Segurança em Aplicações Web com Django: Uma Análise das Vulnerabilidades da OWASP Top 10

Eduardo Felipe da Silva[1], Rafael Perazzo Barbosa Mota[1]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
52.171-900 – Recife – PE – Brazil

{eduardo.fsilva3, rafael.perazzo}@ufrpe.br

Abstract. *Due to the increase in cyberattack incidents in recent years, the topic of web application security has gained increasing prominence. In this context, this work evaluates how the native security features of the Django framework relate to the vulnerabilities identified in the OWASP Top 10. The results show that, although Django natively offers several security features, they do not fully cover all threats. Therefore, the study also suggests additional security practices, following OWASP guidelines, that can mitigate potential threats and improve security in web applications developed with Django.*

Resumo. *Devido ao crescimento nos incidentes de ataques cibernéticos nos últimos anos, a temática da segurança em aplicações web tem ganhado cada vez mais destaque. Diante desse cenário, este trabalho avalia como os recursos de segurança nativos do framework Django se relacionam com as vulnerabilidades identificadas no OWASP Top 10. Os resultados obtidos mostram que, embora o django ofereça nativamente diversos recursos de segurança, eles não cobrem integralmente todas as ameaças. Por isso, o estudo também sugere práticas adicionais de segurança, seguindo as diretrizes do OWASP, que possam mitigar ameaças potenciais e melhorar a segurança em aplicações web desenvolvidas com o Django.*

1. Introdução

O aumento na utilização de aplicações web tem demandado cada vez mais atenção com a segurança durante o processo de desenvolvimento de software. Em 2023, o cenário global de cibersegurança foi marcado por um aumento significativo na frequência e sofisticação dos ataques cibernéticos. Estima-se que ocorriam, em média, 2.220 ataques por dia, totalizando mais de 800.000 incidentes ao longo do ano [Charlton 2024]. Tal cenário torna evidente a necessidade de se adotar técnicas e práticas robustas de proteção, que visem reduzir as ameaças recorrentes em aplicações web.

Nesse contexto, destaca-se o Open Web Application Security Project (OWASP), uma organização sem fins lucrativos que visa promover a segurança de software. A entidade em questão atua como uma comunidade aberta com o intuito de desenvolver, manter e operar aplicações web mais seguras [OWASP 2025]. A escolha da OWASP como referência neste trabalho está relacionada à sua relevância na área de segurança de aplicações web, especialmente por disponibilizar conhecimentos e recomendações voltadas à identificação e mitigação de vulnerabilidades. Entre suas principais contribuições está o OWASP Top 10, um guia que reúne as dez vulnerabilidades mais críticas em aplicações web e serve como referência para orientar o desenvolvimento seguro de aplicações [OWASP 2025]. Devido ao cenário altamente dinâmico de ameaças cibernéticas, o documento é periodicamente atualizado para refletir as mudanças nas práticas de desenvolvimento e as novas realidades da indústria. Diante

disso, este trabalho investiga a sua edição mais recente, publicada em 2025. Nela, é possível destacar algumas vulnerabilidades críticas, tais como quebra de controle de acesso, falhas criptográficas, falhas de autenticação, injeção, entre outras [OWASP 2025].

Em paralelo à questão da segurança, o desenvolvimento ágil e seguro de aplicações web tem sido impulsionado por frameworks como o Django. Sua relevância também está relacionada à ampla adoção da linguagem de programação Python, que atualmente ocupa a primeira posição no TIOBE Index de julho de 2026 [TIOBE Software 2026]. A popularidade da linguagem também pode ser vista na adoção do Django pela comunidade de desenvolvedores, de acordo com a Python Developers Survey 2024, o framework permanece entre os mais utilizados do ecossistema Python, sendo adotado por 35% dos participantes da pesquisa [Python Software Foundation e JetBrains 2024].

Inserido nesse ecossistema, o Django segue o princípio *Don't Repeat Yourself* (DRY), que facilita a criação de aplicações web complexas, diminuindo a necessidade de duplicação de código [Chen et al. 2020]. O Django é amplamente reconhecido por incorporar mecanismos de segurança robustos que incluem proteção contra ataques como *SQL injection*, *Cross-Site Scripting* (XSS) e *Cross-Site Request Forgery* (CSRF), entre outros [Django Project 2026]. Além disso, ele disponibiliza diversos recursos avançados de segurança, como um sistema de autenticação integrado, bem como boas práticas indicadas para escalabilidade, possibilitando que as aplicações web aumentem de modo eficiente e sem interrupções [Chen et al. 2020].

Diante desse cenário, investigar a eficácia dos mecanismos nativos de segurança do Django é cientificamente relevante, uma vez que eventuais limitações podem impactar um grande número de aplicações desenvolvidas com o framework. Assim, considerando que a segurança em aplicações web é um elemento crucial no processo de criação de sistemas online [Kumar e Rani 2022], este trabalho tem como objetivo analisar como o Django lida com as vulnerabilidades listadas no OWASP Top 10 e, quando necessário, propor medidas adicionais que possam ampliar a segurança para além dos recursos nativos oferecidos por ele.

2. Revisão de Literatura

2.1. Fundamentos de Segurança em Aplicações Web

A temática da segurança em aplicações web tem ganhado cada vez mais destaque nos últimos anos, devido ao crescimento nos incidentes de ataques cibernéticos registrados globalmente. De acordo com dados recentes [Trend Micro 2024], somente em 2023, mais de 161 bilhões de ameaças digitais foram interceptadas em todo o mundo, um crescimento de 10% em comparação ao ano anterior, ressaltando ainda mais a importância da utilização de ferramentas adequadas e da adoção de boas práticas no desenvolvimento dessas aplicações.

2.2. Arquitetura e Mecanismos de Segurança do Django

O Django é um framework web de alto nível, escrito em Python, que foi criado para proporcionar um desenvolvimento rápido, seguro e eficiente de aplicações web. Ao contrário de outros frameworks que exigem uma série de configurações iniciais para se iniciar um projeto, o Django segue a filosofia de *batteries included*, ou seja, ele fornece nativamente uma grande variedade de funcionalidades prontas para uso, dentre as quais podemos citar, autenticação de usuários, interface administrativa para gestão de conteúdo, sitemaps, dentre outras.

No que se refere à segurança, o Django não adota o conceito de níveis de segurança selecionáveis durante sua instalação ou configuração. Em vez disso, o framework segue a filosofia *secure by default*, disponibilizando diversos mecanismos de proteção nativos, muitos deles habilitados por padrão. Dessa forma, ele estabelece uma base inicial voltada ao desenvolvimento de aplicações mais seguras, cabendo ao desenvolvedor configurar adequadamente esses recursos e adotar boas práticas para atender aos requisitos específicos de cada aplicação.

Com relação à estrutura, ele se baseia no padrão arquitetural conhecido como Model-Template-View (MTV). Embora seja semelhante ao tradicional Model-View-Controller (MVC), o Django possui particularidades na forma como processa as requisições e gerencia a interface com o usuário, com cada camada possuindo uma funcionalidade específica dentro dessa arquitetura.

2.2.1 Model

É a camada de abstração de dados. No Django, os modelos são classes Python que definem a estrutura das tabelas do banco de dados. Através do Object-Relational Mapping (ORM) nativo, o framework converte automaticamente código Python em comandos SQL. Essa característica é fundamental para a segurança, pois o ORM abstrai a manipulação direta do banco de dados, sanitizando parâmetros e mitigando, por padrão, a maioria dos ataques de injeção de SQL [Django Project 2026].

2.2.2 Template

Corresponde a camada de apresentação, em outras palavras, é o HyperText Markup Language (HTML) final que será entregue ao navegador. O Django possui uma linguagem de template própria Django Template Language (DTL) que é projetada para não permitir a execução de código Python arbitrário. Além disso, o sistema de templates aplica, por padrão, *auto-escaping* de variáveis, convertendo caracteres perigosos em entidades HTML seguras para prevenir ataques de XSS [Django Project 2026].

2.2.3 View

É a camada que atua como o cérebro da lógica de negócios. Ela é responsável por receber a requisição web, consultar os dados necessários através dos models e decidir qual template será renderizado como resposta [Django Project 2026]. É nesta camada que decoradores de segurança, como @login_required ou proteções contra CSRF, são frequentemente aplicados.

Essa separação de responsabilidades não serve apenas para organizar o código, mas também para isolar camadas críticas de segurança, permitindo que validações ocorram em múltiplos níveis antes que um dado malicioso seja processado ou exibido.

2.3. O Projeto OWASP e as Vulnerabilidades Top 10

A OWASP é uma fundação global sem fins lucrativos cujo propósito é melhorar a segurança de software. Ao contrário de empresas privadas de segurança, a OWASP opera como uma comunidade aberta, produzindo artigos, metodologias, documentação e ferramentas que são disponibilizadas gratuitamente para o público.

O projeto mais conhecido da fundação é o OWASP Top 10, uma lista de conscientização para desenvolvedores e profissionais de segurança. Ela contém um consenso amplo sobre os riscos de segurança mais críticos para aplicações web. A versão utilizada como base para análise neste trabalho é a OWASP Top 10:2025, a mais recente até o momento da escrita.

A Figura 1 apresenta a evolução das categorias do OWASP Top 10 entre as edições de 2021 e 2025. Observa-se que, ao longo desse período, algumas vulnerabilidades permaneceram na lista, embora em posições diferentes, enquanto outras foram renomeadas, reclassificadas ou substituídas por novas categorias.

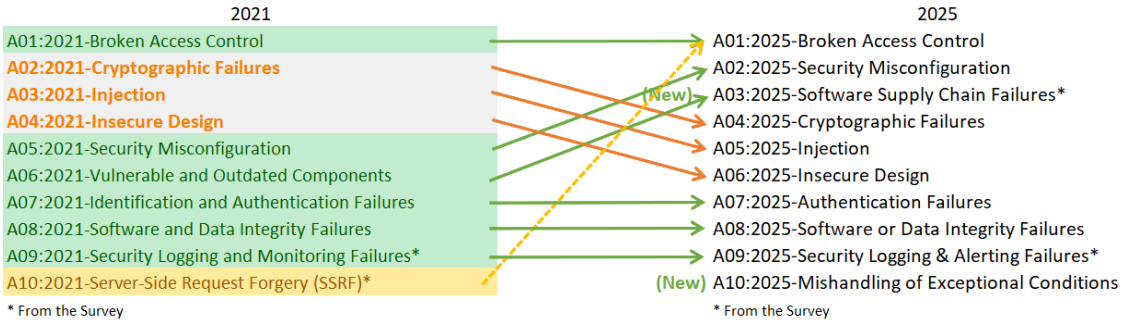

Figura 1. Evolução das categorias do OWASP Top 10 entre 2021 e 2025. Fonte: OWASP (2025).

Como evidenciado na Figura 1, a Quebra de Controle de Acesso (A01: Broken Access Control) se manteve como o risco mais crítico, inclusive englobando outras vulnerabilidades como o SSRF (antiga categoria A10:2021). Destaca-se também a ascensão da Configuração Incorreta de Segurança (A02: Security Misconfiguration), que saltou da quinta posição em 2021 para a segunda posição em 2025, evidenciando que falhas de infraestrutura e deploy estão sendo cada vez mais exploradas. Somado a isso, nota-se a inclusão das categorias de Falhas na Cadeia de Suprimentos de Software (A03: Software Supply Chain Failures) e Tratamento Inadequado de Condições Excepcionais (A10: Mishandling of Exceptional Conditions) na edição de 2025, refletindo mudanças nas ameaças mais relevantes para aplicações web e na forma como elas são avaliadas pela comunidade de segurança.

Essa evolução demonstra o caráter dinâmico do cenário de segurança da informação, no qual novas tecnologias, práticas de desenvolvimento e vetores de ataque influenciam continuamente as principais preocupações da indústria. Nas seções seguintes, cada uma das categorias presentes na versão mais recente do OWASP Top 10

será analisada individualmente, juntamente com os mecanismos de proteção oferecidos pelo framework Django e suas possíveis limitações.

2.4. Trabalhos Relacionados

Os trabalhos apresentados nesta seção foram selecionados a partir da estratégia de busca descrita na Seção 3.2, seguindo os critérios de inclusão e exclusão definidos para esta pesquisa. Dentre os estudos identificados, grande parte aborda a segurança em aplicações web, seja de forma abrangente ou com ênfase em diretrizes como o OWASP Top 10. No entanto, a maior parte desses estudos limita-se a analisar o comportamento de sistemas de órgãos públicos, empresariais ou educacionais frente a algumas das vulnerabilidades elencadas pelo OWASP Top 10, a exemplo dos trabalhos de [Riberu e Emanuel 2024], bem como de [Choiriyah e Qomariasih 2023]. O estudo de [Riberu e Emanuel 2024], analisa a segurança de um site de saúde em um distrito na Indonésia e avalia como este reage quando submetido a testes de penetração que simulam vulnerabilidades descritas pelo OWASP Top 10. O trabalho de [Choiriyah e Qomariasih 2023] adota uma abordagem semelhante, realizando testes em um Sistema de Informações Acadêmicas (SIA) de uma universidade, inclusive utilizando uma ferramenta disponibilizada pelo próprio OWASP, o OWASP Zed Attack Proxy (ZAP), e analisando como esse sistema lida com esses ataques.

Outros estudos, por sua vez, focam em comparar como múltiplas ferramentas de desenvolvimento web enfrentam essas vulnerabilidades, conforme demonstrado em [Aborujilah et al. 2022], que apresenta uma análise comparativa entre os frameworks Laravel, Spring Boot, Django, Ruby, e ASP.NET Core, fornecendo um panorama geral dos recursos oferecidos por cada um deles. Entretanto, o estudo não aprofunda a análise em nenhum dos citados. Há casos também de pesquisas voltadas exclusivamente para o Django, como no caso apresentado em [Sharma et al. 2024], embora seja apresentado como uma visão mais geral dele, centrada na sua arquitetura, no seu processo de instalação e na preparação do ambiente de desenvolvimento.

Sendo assim, a principal contribuição deste trabalho é realizar uma avaliação técnica aprofundada da relação entre os recursos de segurança nativos do Django e o OWASP Top 10, propondo um mapeamento direto entre as vulnerabilidades críticas e seus respectivos mecanismos de mitigação. Adicionalmente, o projeto visa identificar lacunas e práticas inadequadas de segurança no uso do framework, propondo melhorias alinhadas às diretrizes da OWASP. Diferentemente dos estudos existentes, esta pesquisa busca estabelecer uma correlação direta entre as vulnerabilidades do OWASP Top 10 e os recursos oferecidos pelo Django para sua mitigação. Dessa forma, espera-se preencher uma lacuna na literatura e fornecer uma base prática e aplicada que seja relevante tanto para desenvolvedores quanto para pesquisadores da área de segurança da informação, contribuindo para um tema ainda pouco explorado em âmbito nacional e internacional.

3. Metodologia

3.1. Caracterização da pesquisa

Esta pesquisa apresenta caráter exploratório-descritivo, pois tem como objetivo investigar a relação entre os mecanismos de segurança do Django e as vulnerabilidades do OWASP Top 10, ao mesmo tempo em que busca compreender e detalhar os recursos disponibilizados pelo framework para auxiliar na mitigação desses riscos. A abordagem adotada é a qualitativa, uma vez que a análise se baseia na análise interpretativa de referências técnicas e recursos de arquitetura de software, sem recorrer ao uso de métodos estatísticos ou à realização de experimentos quantitativos.

3.2. Estratégias de Busca e Critérios de Seleção

A coleta dos dados para a pesquisa foi realizada através de uma revisão bibliográfica e de análise documental. A revisão bibliográfica focou em artigos científicos e trabalhos acadêmicos relacionados à segurança de aplicações web, que foram obtidos a partir de bases de dados como IEEE Xplore, Google Scholar e ACM Digital Library. Além disso, com o intuito de identificar trabalhos relevantes para o tema, foram utilizadas combinações de palavras-chave como "Django", "OWASP", "Secure Web Development", "Vulnerability Testing" e "Security Analysis on Websites", permitindo assim, direcionar a busca para estudos relacionados a segurança em aplicações web.

Para a análise, foram considerados trabalhos publicados entre 2020 e 2025, redigidos em inglês ou português, que se relacionavam à segurança de aplicações web, ao framework Django ou às diretrizes propostas pela OWASP. Dessa forma, foram incluídos tanto estudos voltados especificamente ao Django quanto trabalhos que abordavam outros frameworks ou realizavam análises baseadas no OWASP Top 10, desde que apresentassem contribuições relevantes para a compreensão dos mecanismos de segurança em aplicações web. Em contrapartida, foram excluídos estudos que não apresentavam relação direta com vulnerabilidades em aplicações web ou que não forneciam informações relevantes para a análise da cobertura de segurança oferecida pelo framework Django.

Além da literatura científica, a pesquisa também contou com uma análise documental, que teve como fontes primárias a documentação oficial do Django (versão 6.0) e a documentação do OWASP Top 10:2025. Essas fontes foram complementadas por guias, recomendações e outros materiais técnicos disponibilizados pela própria OWASP, os quais serviram de base para a identificação de mecanismos de segurança e para compreensão das vulnerabilidades analisadas ao longo deste artigo.

3.3. Procedimentos de Mapeamento

O procedimento de mapeamento foi realizado em três etapas. Inicialmente, foi feito o levantamento das dez categorias de risco presentes no OWASP Top 10:2025, visando compreender suas características, impactos e formas de exploração.

Em seguida, para cada categoria de vulnerabilidade, foi realizada uma busca sistemática na documentação oficial do Django com o objetivo de identificar mecanismos nativos relacionados à sua mitigação. Foram analisados recursos como

middlewares, decorators de views, sistema de autenticação, ORM, mecanismos de proteção contra CSRF e demais funcionalidades de segurança disponibilizadas pelo framework.

Por fim, foi realizada uma análise crítica da cobertura fornecida pelo Django para cada categoria do OWASP Top 10:2025. Essa análise buscou determinar se os mecanismos identificados eram suficientes para mitigar de maneira autônoma os riscos associados ou se dependiam de configurações adicionais e da adoção de boas práticas por parte do desenvolvedor. A partir dessa avaliação, foram identificadas possíveis lacunas e limitações relacionadas à segurança do framework.

4. Análise de Segurança no Django

Este capítulo apresenta os resultados obtidos a partir do mapeamento técnico realizado entre as categorias de risco descritas no OWASP Top 10:2025 e os mecanismos de segurança disponibilizados pelo Django (versão 6.0). Conforme definido na metodologia, cada subseção analisa individualmente uma categoria de vulnerabilidade, descrevendo sua natureza, os recursos de mitigação nativos oferecidos pelo framework e as eventuais limitações que necessitam da adoção de configurações adicionais ou de boas práticas por parte do desenvolvedor.

Para padronizar a avaliação da capacidade de mitigação do Django em relação a cada categoria do OWASP Top 10:2025, foi definida uma escala de cobertura composta por quatro níveis. Os critérios utilizados para essa classificação são apresentados na Tabela 1.

Tabela 1. Níveis de cobertura nativos do django

Nível de Cobertura	Critério de Avaliação
N/A (Não Aplicável)	Utilizado quando a vulnerabilidade não está diretamente relacionada aos mecanismos de segurança do Django, sendo influenciada por fatores externos, como regras de negócio, dependências de terceiros ou processos de desenvolvimento.
Baixa	Quando há pouco ou nenhum recurso nativo, exigindo que o desenvolvedor implemente a proteção quase em sua totalidade ou dependa de soluções externas.
Parcial	Quando o framework fornece recursos robustos e bem consolidados para o problema geral, porém apresenta limitações em cenários específicos que exigem bibliotecas ou configurações complementares.
Alta	Quando a vulnerabilidade é mitigada de forma autônoma ou por meio de configurações simples, seguindo o princípio de secure by default.

4.1. Quebra de Controle de Acesso

A Quebra de Controle de Acesso (Broken Access Control), classificada como o risco mais crítico (A01) na edição de 2025 do OWASP Top 10, ocorre quando uma aplicação

não emprega corretamente as restrições de acesso definidas para cada usuário. Como consequência, um atacante pode acessar recursos para os quais não possui autorização, visualizar informações sensíveis, modificar dados de outros usuários ou até mesmo obter privilégios mais elevados. Na edição de 2025, essa categoria absorveu também os ataques de SSRF, onde o invasor explora a confiança da aplicação para induzir o servidor a realizar requisições HTTP arbitrárias.

No contexto do Django, o nível de cobertura nativa definido para essa vulnerabilidade foi classificado como parcial. O framework oferece uma base sólida para implementação de controle de acesso por meio do módulo django.contrib.auth, que disponibiliza um sistema integrado de autenticação, grupos e permissões. Além disso, o acesso às views pode ser restringido de forma simples utilizando decorators, como login_required e permission_required. O Código abaixo apresenta um exemplo dessa proteção, em que apenas usuários autenticados e com a permissão necessária podem acessar a funcionalidade de edição de documentos.

```python
from django.contrib.auth.decorators import login_required, permission_required
from django.shortcuts import get_object_or_404, render
from .models import Documento


@login_required
@permission_required('app.change_documento', raise_exception=True)
def editar_documento(request, doc_id):
    documento = get_object_or_404(Documento, pk=doc_id)
    return render(request, 'editar.html', {'documento': documento})
```

Por mais que o trecho exibido acima impeça o acesso de usuários não autenticados ou sem a permissão necessária, ele também evidencia a principal limitação que justifica a classificação parcial para essa categoria. O mecanismo de permissões do Django verifica apenas se o usuário possui a autorização exigida para acessar a funcionalidade, mas não valida de maneira automática se ele tem permissão para acessar ou modificar um objeto específico. Assim, um usuário que tenha a permissão genérica change_documento vai poder alterar o parâmetro doc_id na URL (por exemplo, /documentos/3/) e editar documentos pertencentes a outros usuários, caracterizando uma vulnerabilidade do tipo Insecure Direct Object Reference (IDOR), também conhecida como Autorização Quebrada em Nível de Objeto (Broken Object Level Authorization - BOLA).

Para mitigar essa lacuna, é necessário que o desenvolvedor implemente verificações adicionais de autorização em nível de objeto. O código abaixo apresenta uma abordagem utilizando o próprio ORM do Django, restringindo a consulta aos documentos pertencentes ao usuário autenticado.

```python
from django.contrib.auth.decorators import login_required, permission_required
from django.shortcuts import get_object_or_404, render
from .models import Documento


@login_required
@permission_required('app.change_documento', raise_exception=True)
def editar_documento(request, doc_id):
    # O filtro 'proprietario=request.user' garante o controle de acesso ao objeto
    documento = get_object_or_404(
        Documento,
        pk=doc_id,
        proprietario=request.user
    )
    return render(request, 'editar.html', {'documento': documento})
```

Assim, a mitigação dessa vulnerabilidade só é de fato alcançada quando o desenvolvedor implementa verificações de autorização em nível de objeto. E uma das formas de mitigar esse problema consiste em restringir a consulta ao banco de dados ao usuário autenticado, conforme ilustrado no código acima. Outra forma de realizar essa validação é por meio de uma verificação explícita da propriedade do objeto, como na condição if documento.proprietario != request.user: raise PermissionDenied. Mas, independentemente da estratégia adotada, a validação em nível de objeto permanece sob responsabilidade do desenvolvedor para impedir acessos não autorizados.

4.2. Configuração Incorreta de Segurança

A categoria de Configuração Incorreta de Segurança (Security Misconfiguration), classificada como (A02) no OWASP Top 10:2025, refere-se à utilização de configurações inseguras durante a implantação de uma aplicação, como configurações padrão do framework, mensagens de erro detalhadas, cabeçalhos HTTP inadequados e permissões excessivas. Diferentemente de vulnerabilidades causadas por erros no código da aplicação, essa categoria está relacionada principalmente à configurações inadequadas do ambiente onde o sistema é executado.

Para lidar com isso, o Django disponibiliza alguns mecanismos para auxiliar na configuração segura da aplicação, como o *SecurityMiddleware*, que é responsável por adicionar cabeçalhos HTTP de segurança, e o comando python manage.py check --deploy, que verifica automaticamente configurações recomendadas para ambientes de produção. Sendo assim, o nível de cobertura nativa para essa categoria foi classificado como parcial, o código abaixo apresenta alguns dos recursos de segurança que são inseridos automaticamente pelo framework durante a criação de um novo projeto.

```python
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Contudo, a utilização desses mecanismos depende que a aplicação seja configurada de maneira correta. O Django não impede, por exemplo, que uma aplicação seja implantada com DEBUG = True, nem garante que informações sensíveis, como SECRET_KEY ou ALLOWED_HOSTS, sejam configuradas adequadamente. Para reduzir este risco, é fortemente recomendado armazenar essas configurações em variáveis de ambiente, conforme exemplificado no código abaixo.

```python
import os

SECRET_KEY = os.environ["DJANGO_SECRET_KEY"]

ALLOWED_HOSTS = os.environ.get(
    "DJANGO_ALLOWED_HOSTS", ""
).split(",")
```

Portanto, observa-se que o Django fornece mecanismos importantes para auxiliar na configuração segura de aplicações, mas sua eficácia também depende da correta utilização dessas funcionalidades pelo desenvolvedor. Dessa forma, a proteção contra configurações incorretas está diretamente relacionada à adoção das boas práticas recomendadas pelo framework durante a implantação da aplicação.

4.3. Falhas na Cadeia de Suprimentos de Software

As Falhas na Cadeia de Suprimentos de Software (Software Supply Chain Failures), classificadas como (A03) no OWASP Top 10:2025, estão relacionadas aos riscos introduzidos pelos componentes e ferramentas que fazem parte do processo de desenvolvimento de uma aplicação. Essa categoria abrange problemas decorrentes da utilização de dependências de terceiros que podem ser vulneráveis ou maliciosas, de bibliotecas com vulnerabilidades conhecidas (CVEs) e a falhas presentes nos processos de integração e entrega contínua (CI/CD).

Embora o framework disponibilize diversos mecanismos voltados ao desenvolvimento seguro de aplicações, ele não possui recursos nativos destinados à análise da integridade ou da segurança das dependências utilizadas pelo projeto. Dessa forma, a responsabilidade pela segurança da cadeia de suprimentos é atribuída às ferramentas do ecossistema Python e aos processos adotados durante o ciclo de desenvolvimento. Consequentemente, não existem middlewares, decorators ou configurações nativas capazes de impedir que uma biblioteca vulnerável ou comprometida seja adicionada à aplicação. Por isso, a classificação de cobertura fornecida pelo Django para essa categoria foi classificada como N/A (Não Aplicável).

Assim, a mitigação desse tipo de ameaça depende da utilização de ferramentas externas de análise de composição de software (*Software Composition Analysis* – SCA), como o pip-audit. A Figura 2 apresenta o resultado obtido após a execução do pip-audit em um projeto desenvolvido com o framework Django, destacando as dependências que apresentam vulnerabilidades conhecidas e as respectivas versões corrigidas.

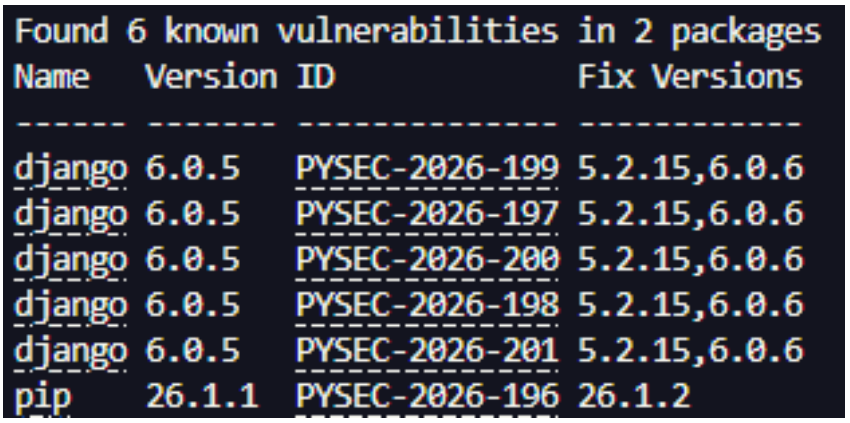

Figura 2. Resultado da auditoria de dependências com o pip-audit

Logo, a principal limitação não vem de uma deficiência do framework, mas sim do fato de que a segurança da cadeia de suprimentos envolve aspectos externos ao seu escopo de atuação. Nesse cenário, a utilização de processos de DevSecOps e de ferramentas de auditoria torna-se essencial para identificar e gerenciar dependências vulneráveis.

4.4. Falhas Criptográficas

As falhas criptográficas (cryptographic failures), classificadas como (A04) no OWASP Top 10:2025, estão relacionadas à exposição de informações sensíveis devido à ausência ou do uso inadequado de mecanismos criptográficos. Entre os principais riscos estão a transmissão de dados em texto claro pela rede e o armazenamento de informações confidenciais sem proteção adequada no banco de dados.

O framework possui mecanismos robustos para a proteção de credenciais, realizando automaticamente o armazenamento seguro de senhas por meio do algoritmo PBKDF2 com SHA-256 e permitindo a utilização de outros algoritmos, como BCrypt e Argon2. Para proteger os dados em trânsito, o Django também oferece configurações nativas no arquivo settings.py, permitindo o redirecionamento automático das requisições para HTTPS, a proteção dos cookies de sessão e do token CSRF e o suporte

ao mecanismo HTTP Strict Transport Security (HSTS), conforme exibido no código abaixo.

```python
# Redireciona automaticamente conexões HTTP para HTTPS
SECURE_SSL_REDIRECT = True

# Impede o envio de cookies de sessão e CSRF por conexões HTTP
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True

# Habilita HSTS, instruindo o navegador a utilizar apenas HTTPS por 1 ano
SECURE_HSTS_SECONDS = 31536000
```

As configurações apresentadas acima não são habilitadas automaticamente pelo framework, ou seja, devem ser configuradas durante a implantação da aplicação. Dessa forma, a proteção dos dados em trânsito depende tanto da configuração correta do ambiente quanto da utilização de conexões HTTPS. Além disso, a cobertura oferecida pelo Django não é total, pois o framework não realiza, de forma nativa, a criptografia de dados armazenados em nível de campo no banco de dados. Assim, caso um desenvolvedor utilize um campo como models.CharField() para armazenar informações sensíveis, esses dados serão gravados em texto claro, exceto se forem adotadas medidas adicionais de proteção. Para suprir essa limitação, é fortemente recomendada a utilização de bibliotecas especializadas em criptografia de campos, ou de mecanismos que sejam equivalentes disponibilizados pelo próprio sistema gerenciador do banco de dados.

Com isso, observa-se que o Django oferece uma cobertura abrangente, mas não total, para a proteção de credenciais e da comunicação entre cliente e servidor e, por isso, a cobertura nativa para essa categoria foi classificada como parcial. Isso torna evidente que a proteção de informações sensíveis armazenadas em repouso permanece sob responsabilidade do desenvolvedor, que deve adotar mecanismos complementares de criptografia sempre que necessário.

4.5. Injeção

A injeção (injection), classificada como (A05) no OWASP Top 10:2025, ocorre quando dados fornecidos pelo usuário são interpretados como parte de comandos ou consultas executadas pela aplicação. Como consequência, um invasor pode manipular o comportamento do sistema, acessar informações não autorizadas ou executar comandos arbitrários. Entre os ataques mais conhecidos desta categoria destacam-se a Injeção de SQL (SQL Injection), além de outras formas de injeção em interpretadores e comandos.

Um dos principais mecanismos de proteção do framework é o seu mapeamento objeto-relacional (ORM), cuja API baseada em QuerySets constrói consultas parametrizadas automaticamente. Dessa forma, os dados fornecidos pelo usuário são

tratados como parâmetros da consulta, impedindo que sejam interpretados como comandos SQL. O código a seguir apresenta um exemplo desse comportamento.

```python
# Obtém o valor informado pelo usuário
busca = request.GET.get("username")


# O ORM parametriza automaticamente a consulta
usuario = Usuario.objects.filter(username=busca)
```

Além da proteção contra SQL Injection, o mecanismo de templates do Django realiza automaticamente o auto-escaping de caracteres especiais em conteúdo HTML, reduzindo significativamente o risco de ataques de XSS quando os templates são utilizados conforme recomendado pelo framework.

Entretanto, essas proteções podem ser comprometidas quando o desenvolvedor opta por contornar os mecanismos nativos disponibilizados pelo framework. No caso do XSS, por exemplo, a utilização, em situações específicas, de recursos como {% autoescape off %} ou da função mark_safe() desabilita o auto-escaping, permitindo que conteúdos potencialmente maliciosos sejam renderizados diretamente no navegador. Da mesma forma, em situações que exigem consultas SQL específicas, o uso de recursos como RawSQL ou a execução de comandos diretamente por meio do cursor do banco de dados contorna a proteção oferecida pelo ORM, fazendo com que a validação e a parametrização dos dados passem a ser responsabilidade do desenvolvedor.

Assim, observa-se que o Django oferece uma cobertura eficiente para a prevenção de vulnerabilidades de injeção, adotando mecanismos de proteção nativos que seguem o princípio *secure by default*. Em razão desse conjunto de mecanismos, o nível de cobertura nativa para essa categoria foi classificado como alto. As principais lacunas surgem apenas quando esses mecanismos são intencionalmente desativados ou contornados, reforçando a importância da utilização das APIs e das práticas recomendadas pelo próprio framework.

4.6. Design Inseguro

A categoria de Design Inseguro (Insecure Design), classificada como (A06) no OWASP Top 10:2025, está relacionada a falhas conceituais presentes na arquitetura e nas regras de negócio das aplicações. Diferentemente de vulnerabilidades que ocorrem a partir de erros de implementação, essa categoria está associada à ausência ou à modelagem inadequada de ameaças, bem como à definição incorreta dos requisitos de segurança. Entre os exemplos mais comuns estão fluxos de recuperação de senha que permitem a identificação de usuários válidos, a ausência de restrições para operações sensíveis e regras de negócio que podem ser exploradas de maneira indevida por usuários mal-intencionados.

Embora o framework disponibilize diversos mecanismos robustos que proporcionam uma maior segurança para as aplicações, como autenticação, autorização e proteção contra vulnerabilidades específicas, ele não é capaz de avaliar

automaticamente se as regras de negócio definidas pelo desenvolvedor foram projetadas de maneira segura. Em outras palavras, o Django executa a lógica implementada na aplicação conforme especificada, sem distinguir se determinadas decisões de projeto podem resultar em riscos de segurança. Por esse motivo, a cobertura fornecida pelo Django para essa categoria foi classificada como N/A (Não Aplicável).

Dessa forma, a mitigação dessa categoria depende principalmente da adoção de práticas de desenvolvimento seguro durante as etapas de análise e projeto do sistema, uma vez que não é possível automatizar decisões relacionadas à arquitetura e às regras de negócio da aplicação. Logo, técnicas como modelagem de ameaças, definição adequada dos requisitos de segurança, aplicação do princípio do menor privilégio e utilização de estratégias de defesa em profundidade são fundamentais para reduzir a probabilidade de ocorrência dessas falhas. Consequentemente, a responsabilidade pela prevenção de falhas de design recai principalmente sobre o desenvolvedor e sobre as práticas de engenharia de software adotadas durante o desenvolvimento do sistema.

4.7. Falhas de Autenticação

As Falhas de Autenticação (Authentication Failures), classificadas como (A07) no OWASP Top 10:2025, reúne vulnerabilidades relacionadas ao comprometimento da identidade dos usuários, incluindo vazamento de credenciais, gerenciamento inadequado de sessões, exposição de tokens e ataques de força bruta.

O framework disponibiliza, por meio do módulo django.contrib.auth, um sistema completo para autenticação, gerenciamento de usuários e sessões, além de validadores de senha configurados pela variável AUTH_PASSWORD_VALIDATORS, conforme o código exibido a seguir.

```
Python

AUTH_PASSWORD_VALIDATORS = [
    {'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValid
ator'},
    {'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
     'OPTIONS': {'min_length': 12}},
    {'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator'},
    {'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator'},
]
```

Entretanto, o Django não fornece, de forma nativa, mecanismos de limitação de tentativas de autenticação (*rate limiting*) nem autenticação multifator (Multi-Factor Authentication – MFA). Para suprir essas lacunas, é recomendada a utilização de bibliotecas complementares, como django-axes, que é responsável por monitorar e limitar tentativas consecutivas de autenticação, bloqueando temporariamente usuários ou endereços IP após um número configurável de falhas, o que reduz a eficácia de

ataques de força bruta. Outra alternativa é o django-mfa2, que adiciona uma segunda camada de autenticação ao processo de login, oferecendo suporte a outros métodos de autenticação, como aplicativos compatíveis com Time-based One-Time Password (TOTP), chaves de segurança e códigos de recuperação, dificultando, assim, o acesso não autorizado mesmo em casos de comprometimento das credenciais.

Diante disso, o nível de cobertura nativa para essa categoria foi classificado como alto. Observa-se que o Django oferece uma cobertura elevada para a prevenção de falhas de autenticação, embora mecanismos complementares, como proteção contra ataques automatizados e autenticação multifator, dependam da integração com soluções externas.

4.8. Falhas de Integridade de Software ou Dados

As Falhas de Integridade de Software ou Dados (Software or Data Integrity Failures), classificadas como (A08), englobam vulnerabilidades relacionadas à alteração indevida de dados, ao uso de mecanismos inseguros de serialização e à ausência de verificação da integridade de componentes e dependências utilizados pela aplicação.

Entre os recursos disponibilizados pelo Django para reduzir esse tipo de risco destaca-se o módulo django.core.signing, que permite assinar criptograficamente dados trocados entre cliente e servidor. Esse mecanismo possibilita verificar se as informações foram alteradas após sua geração, o que contribui para preservar a sua integridade, conforme demonstrado abaixo.

```python
from django.core import signing

valor_assinado = signing.dumps({"user_id": 42, "role": "admin"})

try:
    dado_original = signing.loads(valor_assinado, max_age=3600)
except signing.BadSignature:
    print("O dado foi corrompido ou adulterado!")
```

Ainda assim, ele não impede que o desenvolvedor utilize mecanismos inseguros de serialização, como o módulo *pickle*, que é utilizado para serializar e desserializar objetos Python. Quando empregado para processar dados de fontes não confiáveis, esse módulo pode permitir a execução de código maliciosos, comprometendo a segurança da aplicação. Além disso, o Django não realiza a verificação da integridade de dependências e artefatos distribuídos durante o processo de desenvolvimento ou implantação da aplicação. Dessa forma, a proteção dessa categoria depende principalmente da adoção de boas práticas de desenvolvimento e da utilização de ferramentas complementares para validação da integridade dos componentes empregados.

Considerando essas características, a cobertura nativa do Django para essa categoria foi classificada como parcial, uma vez que o framework disponibiliza mecanismos para garantir a integridade de dados específicos da aplicação, mas não contempla aspectos relacionados à cadeia de desenvolvimento e ao uso seguro de componentes externos.

4.9. Falhas de Registro de Segurança e Alertas

A categoria de Falhas de Registro de Segurança e Alertas (Security Logging & Alerting Failures), classificada como (A09), refere-se à ausência ou a implementação incorreta de mecanismos de registro e monitoramento de eventos de segurança, dificultando a identificação, a investigação e a resposta a incidentes.

Para auxiliar nesse processo, o Django disponibiliza uma infraestrutura baseada no módulo logging da linguagem Python, responsável pelo registro de eventos gerados pela aplicação. Esse mecanismo permite direcionar os registros para diferentes destinos, como arquivos, consoles ou serviços externos, conforme exibido no código a seguir.

Python

```
LOGGING = {
    'version': 1,
    'handlers': {
        'file': {
            'level': 'WARNING',
            'class': 'logging.FileHandler',
            'filename': '/var/log/django/security.log',
        },
    },
    'loggers': {
        'django.security': {
            'handlers': ['file'],
            'level': 'WARNING',
            'propagate': False,
        },
    },
}
```

Apesar de fornecer essa infraestrutura, o framework não disponibiliza mecanismos nativos para correlação de logs, detecção de comportamentos suspeitos ou geração automática de alertas. Dessa forma, uma estratégia eficaz de monitoramento depende da integração com ferramentas especializadas, como plataformas SIEM ou soluções de monitoramento de aplicações.

Considerando essas características, a cobertura nativa do Django para essa categoria foi classificada como baixa, uma vez que o framework limita-se ao registro

dos eventos, cabendo ao desenvolvedor implementar mecanismos adicionais para detecção e resposta a incidentes.

4.10. Tratamento Inadequado de Condições Excepcionais

O Tratamento Inadequado de Condições Excepcionais (Mishandling of Exceptional Conditions), classificado como (A10) no OWASP Top 10:2025, está relacionado a exposição de informações sensíveis decorrente do tratamento inadequado de erros e exceções pela aplicação. Entre as consequências mais comuns estão o vazamento de rastreamentos de pilha (stack traces), caminhos internos do sistema, nomes de variáveis e mensagens detalhadas de erro, que podem fornecer informações úteis para um atacante compreender a arquitetura da aplicação.

Para lidar com esse risco, o Django adota comportamentos distintos para os ambientes de desenvolvimento e produção por meio da variável DEBUG. Quando configurada com DEBUG = False, o framework substitui automaticamente páginas detalhadas de erro por respostas genéricas para exceções, como HTTP 404 e HTTP 500, evitando a exposição de informações técnicas ao usuário final. O código abaixo apresenta um exemplo dessa configuração.

```python
# Ambiente de produção
DEBUG = False


handler404 = "core.views.pagina_nao_encontrada"
handler500 = "core.views.erro_interno_customizado"
```

Mas, a eficácia desse mecanismo depende também que a aplicação seja configurada corretamente. Caso o ambiente de produção seja executado com DEBUG = True, mensagens detalhadas de erro poderão ser exibidas aos usuários. Além disso, a própria implementação da aplicação pode comprometer essa proteção quando exceções são capturadas e suas mensagens são retornadas diretamente nas respostas HTTP, expondo informações internas do sistema.

Diante dessas características, a cobertura nativa do Django para essa categoria foi classificada como alta, pois o framework fornece mecanismos eficientes para o tratamento seguro de exceções. Contudo, sua eficácia depende da utilização das configurações recomendadas para ambientes de produção e da implementação adequada do tratamento de erros pela equipe de desenvolvimento.

5. Discussão e Resultados

A análise das dez categorias presentes no OWASP Top 10:2025 permitiu avaliar o nível de cobertura nativa oferecido pelo framework Django diante das principais vulnerabilidades que afetam as aplicações web. Os resultados obtidos mostraram que o framework dispõe de diversos mecanismos de segurança por padrão, mas também evidenciam que sua efetividade depende da correta configuração do ambiente de trabalho, da implementação adequada das regras de negócio e da adoção de práticas

complementares, por parte dos desenvolvedores, para proporcionar um desenvolvimento seguro. Além disso, foi possível observar que a cobertura oferecida pelo framework varia conforme a natureza da vulnerabilidade analisada, apresentando maior efetividade em categorias relacionadas à implementação técnica e uma menor cobertura em aspectos que dependem de decisões arquiteturais, operacionais ou do processo de desenvolvimento. Essa constatação reforça que a segurança de uma aplicação web deve ser compreendida como um processo contínuo, no qual o framework representa apenas um dos elementos envolvidos na mitigação dos riscos. Assim, a adoção de boas práticas durante todas as etapas do desenvolvimento torna-se tão importante quanto os mecanismos de proteção disponibilizados pela própria tecnologia.

A Tabela 2 ilustra os resultados obtidos ao longo da pesquisa, relacionando cada categoria do OWASP Top 10:2025 ao seu respectivo nível de cobertura identificado no Django.

Tabela 2. Resultado da análise da cobertura nativa do Django frente ao OWASP Top 10:2025

Vulnerabilidade OWASP Top 10 (2025)	Nível de Cobertura do Django
A01:2025 – Quebra de Controle de Acesso (Broken Access Control)	Parcial
A02:2025 – Configuração Incorreta de Segurança (Security Misconfiguration)	Parcial
A03:2025 – Falhas na Cadeia de Suprimentos de Software (Software Supply Chain Failures)	Não Aplicável
A04:2025 – Falhas Criptográficas (Cryptographic Failures)	Parcial
A05:2025 – Injeção (Injection)	Alta
A06:2025 – Design Inseguro (Insecure Design)	Não Aplicável
A07:2025 – Falhas de Autenticação (Authentication Failures)	Alta
A08:2025 – Falhas de Integridade de Software ou Dados (Software or Data Integrity Failures)	Parcial
A09:2025 – Falhas de Registro de Segurança e Alertas (Security Logging & Alerting Failures)	Baixa
A10:2025 – Tratamento Inadequado de Condições Excepcionais (Mishandling of Exceptional Conditions)	Alta

Com isso, é possível observar que o Django apresentou alta cobertura em três categorias (A05, A07 e A10), cobertura parcial em quatro categorias (A01, A02, A04 e A08), baixa cobertura em uma categoria (A09) e duas categorias foram consideradas não aplicáveis (A03 e A06), por tratarem de aspectos que estão fora do escopo de atuação de um framework web. Esses resultados comprovam que o Django oferece uma base sólida de proteção para vulnerabilidades diretamente relacionadas ao desenvolvimento da aplicação, especialmente aquelas mitigadas por mecanismos incorporados ao próprio framework. Além disso, observa-se que o Django não adota níveis de segurança configuráveis, em vez disso, ele segue a filosofia secure by default, na qual o framework fornece mecanismos de proteção nativos cuja efetividade depende de sua correta configuração e utilização.

Contudo, essa proteção está concentrada principalmente nos aspectos internos relacionados ao desenvolvimento da aplicação. As categorias que envolvem fatores externos, como a cadeia de suprimentos, decisões de arquitetura, monitoramento contínuo e configurações do ambiente, exigem a adoção de práticas complementares e o uso de ferramentas especializadas. Dessa forma, foi possível observar que a segurança proporcionada pelo Django está diretamente relacionada à utilização adequada de seus recursos e ao comprometimento da equipe de desenvolvimento com a aplicação das boas práticas de segurança.

5.1 O Django é suficiente para proteger uma aplicação?

Os resultados comprovam que o Django oferece uma base sólida para o desenvolvimento de aplicações web mais seguras. Recursos como o ORM, o sistema de autenticação, os mecanismos de proteção contra CSRF, o tratamento seguro de exceções e os recursos voltados à configuração de HTTPS reduzem significativamente a ocorrência de diversas vulnerabilidades previstas pelo OWASP Top 10:2025.

Contudo, a análise também mostra que o framework, por si só, não é capaz de garantir a segurança completa de uma aplicação. Vulnerabilidades relacionadas à lógica de negócio, ao controle de acesso em nível de objeto, à configuração do ambiente, ao gerenciamento de dependências e ao monitoramento contínuo permanecem sob responsabilidade da equipe de desenvolvimento. Dessa forma, fica claro que a segurança da aplicação depende da combinação entre os mecanismos nativos oferecidos pelo Django e a adoção de práticas adequadas de desenvolvimento seguro.

5.2 Principais Limitações e Responsabilidades do Desenvolvedor

Os resultados obtidos mostram que as principais limitações identificadas durante a análise decorrem não da ausência de mecanismos de segurança no Django, mas da forma como esses recursos são utilizados durante o desenvolvimento da aplicação. Em grande parte das categorias classificadas como de cobertura parcial ou baixa, o framework disponibiliza ferramentas capazes de mitigar os riscos, mas sua efetividade depende da forma como esses recursos são implementados e configurados pela equipe de desenvolvimento.

Entre as situações mais recorrentes estão a ausência de verificações de autorização em nível de objeto, configurações inadequadas em ambientes de produção, o

armazenamento de informações sensíveis sem mecanismos adicionais de criptografia, a utilização de componentes externos vulneráveis e a falta de monitoramento contínuo da aplicação. Esses aspectos demonstram que o uso de um framework reconhecido por suas funcionalidades de segurança não elimina a necessidade de boas práticas de desenvolvimento nem substitui o conhecimento técnico dos profissionais envolvidos.

Nesse contexto, observa-se que a segurança de aplicações desenvolvidas com Django deve ser tratada como um processo contínuo, que envolve tanto a utilização adequada dos recursos nativos do framework quanto a adoção de medidas complementares. Entre elas destacam-se a correta configuração do ambiente de produção, a implementação de controles de autorização em nível de objeto, a realização periódica de auditorias nas dependências da aplicação, a integração com mecanismos de monitoramento e a incorporação de práticas de segurança ao longo de todo o ciclo de desenvolvimento do software. Dessa forma, as lacunas identificadas nesta pesquisa podem ser significativamente reduzidas, ampliando o nível de proteção das aplicações desenvolvidas com o framework.

6. Conclusão

Este trabalho contribui para a área de segurança de aplicações *web* ao analisar o nível de cobertura nativa oferecido pelo framework Django frente às vulnerabilidades descritas no OWASP Top 10:2025. A análise realizada permitiu relacionar os mecanismos de segurança disponibilizados pelo framework às principais categorias de vulnerabilidades atualmente reconhecidas pela OWASP, fornecendo uma visão mais ampla sobre os pontos em que o Django oferece proteção nativa e aqueles que permanecem sob responsabilidade da equipe de desenvolvimento. Dessa forma, o estudo fornece um panorama que pode auxiliar desenvolvedores e pesquisadores na adoção de práticas mais seguras durante o desenvolvimento de aplicações web.

Os resultados obtidos mostraram que o Django possui mecanismos robustos para mitigar várias dessas vulnerabilidades relacionadas à injeção, às falhas de autenticação e ao tratamento inadequado de condições excepcionais. Contudo, algumas categorias como quebra de controle de acesso, configuração incorreta de segurança, design inseguro, falhas de integridade de software ou dados dependem da correta configuração do framework e da implementação de mecanismos adicionais pela equipe de desenvolvimento. Assim, foi possível observar que a efetividade dos recursos nativos do Django está diretamente relacionada à adoção de boas práticas de desenvolvimento seguro e à correta configuração do ambiente de implantação.

Como limitação deste estudo, destaca-se que a pesquisa possui caráter bibliográfico e analítico, sendo baseada na documentação oficial do Django, nas recomendações do OWASP e na literatura especializada, sem a realização de experimentos práticos para avaliar a efetividade dos mecanismos de segurança em diferentes cenários de ataque. Além disso, a classificação dos níveis de cobertura apresentada neste trabalho foi construída a partir da análise dessas evidências, podendo ser aperfeiçoada e validada por futuras investigações experimentais.

Como trabalho futuro, sugere-se a realização de estudos experimentais para validar, na prática, a eficácia dos mecanismos de segurança analisados. Isso pode ser

feito por meio da implementação de aplicações em Django e da exploração controlada de vulnerabilidades em diferentes cenários de configuração. Essa abordagem permitiria avaliar como diferentes perfis de segurança, desde ambientes de desenvolvimento mais flexíveis até configurações de produção mais restritas, influenciam a proteção das aplicações. Esse estudo pode complementar os resultados obtidos nesta pesquisa e contribuir para uma compreensão mais ampla sobre a efetividade dos mecanismos de segurança nativos oferecidos pelo Django.

Referências

Aborujilah, A., Adamu, J., Shariff, S. M. and Long, Z. A. (2022) "Descriptive Analysis of Built-in Security Features in Web Development Frameworks", 2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Korea, Republic of, pp. 1–8.

Charlton, E. (2024). "2023 was a big year for cybercrime – here's how we can make our systems safer". World Economic Forum. Disponível em: https://www.weforum.org/stories/2024/01/cybersecurity-cybercrime-system-safety/. Acesso em: 06 jul. 2026.

Chen, S. et al. (2020) "Django Web Development Framework: Powering the Modern Web", American journal of trade and policy, v. 7, n. 3, p. 99–106.

Choiriyah, A. and Qomariasih, N. (2023) "Security Analysis on Websites belonging to the Health Service Districts in Indonesia based on the Open Web Application Security Project (OWASP) Top 10 2021", 2023 International Conference on Information Technology and Computing (ICITCOM), Yogyakarta, Indonesia, pp. 267-272.

Django Project (2026). "Django Documentation". Disponível em: https://docs.djangoproject.com/en/6.0/. Acesso em: 06 jul. 2026.

Kumar, S. A. and Rani, Y. U. (2022) "Implementation and analysis of Web application security measures using OWASP Guidelines", 2022 International Conference on Recent Trends in Microelectronics, Automation, Computing and Communications Systems (ICMACC), Hyderabad, India, pp. 182-187.

OWASP (2025) "About the OWASP foundation". Disponível em: https://owasp.org/about/. Acesso em: 6 jul. 2026.

OWASP (2025). "OWASP Top 10:2025". Disponível em: https://owasp.org/Top10/2025/. Acesso em: 06 jul. 2026.

Python Software Foundation e JetBrains (2024). "Python Developers Survey 2024". Disponível em: https://lp.jetbrains.com/python-developers-survey-2024/. Acesso em: 06 jul. 2026.

Riberu, L. H. and Emanuel, A. W. R. (2024) "Vulnerability Testing and Analysis Using OWASP Top 10 on Academic Information System at University XYZ", 2024 International Conference of Adisutjipto on Aerospace Electrical Engineering and Informatics (ICAAEEI), p. 1–5.

Sharma, M., Khan, M. S. and Singh, J. (2024) “Python & Django the Fastest Growing Web Development Technology”, 2024 IEEE 1st Karachi Section Humanitarian Technology Conference (KHI-HTC), Tandojam, Pakistan, pp. 1–9.

TIOBE Software (2026). “TIOBE Index for July 2026”. Disponível em: https://www.tiobe.com/tiobe-index/. Acesso em: 06 jul. 2026.

Trend Micro (2024). “Ataques cibernéticos praticamente dobram em cinco anos, revela balanço da Trend Micro”. Disponível em: https://www.trendmicro.com/pt_br/about/newsroom/press-releases/2024/2024-03-06.html. Acesso em: 06 jul. 2026.