado será executado localmente, com impacto potencialmente crítico.

7.8.4 Recomendações de Mitigação

No código a seguir, representado pela figura 16, é apresentado um exemplo de correção para que haja uma validação segura dos dados utilizados:

Figura 16 Código Corrigido A08

```ruby
require 'net/http'
require 'openssl'
uri = URI("https://atualizacoes.exemplo.com/upgrade.rb")
res = Net::HTTP.get(uri)

expected_hash = "ab34...cdef"
if Digest::SHA256.hexdigest(res) == expected_hash
  eval(res)
else
  raise "Falha na verificação de integridade do código!"
end
```

Fonte: Autores

Na correção proposta na figura 16, foi utilizado uma verificação de *hash*, garantindo que a informação não foi alterada durante transição, o que proporciona uma validação de que pelo menos o artigo chegou integro em seu destino

Boas práticas incluem:

- Sempre obter *software* de fontes oficiais e confiáveis (preferencialmente pacotes assinados);
- Implementar assinaturas digitais (*PCKS*, *GPG*) nas atualizações e verificar no cliente antes de aplicar;

- Em pipelines *CI/CD*, restringir acesso e habilitar aprovação manual de mudanças críticas;
- Não permitir deserialização de objetos de origens não confiáveis; se necessário, usar formatos seguros (*JSON* puro ou bibliotecas de análise que imponham *whitelist*);
- Ferramentas de segurança (*OWASP* Dependency Check e *software bill of materials*) ajudam a garantir que apenas componentes verificados são usados.

7.8.5 Possíveis Impactos da Exploração

A exploração de falhas na integridade de *software* e dados pode ter consequências devastadoras em escala global. Quando atacantes conseguem injetar código malicioso em sistemas centrais, como servidores de atualização, o comprometimento se propaga rapidamente, como demonstrado no ataque à *SolarWinds*, onde uma única alteração maliciosa afetou milhares de organizações simultaneamente. Esse tipo de invasão não só causa danos imediatos, mas também permite a instalação de *backdoors* persistentes, que mantêm acesso furtivo aos sistemas por longos períodos, facilitando espionagem contínua e coleta sigilosa de informações. Além disso, *scripts* maliciosos embutidos em processos legítimos podem extrair credenciais privilegiadas, chaves de criptografia ou certificados digitais, como ocorreu no incidente da *Codecov*, onde invasores roubaram credenciais da *AWS* através de alterações no ambiente de integração contínua.

Os impactos vão além do técnico: a violação da cadeia de suprimentos de *software* mina a confiança de clientes e parceiros, gerando crises reputacionais difíceis de reparar. Organizações que falham em implementar verificações rigorosas de integridade podem enfrentar penalidades legais severas, especialmente em setores regulamentados. A combinação de danos operacionais, perda de propriedade intelectual e responsabilização jurídica transformam essas falhas em ameaças estratégicas, exigindo não apenas respostas a incidentes, mas a adoção de controles proativos como assinaturas digitais, verificações de *checksum* e monitoramento contínuo de alterações não autorizadas em *pipelines* de desenvolvimento e distribuição.

7.9 *A09:2021 Security Logging and Monitoring Failures*

Essa categoria refere-se à ausência ou deficiência de *logs* e alertas sobre eventos de segurança. Aplicações que não registram adequadamente *logins*, transações críticas ou erros

de autenticação deixam atividades maliciosas invisíveis. Sem monitoramento eficaz (SIEM ou alertas em tempo real), uma invasão pode persistir sem detecção. O *OWASP* destaca falhas como não logar eventos auditáveis (*login*, ações críticas) e não manter *logs* fora do sistema principal. Também enfatiza que *logs* devem ser integrados a sistemas de correlação e segurados contra adulteração (armazenamento *append-only*, criptografia) para manter a integridade dos registros

7.9.1 Formas Comuns de Exploração

A falta de *logs* de segurança e falha de monitoramento amplifica a gravidade de outras vulnerabilidades, já que praticamente garante que invasores dificilmente serão identificados, e facilitando a persistência de um invasor no sistema. Exemplos são:

- Passagem sem rastros: ao invadir um sistema sem que haja registro de *login* bem-sucedido, o atacante caminha livremente (por exemplo, se a aplicação não loga acessos válidos, é impossível saber quando um intruso entrou);
- Ocultação de ações: destruição seletiva de *logs* ou utilização de canais não registrados impede auditoria. Se erros de autorização não geram alerta, um ataque de escalonamento pode passar despercebido;
- Falta de alertas em falhas de segurança: se não há notificação automática de padrões anômalos (múltiplas tentativas de *login*, requisições incomuns), ataques volumétricos continuam sem limitação;
- Armazenamento local de *logs*: guardar *logs* somente no servidor atacado permite que invasores apaguem evidências; sem replicação ou *logging* externo, a visibilidade é nula.

7.9.2 Casos Reais

Caso Marriott/Starwood (2018): A invasão ao banco de dados da rede Starwood (adquirida pela Marriott) teve uma longa janela de detecção. O ataque começou em 2014, mas só foi identificado em 2018 graças a uma ferramenta interna de segurança que foi implementado apenas após aquisição da empresa (FRUHLINGER, 2020). Estima-se que até 500 milhões de registros de hóspedes foram roubados (passaportes, cartões de crédito etc.). A apuração posterior pela ICO (regulador do Reino Unido) enfatizou que a Marriott falhou em monitorar contas privilegiadas e bancos de dados adequadamente (HUTTON, 2020), atrasando a resposta. A multa aplicada foi de £18,4 milhões por violar o GDPR (considerando

apenas o período pós-maio de 2018). Esse caso exemplifica como a falta de *logs* efetivos e alerta em tempo real permite que um invasor permaneça por anos sem ser notado. Após o incidente, o grupo adotou sistemas de SIEM aprimorados e roteiros de resposta automatizada para detecção de anomalias em bancos de dados legados.

Caso Equifax (2017): Caso anteriormente citado anteriormente, porém, além da utilização de componentes desatualizados, observou-se que a Equifax demorou semanas para reconhecer a extensão do *breach*. A empresa só comunicou o vazamento ao público 6 semanas depois da invasão ter sido confirmada internamente (EPIC, 2021). Na sequência, reforçou-se a noção de "*security by design*": empresas passaram a investir em registro de eventos (*logs*) e monitoramento contínuo (SOAR, UEBA) para detecção precoce, sob pena de responsabilidade regulatória e penal, como destacado pelo SEC no caso.

7.9.3 Exemplo de Vulnerabilidade

No código a seguir, representado pela figura 17, é mostrado um exemplo simples que envolve uma lógica de *login*:

Figura 17 Código Vulnerável A09

```ruby
class SessionsController < ApplicationController
  def create
    user = User.find_by(email: params[:email])
    if user&.authenticate(params[:password])
      session[:user_id] = user.id
      redirect_to root_path
    else
      render plain: "Falha de login"
    end
  end
end
```

Fonte: Autores

Neste código, sucessos e falhas de *login* não são registrados em lugar algum. Essa omissão impede que administradores saibam quem acessa ou tenta acessar a aplicação, tornando impossível detectar invasões por análise de *log*.

7.9.4 Recomendações de Mitigação

No código a seguir, representado pela figura 18, é exemplificado uma possível correção para a vulnerabilidade:

Figura 18 Código Corrigido A09

```ruby
class SessionsController < ApplicationController
  def create
    user = User.find_by(email: params[:email])
    if user&.authenticate(params[:password])
      session[:user_id] = user.id
      logger.info
        "Login realizado: usuário=#{user.id} email=#{user.email}"
      redirect_to root_path
    else
      logger.warn
        "Falha de login: email=#{params[:email]} IP=#{request.remote_ip}"
      render plain: "Credenciais inválidas"
    end
  end
end
```

Fonte: Autores

Na correção, foi atribuído um *logger* para que o registro de falhas e acessos bem-sucedidos sejam devidamente armazenados para análise posterior.

Para garantir uma estratégia eficaz de *logs* e monitoramento, recomenda-se implementar registros detalhados de todas as tentativas de autenticação, incluindo *logins* bem-sucedidos e malsucedidos, com informações contextuais como identificação do usuário, endereço IP e *timestamp*. Esses registros devem ser armazenados em sistemas externos e imutáveis para prevenir adulteração, utilizando mecanismos como *write-once* ou criptografia para garantir sua integridade.

Além disso, é essencial monitorar operações sensíveis, como alterações de credenciais, transações financeiras e modificações em configurações críticas, tanto em nível de aplicação quanto de infraestrutura. A implementação de ferramentas de correlação e detecção em tempo real (SIEM/IDS) permite identificar padrões suspeitos automaticamente, como múltiplas falhas de autenticação em curto período ou acessos fora do horário comercial.

A retenção dos *logs* por um período adequado às necessidades operacionais e regulatórias possibilita análises forenses e a investigação de incidentes, enquanto alertas configurados corretamente aceleram a resposta a potenciais violações. Essa abordagem não apenas melhora a visibilidade sobre a segurança dos sistemas, mas também apoia a conformidade com frameworks e regulamentações de proteção de dados.

7.9.5 Possíveis Impactos da Exploração

A ausência de sistemas adequados de registro e monitoramento permite que invasores operem livremente por longos períodos, como demonstrado em diversos casos de ataques persistentes que permaneceram ativos por meses ou até anos antes da detecção (ANDERSON, 2023). Esse tempo prolongado de acesso não autorizado amplifica

significativamente os danos, permitindo a exfiltração contínua de dados sensíveis e a escalada de privilégios dentro do ambiente comprometido.

A falta de *logs* detalhados e imutáveis cria uma grave lacuna de *accountability*, impossibilitando a reconstrução precisa de atividades maliciosas e dificultando a identificação dos responsáveis por violações. Essa situação compromete não apenas a resposta a incidentes, mas também processos judiciais ou investigações regulatórias. Organizações que negligenciam esses controles também enfrentam riscos de conformidade, já que regulamentações como LGPD e GDPR exigem explicitamente o registro de acessos a dados pessoais. A incapacidade de demonstrar esses registros durante auditorias pode resultar em penalidades financeiras expressivas, além de exigências adicionais de remedição.

Por fim, os danos à reputação são inevitáveis quando violações são descobertas por terceiros ou tornadas públicas tardiamente. A combinação de todos os fatores apresentados transforma falhas de monitoramento em riscos estratégicos, com impactos operacionais, financeiros e reputacionais de longo prazo.

7.10 *A10:2021 Server-Side Request Forgery*

SSRF ocorre quando um aplicativo aceita uma *URL* de entrada do usuário e faz uma requisição *HTTP* sem validação adequada, permitindo que o atacante direcione o servidor para recursos não intencionais. Isso possibilita que invasores acessem serviços internos não expostos publicamente (*firewalls*, redes internas) ou serviços de *metadata* em nuvens (para obter credenciais *IAM*). Por exemplo, explorando *SSRF* em uma instância *AWS*, um atacante pode consultar a *URL* de *metadata* e obter chaves temporárias sem permissão. Com isso, é possível burlar firewalls: o servidor se torna intermediário em ataques contra seu próprio ambiente.

7.10.1 Formas Comuns de Exploração

A exploração de *SSRF* ocorre principalmente de quatro maneiras, cada uma com impactos significativos:

- Varredura de rede interna: Ao injetar *URLs* internas, o atacante força o servidor a escanear sua própria rede. Com isso, respostas lentas, erros específicos ou diferenças no comportamento revelam portas abertas e serviços ativos, mapeando alvos para ataques futuros;

- Exfiltração de dados internos: Requisições a *URIs* ou *APIs* locais não protegidas (painéis de administração, bancos de dados internos) permitem vazar credenciais, configurações e outros dados sensíveis. Arquivos como "*.env"*, chaves *SSH* e tokens de acesso são alvos frequentes;
- Acesso a serviços de *metadata* em nuvem: Provedores como *AWS* e *GCP* usam *endpoints* especiais para gerenciar configurações. Um *SSRF* bem-sucedido pode roubar tokens de *IAM*, chaves de *API* e outros segredos, escalando para comprometer toda a infraestrutura em nuvem;
- *Bypass* de restrições de segurança: Sistemas que bloqueiam acesso externo, mas confiam em requisições internas, podem ser enganados. No caso da *Capital One,* um *SSRF* burlou um *WAF*, acessando metadados da *AWS* e resultando em um vazamento massivo. Técnicas como redirecionamentos maliciosos e abuso de *proxies* internos também são comuns.

7.10.2 Casos Reais

Caso *Capital One* (2019): No caso já citado anteriormente, a *Capital One* viveu um ataque clássico de *SSRF* em 2019. Invasores exploraram uma configuração incorreta em um firewall *WAF* na *AWS*, permitindo que um serviço *web* interno acessasse o serviço de metadados da instância na nuvem (um típico vetor de *SSRF*). Este incidente ilustra a gravidade do *SSRF*: trata-se de um bug de autorização interna que, embora silencioso, resultou em grandes vazamentos. As lições incluem reforçar os controles de acesso de redes internas (bloquear metadados de contêineres) e testar aplicações para SSRF em segurança de nuvem.

Caso Alibaba *Cloud* (2021): Em 2021, uma vulnerabilidade crítica de *SSRF* foi descoberta na plataforma de *big data* da Alibaba *Cloud*, revelando uma fragilidade em como serviços *cloud* processam requisições não confiáveis. O problema ocorria em um componente interno que aceitava *URLs* fornecidas por usuários sem a devida validação. Ao injetar o *endpoint* de metadados da Alibaba *Cloud*, atacantes conseguiam enganar o servidor, fazendo-o vazar *tokens* de acesso temporários, credenciais de instâncias e outras informações sensíveis do ambiente *cloud*. O impacto foi significativo: milhares de clientes corporativos tiveram seus metadados expostos, incluindo configurações de rede e chaves de *API*. Isso abriu caminho para potenciais escalações de privilégio dentro de ambientes *cloud* comprometidos, colocando em risco dados críticos de negócios. A Alibaba *Cloud* agiu rapidamente para corrigir a falha e

notificar clientes afetados, mas o incidente ainda resultou em repercussões regulatórias sob as rígidas leis chinesas de proteção de dados, embora os valores exatos de multas não tenham sido divulgados publicamente.

7.10.3 Exemplo de Vulnerabilidade

Para exemplificar uma vulnerabilidade *SSRF,* será utilizado o trecho de código representado pela figura 19:

Figura 19 Código Vulnerável A10

```ruby
require 'net/http'
image_url = params[:url]  # URL fornecida pelo usuário
image_data = Net::HTTP.get(URI(image_url))
send_data image_data, type: 'image/png', disposition: 'inline'
```

Fonte: Autores

Neste código Ruby, o servidor obtém dados de uma URL arbitrária fornecida pelo usuário. Um invasor pode inserir qualquer URL maliciosa, como uma URL que aponta para um endereço interno, para que o servidor busque as credenciais de IAM da instância AWS, expondo informações sensíveis. Ou usar *localhost* para atingir serviços internos.

7.10.4 Recomendações de Mitigação

Para demonstrar uma possível correção para a vulnerabilidade descrita, será utilizada o exemplo de código representado pela figura 20.

Figura 20 Código Corrigido A10

```ruby
allowed_hosts = ["imagens.example.com", "assets.example.com"]
uri = URI(params[:url])
if uri.host.in?(allowed_hosts)
  image_data = Net::HTTP.get(uri)
  send_data(image_data, type: 'image/png', disposition: 'inline')
else
  render plain: "URL não autorizada", status: :bad_request
end
```

Fonte: Autores

No exemplo corrigido, existe uma checagem explicita de quais *hosts* são permitidos na entrada do usuário, invalidando endereços internos ou renderizações de sites externos dentro da aplicação.

Em suma, para uma boa proteção contra *SSRF*, o primeiro passo é implementar *whitelists* restritivas para domínios, permitindo apenas *HTTP*/*HTTPS* e bloqueando esquemas perigosos. Desative redirecionamentos automáticos e normalize todas as *URLs* para detectar ofuscação. Em ambientes *cloud*, proteja *endpoints* de metadados exigindo cabeçalhos específicos e políticas *IAM* restritivas.

Serviços internos devem ficar em redes separadas, longe de aplicações *web* públicas. Essa combinação de validação na aplicação e proteção na infraestrutura forma uma barreira eficaz contra *SSRF*, reduzindo riscos de vazamentos e acessos não autorizados a sistemas sensíveis. Aplicar essas medidas de forma consistente é crucial para segurança em ambientes modernos.

7.10.5 Possíveis Impactos da Exploração

A exploração de vulnerabilidades *SSRF* pode resultar em consequências graves e de amplo espectro para organizações. Um dos impactos mais imediatos é o roubo de credenciais privilegiadas, especialmente em ambientes de nuvem, onde a obtenção de tokens de metadados pode conceder aos atacantes controle total sobre contas inteiras, como ocorreu no conhecido caso da *Capital One*. Esse tipo de comprometimento permite não apenas o acesso não autorizado, mas também a exposição de dados internos críticos, incluindo informações armazenadas em *APIs* administrativas, bancos de dados de configuração e arquivos sensíveis do sistema, como */etc/passwd* em servidores Linux, colocando em risco segredos corporativos e dados pessoais protegidos por regulamentações.

Além disso, os invasores podem utilizar o acesso inicial obtido via *SSRF* para realizar movimentação lateral na rede, contornando firewalls e outros mecanismos de segurança perimetral. Essa capacidade de se espalhar pela infraestrutura permite a execução remota de código, a instalação de *backdoors* persistentes e até mesmo a realização de ataques de negação de serviço contra componentes críticos. A desestabilização da infraestrutura é outro risco significativo, já que varreduras intensivas ou explorações maliciosas podem sobrecarregar serviços internos, causando indisponibilidade e interrupções operacionais.

A dificuldade de detecção agrava esses impactos, pois muitas explorações de *SSRF* ocorrem de forma silenciosa, sem gerar alertas imediatos. O tempo entre a exploração inicial e sua descoberta pode permitir que os danos se amplifiquem, enquanto os custos de remediação, incluindo investigações forenses, notificações a clientes e multas regulatórias, representam um fardo adicional para as organizações. Em conjunto, esses fatores transformam o *SSRF* em

uma ameaça multidimensional, capaz de comprometer simultaneamente a confidencialidade, integridade e disponibilidade dos sistemas, com repercussões que vão desde perdas financeiras até danos irreparáveis à reputação.

8 Resultados e Discussões

Este trabalho se consolida como uma referência fundamental para a compreensão das vulnerabilidades *web* críticas apresentadas no *OWASP Top* 10, oferecendo aos desenvolvedores e profissionais de segurança da informação um guia prático para identificação, prevenção e mitigação de riscos no desenvolvimento de *software*. Os resultados obtidos demonstram que, mais do que simplesmente listar vulnerabilidades, é essencial compreender os mecanismos subjacentes a cada falha e seu potencial impacto nos sistemas.

A análise realizada revela que a correção efetiva das vulnerabilidades vai além da aplicação técnica de *patches*, exige uma mudança de paradigma no processo de desenvolvimento. O estudo evidencia que:

- A maioria das vulnerabilidades surge de práticas inadequadas na fase de projeto (*Insecure Design*);
- Falhas de implementação básicas continuam prevalecendo mesmo em aplicações modernas;
- O custo da remediação pós-implantação é significativamente maior do que a prevenção durante o desenvolvimento.

A relevância deste trabalho se amplifica no contexto atual de acelerada transformação digital, onde aplicações *web* assumem papel central nas operações empresariais. Ao sistematizar o conhecimento sobre as principais ameaças e apresentar estratégias de proteção alinhadas com as melhores práticas do mercado, esta pesquisa se torna um instrumento valioso para:

- Profissionais de desenvolvimento que buscam criar aplicações mais seguras;
- Equipes de segurança que necessitam de referências para auditorias;
- Gestores de *TI* que precisam priorizar investimentos em segurança.

Os resultados reforçam a premissa de que a segurança deve ser incorporada desde as primeiras etapas do ciclo de desenvolvimento (*Security by Design*), e não tratada como acréscimo posterior. Esta abordagem proativa, quando combinada com o conhecimento das vulnerabilidades aqui apresentadas, pode reduzir drasticamente a superfície de ataque de aplicações *web* e mitigar riscos antes que sejam explorados.

9 Conclusões

Este trabalho demonstra a importância fundamental da segurança em aplicações *web* para profissionais da área de tecnologia, especialmente para aqueles que atuam no desenvolvimento de sistemas. Ao analisar as vulnerabilidades presentes no *OWASP Top* 10, fica evidente como falhas no ciclo de desenvolvimento podem comprometer não apenas sistemas isolados, mas toda a infraestrutura de uma organização. A pesquisa reforça que a segurança deve ser incorporada desde as primeiras etapas de projeto, seguindo a abordagem *Security by Design*, e não tratada como medida reativa após a detecção de vulnerabilidades.

A área de análise e desenvolvimento de sistemas é particularmente beneficiada por este tipo de estudo, pois capacitam profissionais a criar soluções mais robustas e seguras desde sua concepção. O conhecimento sobre as principais ameaças e suas formas de mitigação permite que desenvolvedores e arquitetos de sistemas tomem decisões mais informadas durante todo o processo de desenvolvimento. Em um cenário onde aplicações *web* são a base para a maioria dos serviços digitais, compreender esses riscos se torna essencial para garantir a confiabilidade e integridade dos sistemas.

Além disso, a segurança cibernética deixou de ser preocupação exclusiva de especialistas para se tornar parte integrante da formação de qualquer profissional de tecnologia. As vulnerabilidades discutidas neste trabalho destacam a necessidade de uma abordagem multidisciplinar, envolvendo não apenas desenvolvedores, mas também arquitetos de *software*, administradores de sistemas e gestores de *TI*. A constante evolução das ameaças exige que os profissionais se mantenham atualizados sobre as melhores práticas de segurança, tornando pesquisas como estas valiosas fontes de conhecimento atualizado.

Por fim, este estudo serve como base para o desenvolvimento de aplicações mais seguras e para a conscientização sobre a importância da segurança em todas as etapas do ciclo de vida do *software*. Em um mundo cada vez mais digitalizado, onde ataques cibernéticos causam impactos significativos, o conhecimento sobre vulnerabilidades *web* e suas formas de prevenção se mostra indispensável para profissionais de tecnologia comprometidos com a criação de sistemas confiáveis e protegidos.

Como possível próximo passo, pode ser citado a implementação de laboratórios práticos para a exploração detalhada de cada vulnerabilidade em ambientes controlados, permitindo a validação empírica dos conceitos teóricos. Paralelamente, sugere-se a portabilidade dos códigos desenvolvidos em *Ruby* para outras linguagens de programação, como *Python*, *Java* e *C/C++*, a fim de ampliar a versatilidade e aplicabilidade das soluções

em diferentes contextos tecnológicos. Complementarmente, torna-se essencial a elaboração de documentação técnica abrangente, incluindo procedimentos passo a passo para reprodução dos testes, análise comparativa entre as implementações em diversas linguagens e recomendações específicas de mitigação para cada cenário analisado. Essas iniciativas têm como objetivo fortalecer o rigor metodológico da pesquisa, garantir a reprodutibilidade dos experimentos e expandir a utilidade do material desenvolvido para profissionais e pesquisadores da área de segurança cibernética.

Referências

ASERTO. ***When access controls fail.*** Disponível em: https://www.aserto.com/blog/when-access-controls-fail. Acesso em: 5 jan. 2025.

BACH-NUTMAN, Matthew. ***Understanding the Top 10 OWASP vulnerabilities.*** arXiv preprint arXiv:2012.09960, 2020. Disponível em: https://arxiv.org/abs/2012.09960. Acesso em: 12 fev. 2025.

BRASIL. Ministério do Esporte. **Lei Geral de Proteção de Dados (LGPD).** Disponível em: https://www.gov.br/esporte/pt-br/acesso-a-informacao/lgpd. Acesso em: 18 mar. 2025.

CLOUDFLARE. ***Mirai botnet.*** Disponível em: https://www.cloudflare.com/pt-br/learning/ddos/glossary/mirai-botnet. Acesso em: 3 jun. 2025.

CODACY. ***Cryptographic failures: OWASP Top 10.*** Disponível em: https://blog.codacy.com/cryptographic-failures-*OWASP-Top*-10. Acesso em: 22 mai. 2025.

SCROXTON, Alex. ***More data stolen in 2023 MOVEit attacks comes to light***. Disponível em: https://www.computerweekly.com/news/366615522/More-data-stolen-in-2023-MOVEit-attacks-comes-to-light. Acesso em: 7 abr. 2025.

CSO ONLINE. ***Inside the Russian hack of Yahoo: how they did it.*** Disponível em: https://www.csoonline.com/article/560623/inside-the-russian-hack-of-yahoo-how-they-did-it.html. Acesso em: 14 jan. 2025.

CSO ONLINE. ***Marriott data breach FAQ: how did it happen and what was the impact?*** Disponível em: https://www.csoonline.com/article/567795/marriott-data-breach-faq-how-did-it-happen-and-what-was-the-impact.html. Acesso em: 8 mar. 2025.

CSO ONLINE. ***The biggest data breaches of the 21st century.*** Disponível em: https://www.csoonline.com/article/534628/the-biggest-data-breaches-of-the-21st-century.html. Acesso em: 29 jan. 2025.

LANGLEY, Mitchell. ***When credentials fail: how authentication failure led to the Change Healthcare ransomware attack***. Disponível em: https://dailysecurityreview.com/ransomware/when-credentials-fail-how-authentication-failure-led-to-the-change-healthcare-ransomware-attack/. Acesso em: 17 mai. 2025.

MONTALBANO, Elizabeth. ***#ResumeLooters attackers steal millions of career records***. Disponível em: https://www.darkreading.com/remote-workforce/-resumelooters-attackers-steal-millions-career-records. Acesso em: 9 abr. 2025.

SHERIDAN, Kelly. ***Twilio security incident shows danger of misconfigured S3 buckets.*** Disponível em: https://www.darkreading.com/cloud-security/twilio-security-incident-shows-danger-of-misconfigured-s3-buckets. Acesso em: 21 fev. 2025.

EMSISOFT. ***Unpacking the MOVEit breach: statistics and analysis***. Disponível em: https://www.emsisoft.com/en/blog/44123/unpacking-the-moveit-breach-statistics-and-analysis. Acesso em: 11 jun. 2025.

EPIC.ORG. ***Equifax data breach archive.*** Disponível em: https://archive.epic.org/privacy/data-breach/equifax. Acesso em: 4 mai. 2025.

EQUIFAX. ***Equifax releases details on cybersecurity incident.*** Disponível em: https://investor.equifax.com/news-events/press-releases/detail/237/equifax-releases-details-on-cybersecurity-incident. Acesso em: 30 mar. 2025.

FEDERAL TRADE COMMISSION (FTC). ***Equifax data breach settlement.*** Disponível em: https://www.ftc.gov/enforcement/refunds/equifax-data-breach-settlement. Acesso em: 19 abr. 2025.

FORTINET. ***SolarWinds cyber attack.*** Disponível em: https://www.fortinet.com/br/resources/cyberglossary/solarwinds-cyber-attack. Acesso em: 13 mai. 2025.

GITGUARDIAN. ***State of secrets sprawl report 2024.*** 2024. Disponível em: https://www.gitguardian.com/state-of-secrets-sprawl-report-2024. Acesso em: 6 fev. 2025.

ANDERSON, Jane. ***Florida web designer settles with DOJ***. Disponível em: https://www.jdsupra.com/legalnews/florida-web-designer-settles-with-doj-7910656. Acesso em: 25 abr. 2025.

AIJAZ, Duresham. ***Capital One data breach settlement.*** Disponível em: https://www.purewl.com/capital-one-data-breach-settlement. Acesso em: 16 mar. 2025.

SCHROEDER, Pete. ***Capital One to pay $80 million fine after data breach.*** Disponível em: https://www.reuters.com/article/business/capital-one-to-pay-80-million-fine-after-data-breach-idUSKCN2522D8/. Acesso em: 28 fev. 2025.

SAMPAIO, Felipe Ferreira. ***Uma análise prática das principais vulnerabilidades em aplicações web baseado no Top 10 OWASP.*** 2021. 60 f. Trabalho de Conclusão de Curso (Graduação em Redes de Computadores) – Universidade Federal do Ceará, Campus de Quixadá, Quixadá, 2021. Disponível em: http://repositorio.ufc.br/handle/riufc/62466. Acesso em: 20 jun. 2025.

SOLDERA, Pedro Henrique Rodrigues. **Análise De Vulnerabilidades Em Autenticação.** Trabalho de Conclusão de Curso (Graduação em Tecnologia e Análise de Desenvolvimento) – Faculdade de Tecnologia de Rio Preto, 2023.

STOCKLEY, Mark. ***Yahoo breach: I've closed my account because it uses MD5 to hash my password***. 2016. Disponível em: https://news.sophos.com/en-us/2016/12/15/yahoo-breach-ive-closed-my-account-because-it-uses-md5-to-hash-my-password. Acesso em: 1 abr. 2025.

U.S. SECURITIES AND EXCHANGE COMMISSION (SEC). ***Press release 2018-71.*** 2018. Disponível em: https://www.sec.gov/newsroom/press-releases/2018-71. Acesso em: 15 abr. 2025.

U.S. SECURITIES AND EXCHANGE COMMISSION (SEC). ***Press release 2023-227.*** 2023. Disponível em: https://www.sec.gov/newsroom/press-releases/2023-227. Acesso em: 18 mai. 2025.

SAID, Fátima. ***OWASP Top 10 and their remedies.*** Disponível em: https://xygeni.io/pt/blog/owasp-Top-10-and-their-remedies/. Acesso em: 7 jun. 2025.