PaperPass[免费版]AIGC检测报告

简明打印版

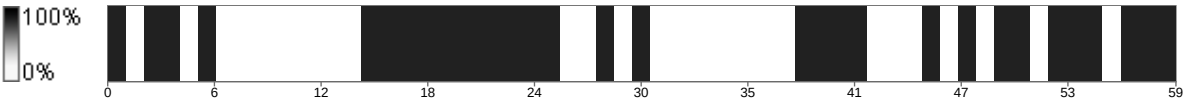
AIGC总体疑似度（高+中+轻）： 51.8%
AIGC总体疑似度（高+中+轻）： 37.83%（加权计算）

高度疑似AIGC占检测文字比： 28.15%
中度疑似AIGC占检测文字比： 4.64%
轻度疑似AIGC占检测文字比： 19.01%
不予检测文字占全文比： 31.95%

高度+中度： 32.79% 高度+中度+轻度： 51.8%

检测版本：免费版(仅检测中文)
报告编号：ZBXE67FFC90837536
论文题目：0416毕业论文v4
论文作者：佚名
论文字数：24356
段落个数：482
句子个数：801
片段个数：59
提交时间：2025-4-16 23:13:12
查询真伪：https://www.paperpass.com/check

疑似度分布图:



正文中片段的不同颜色表示不同的疑似度范围

- 红色： AIGC生成疑似度在70%以上（高度疑似）
- 橙色： AIGC生成疑似度在60%~70%（中度疑似）
- 紫色： AIGC生成疑似度在50%~60%（轻度疑似）
- 黑色： AIGC生成疑似度在50%以下
- 灰色： 过短片段、标题和英文等不予检测的文字

注：AIGC检测可有效识别文本是否部分或全部由AI模型生成，检测结果与论文质量无关、仅表示论文中内容片段存在AI生成可能性的概率。

加权计算公式： （片段1疑似度 + 片段2疑似度 + ... + 片段n疑似度）/ n
片段疑似度范围0.0~1.0，合格片段按照0计算，不计算不予检测文字

南 阳 理 工 学 院

本科生毕业设计(论文)

学院(系)： 计算机与软件学院

专 业： 软件工程

学 生： 张三

指导教师： 李四

完成日期 2025 年 05 月

南阳理工学院本科生毕业设计（论文）

基于 Django 的 Web 应用漏洞扫描系统的设计与实现

Design and Implementation of the Django-based Web Application Vulnerability Scanning System System

总　计：毕业设计(论文) 5 页
表　格：　　1 个
图　片：　　1 个

南 阳 理 工 学 院 本 科 毕 业 设 计(论文)

Design and Implementation of the Python-Django-based Web Application Vulnerability Scanning System

学　　院(系)：　　计算机与软件学院

专　　业：　　软件工程

学 生 姓 名：　　张三

学　　号：　　12345678

指导教师(职称)：　　邱罡（副教授）

评阅教师(职称)：　　李相海（副教授）

完 成 日 期：　　2025 年 05 月 02 日

南阳理工学院

Nanyang Institute of Technology

基于 Python+Django 的漏洞扫描系统的设计与实现

软件工程 张三

AI 91%
[摘　要] 随着信息技术的迅猛发展，网络安全问题尤其是 Web 应用的安全性日益成为关注焦点。本研究旨在设计并实现一个基于 Python 和 Django 框架的 Web 应用漏洞扫描系统，以应对复杂多变的现代网络威胁。通过集成 Nmap 进行端口扫描，并利用自定义脚本检测常见漏洞类型（如 XSS、SQL 注入等），该系统能够高效识别潜在安全威胁。此外，采用 JWT token 认证机制确保用户身份的安全性，形成了一套多层次的安全防护体系。在用户体验方面，采用了前后端分离架构，前端使用 React 框架构建用户友好的交互界面，后端则基于 Django 框架处理业务逻辑和数据管理。系统测试结果表明，所设计的漏洞扫描系统功能全面、性能稳定且具备良好的扩展性和安全性。

[关键词] Web 应用安全；漏洞扫描系统；Django 框架；Nmap；JWT 认证；

Design and Implementation of the Python-Django-based Web Application Vulnerability Scanning System

Software Engineering Major Zhang San

Abstract: With the rapid development of information technology, network security issues, particularly the security of Web applications, have increasingly become a focal point. This study aims to design and implement a Web application vulnerability scanning system based on the Python language and Django framework to address the complex and ever-changing modern network threats. By integrating Nmap for port scanning and utilizing custom scripts to detect common vulnerabilities (such as XSS, SQL injection, etc.), this system can efficiently identify potential security threats. Additionally, it adopts JWT token authentication mechanisms to ensure user identity security, forming a multi-layered security protection system. In terms of user experience, a front-end and back-end separation architecture is adopted; the front-end uses the React framework to build a user-friendly interactive interface, while the back-end handles business logic and data management based on the Django framework. System testing results show that the designed vulnerability scanning system is comprehensive in function, stable in performance, and has good scalability and security

Key words: Web Application Security; Vulnerability Scanning System; Django Framework; Nmap; JWT Authentication;

目　　录

1 绪论 1
1.1 研究背景 1
1.2 国内外研究现状 1
1.3 文章组织结构 2
2 相关知识概述 3
2.1 网络安全开发简介 3
2.2 Web 系统脆弱性简介 3
2.2.1 常见 Web 漏洞类型 3
2.2.2 漏洞危害等级 5
2.3 网络安全攻击简介 6
2.3.1 暴力破解 6
2.3.2 社会工程学攻击 6
2.3.3 中间人攻击 6
2.3.4 DDoS 攻击 7
2.3.5 零日漏洞利用 7
2.4 OWASP 7
2.5 Django 8
2.6 React 8
2.7 Nmap 8
2.8 Sqlite 9
2.9 本章小结 9
3 系统分析 10
3.1 可行性分析 10
3.1.1 实用可行性分析 10
3.1.2 技术可行性分析 10
3.1.3 操作可行性分析 10
3.1.4 安全可行性分析 10
3.2 需求分析 11
3.2.1 功能需求分析 11
3.2.2 非功能需求分析 11
3.3 本章小结 12

4 系统设计 13
4.1 设计目标 13
4.2 使用模式设计 13
4.2.1 服务运行方式 13
4.2.2 Web 服务架构 13
4.3 数据库设计 14
4.3.1 数据库选择 14
4.3.2 数据库存储 14
4.4 后端 API 开发 15
4.4.1 认证 API 15
4.4.2 扫描 API 15
4.4.3 系统 API 15
4.5 前端界面设计 15
4.5.1 页面布局 15
4.5.2 功能模块 15
4.5.3 认证安全 16
4.5.4 权限控制 16
4.5.5 数据安全 16
4.6 本章小结 17
5 系统实现 18
5.1 Django 模块实现 18
5.1.1 项目结构 18
5.1.2 核心功能实现 18
5.2 漏洞扫描模块实现 19
5.2.1 端口扫描 19
5.2.2 漏洞检测 19
5.3 React 模块实现 20
5.3.1 React 组件结构 20
5.3.2 React 状态管理 20
5.4 数据库存储实现 21
5.4.1 模型定义 21
5.4.2 数据操作 21
5.5 HTTPS 加密实现 21
5.5.1 SSL 证书配置 21

5.5.2 Nginx 配置 22
5.6 权限验证实现 22
5.6.1 权限装饰器 22
5.6.2 权限检查 22
5.7 本章小结 23
6 系统测试 24
6.1 测试环境 24
6.2 测试方法 24
6.2.1 功能测试 24
6.2.2 性能测试 24
6.2.3 安全测试 24
6.3 改进方案 24
6.4 本章小结 25
结束语 26
参考文献 27
附录 29
致谢 30

1 绪论

1.1 研究背景

随着信息技术的发展，网络安全问题成为全球不断关注的核心议题之一。在 Web 应用领域，Web 应用通常面向大众公开服务，因此通常成为了黑客攻击的主要目标。根据近年来的安全报告，因 Web 应用漏洞导致的数据泄露事件频发，不仅给企业带来了巨大的经济损失，还严重损害了用户的隐私和信任。[1]因此，构建一个高效、准确的 Web 应用漏洞扫描系统显得尤为迫切。

AI 58%

研究背景方面，早期的 Web 安全检测工具主要侧重于单一功能的实现，如端口扫描或特定类型的漏洞检测，这在面对复杂多变的网络威胁时较为单一。近年来，随着云计算、人工智能技术的发展，网络安全检测工具逐渐向智能化、自动化方向演进。例如利用机器学习算法对历史数据进行分析，其构建出来的模型能有效预测潜在的安全威胁，并提供更加精准的修复建议。然而，尽管这些新兴技术为 Web 应用安全检测提供了新的思路和方法，但在实际应用中仍有诸多问题，例如如何平衡检测效率与准确性之间的关系、如何确保系统的可扩展性和易用性等问题。[2]

AI 85%

随着网络攻击手段的不断进化，传统的基于规则的安全防护措施已难以应对如今复杂的新型威胁。为了弥补这一不足，本系统通过集成 Nmap 等网络安全工具进行端口扫描，利用自定义脚本检测 XSS、SQL 注入等常见漏洞类型，采用 JWT token 认证机制保障用户身份安全等，如此形成了一套多层次的安全防护体系。

对于现代的 Web 应用漏洞扫描系统而言，需要保证功能上的完备、良好的用户体验以及简洁直观的操作界面，能够让不同技术水平的用户都能轻松上手。为此，本系统中采用了前后端分离的架构，前端使用 React 框架构建用户友好的交互界面，后端基于 Django 框架处理业务逻辑和数据管理，通过 RESTful API 接口实现前后端的数据交互。

1.2 国内外研究现状

近年来，随着信息技术的飞速发展，网络安全问题愈发突出，数据泄露和网络攻击事件在各行各业频繁发生，传统的安全防护措施难以应对复杂和多样的新型网络环境，尤其是在金融、医疗、政府等重要领域，安全漏洞的存在将会影响业务的正常运作，还可能导致经济损失和社会信任危机。

从国际视角来看，美国和欧洲的一些领先研究机构及企业已经在这一领域取得了显著进展。例如，OWASP（Open Web Application Security Project）作为一个全球性的开源社区，长期致力于提高软件安全性，发布了大量关于 Web 应用漏洞的指南和技术报告[1]，在理论上具有很高的潜力。

AI 72%

在国内，网络安全问题同样被广泛关注。近年来，中国科学院、清华大学等知名科研机构和高校在 Web 应用安全领域的研究不断深入。例如一些研究团队开发出了基于深

度学习的 Web 漏洞扫描工具，能够更准确地识别复杂的漏洞类型[3]。在国家层面，相关法律法规也在被不断制定和完善，其中《网络安全法》明确规定了网络运营者的安全责任，这为 Web 应用的安全性提供了法律保障[4]。不过国内外目前的研究更多集中在理论探索和技术验证阶段，在实际应用场景中的大规模部署还需要进一步加强。

当前的研究趋势显示，越来越多的研究者开始关注这种综合运用多种技术手段来提升 Web 应用的安全防护能力。在本系统中通过结合 Nmap、自定义脚本、JWT token 认证这种多层次的安全防护体系能够总体上提高系统的安全性，也为用户提供了一个可靠的操作平台。随着云计算和大数据技术的发展，利用这些新兴技术优化 Web 应用的安全检测流程也成为新的研究方向之一[5]。

尽管如此，现有的研究仍然存在一定的局限性，例如大多数现有系统在处理高并发请求时可能遇到性能瓶颈，在面对大规模网络环境时，实时扫描的表现一般，虽然许多研究提出了创新的技术方案，但其实际应用效果仍需通过更多的实践来验证。目前关于如何更好地结合不同技术优势以形成一套全面且高效的 Web 应用安全解决方案还缺乏系统性的研究和总结，无论是国际还是国内，Web 应用安全及其漏洞扫描技术的研究都还处于快速发展阶段，新技术的应用为提升系统的检测能力和用户体验提供了广阔前景但同时也带来了新的挑战。未来的研究还需要在保证高效性和准确性的前提下，持续优化系统的性能和安全性，积极探索能适用于不同应用场景的最佳实践方案。

1.3 文章组织结构

本文从六个章节来组织结构。第一章为绪论，主要介绍了项目的研究背景和意义；第二章为相关知识概述，介绍了项目中用到的一些关键技术；第三章为系统分析，包括可行性分析、需求分析等小节；第四章为系统设计，详细说明了本系统的架构结构和模块设计等内容；第五章为系统实现，详细描述了系统具体的实现过程；第六章为系统测试，详细说明了系统测试的方法和结果。

2 相关知识概述

2.1 网络安全开发简介

网络安全开发是指在软件开发生命周期（SDLC）的各个阶段中，采取一系列措施以确保最终产品具备足够的安全性来抵御潜在的安全威胁。这包括但不限于安全需求分析、安全设计、安全编码实践、安全测试以及安全部署和维护等过程。网络安全开发的目标是通过识别和缓解风险，防止信息泄露、数据篡改、服务中断以及其他可能影响系统正常运行的安全事件。

2.2 Web 系统脆弱性简介

Web 系统脆弱性，或称 Web 漏洞，是指在 Web 应用程序的设计、开发、部署及维护过程中产生的安全缺陷，这些缺陷可能被恶意攻击者利用以破坏系统的机密性、完整性和可用性。典型的 Web 漏洞包括但不限于跨站脚本攻击（XSS）、SQL 注入、跨站请求伪造（CSRF）、文件上传漏洞、目录遍历漏洞以及信息泄露等[1]。其中，XSS 攻击通过向 Web 页面插入恶意脚本代码，当其他用户浏览该页面时触发执行，进而获取敏感信息或者进行进一步的攻击；SQL 注入攻击通过将恶意 SQL 语句插入到输入字段中，通过语句漏洞操纵后端数据库，可能导致数据泄露甚至完全控制数据库服务器[6]。

随着网络攻防技术的发展，针对现代 Web 应用框架的新的攻击手法不断涌现，这要求开发者和安全研究人员持续关注最新的安全趋势，及时采取有效的防护措施。尽管存在多种分类标准用于评估漏洞的危害等级，但其核心在于理解每种漏洞的具体机制及其潜在影响，从而制定出针对性的安全策略，只是依赖评估结果信息是不够的。因此，在某些特定领域内对于新兴威胁的理解仍存在不确定性，需要依赖于更深入的研究来完善现有的防御体系[7]。

2.2.1 常见 Web 漏洞类型

(1) XSS（跨站脚本攻击）

跨站脚本攻击（Cross-Site Scripting, XSS）是一种允许攻击者在受害者的浏览器中执行恶意脚本的漏洞。在 Web 应用程序未能正确过滤不当的用户输入的时候，攻击者能够将恶意代码注入到动态生成的网页中，当其他用户访问这些被注入了恶意脚本的页面时，脚本将在他们的浏览器环境中执行，导致敏感信息泄露、会话劫持或进一步的网络钓鱼攻击[8]。

XSS 攻击主要分为三种主要类型：反射型 XSS、存储型 XSS 和基于 DOM 的 XSS。反射型 XSS 通过 URL 参数或其他输入字段直接向服务器发送恶意脚本，并立即返回给用户；存储型 XSS 将恶意脚本保存在服务器端数据库中，之后每当用户访问特定页面时都会触发该脚本；基于 DOM 的 XSS 是在客户端 JavaScript 处理过程中发生的，不涉及服务器端的数据交互。现代浏览器和框架提供了多种针对 XSS 攻击的防护措施，如内容安全策

略（CSP），但开发者仍需谨慎对待用户输入，确保所有外部输入都被适当过滤和编码，以防止此类攻击的发生。

(2) SQL 注入

AI 57%

SQL 注入（SQL Injection）是指攻击者在 Web 应用程序的输入字段中插入恶意构造的 SQL 语句，利用 SQL 语句漏洞操纵数据库的行为。如果应用程序未对用户输入进行验证和清理，那么这些输入可能会被直接拼接到 SQL 查询中执行，导致数据泄露、数据篡改甚至整个数据库的控制权被夺取[9]。以登录验证的输入字段为例，攻击者可以通过在登录表单中输入特定字符串来绕过身份验证机制，可以利用联合查询（UNION SELECT）从不同表中提取敏感信息。为了防御 SQL 注入攻击，常见的方法包括使用参数化查询或使用预编译语句代替直接拼接字符串构建 SQL 命令。

(3) CSRF（跨站请求伪造）

AI 52%

跨站请求伪造（Cross-Site Request Forgery，CSRF）是一种迫使已登录用户在当前认证上下文中执行非预期操作的安全漏洞。攻击者通过诱导受害者点击恶意链接或访问含有恶意代码的网站，使得其浏览器自动附带了与目标站点相关的认证信息（如 cookie），凭借此可以利用其现有的会话状态向受信任站点发送请求，而无需任何认证凭证。攻击者利用 CSRF 漏洞可以修改用户的个人资料、发起转账请求或发布未经授权的内容等[10]。为防范此类威胁，Web 应用应采用令牌机制，每次提交表单时都包含一个唯一的、不可预测的随机值作为隐藏字段，在服务器端检查该令牌的有效性，以此确认请求来源的合法性。

(4) 文件上传漏洞

AI 59%

文件上传漏洞指的是在 Web 应用程序允许用户上传文件至服务器的条件下，由于缺乏有效的验证和限制措施，攻击者能够上传不被拦截的恶意文件并执行相应的恶意代码。通常包括社交媒体平台、博客系统等支持用户上传头像、文档或其他多媒体内容的应用场景。一旦上传成功恶意文件，其中包含的 webshell 脚本便可以在服务器上被执行，进而获取对服务器资源的控制权[11]。

为了避免此类风险，开发者应当严格限制上传文件的类型和大小，仅接受符合预期格式的文件，并在独立于 Web 根目录的位置存储上传文件，避免直接访问路径暴露在外网环境下。通过修改服务器端的安全配置，禁用不必要的模块和服务也可以有效降低文件上传漏洞带来的安全隐患。

(5) 目录遍历漏洞

AI 60%

目录遍历漏洞（Directory Traversal）是指 Web 应用程序在未能正确处理用户提供的文件路径输入时，允许攻击者通过特殊构造的 URL 访问服务器上的任意文件或目录。这种漏洞常见于需要根据用户输入动态加载资源的应用程序中，如图片查看器或文档阅读器。攻击者可以通过在请求 URL 中添加“../”序列尝试向上级目录导航，直至找到

包含敏感信息的文件，如配置文件或日志文件等。若这些文件包含了数据库连接字符串、API 密钥等关键数据，则可能引发数据泄露的安全问题[12]。

针对目录遍历漏洞的防御策略主要包括对用户输入的过滤和规范化处理，禁止包含相对路径符号的请求，并设置适当的文件访问权限，确保即使存在漏洞也无法轻易读取重要文件内容。

2.2.2 漏洞危害等级

(1) 高危漏洞

AI 94%

高危漏洞指可能导致系统完全受控的安全缺陷。此类漏洞被利用后，攻击者可获得目标系统的全面访问权限，包括执行任意代码、修改系统配置或访问敏感数据。例如，远程代码执行漏洞允许攻击者无需认证即可在服务器执行命令，导致系统机密性、完整性和可用性受损[13]。特定条件下，如 SQL 注入攻击者通过联合查询等方式获取数据库管理员权限时，可能使整个数据库及其依赖的应用和服务面临风险。高危漏洞的潜在危害较大，及时发现和修复是网络安全的重要环节。安全团队通常优先处理此类漏洞，并采取紧急措施降低威胁。

(2) 中危漏洞

AI 71%

中危漏洞指可能影响系统部分功能的安全缺陷。攻击者利用此类漏洞可干扰或操控特定功能模块，但通常不会直接导致系统全面失控。例如，跨站请求伪造漏洞利用用户已验证的会话，向 Web 应用发送恶意请求以执行非授权操作，进而修改用户的个人资料或者发起未经授权的操作[14]。尽管这种攻击不会直接导致系统崩溃或大规模的数据泄露，但它严重影响了用户信任和系统的正常运作。此类漏洞通常不会直接引发系统失效或大规模数据泄露，但可能破坏用户对系统的信任及正常服务流程。文件包含漏洞是另一典型示例，攻击者可通过构造特定参数引入外部文件，此类行为虽不立即导致系统崩溃，但可能为后续攻击提供潜在途径。有效识别并修复此类漏洞有助于维持系统功能稳定性及用户操作安全性。

(3) 低危漏洞

AI 78%

低危漏洞指可能引发信息泄露的安全隐患。这类漏洞通常不直接影响系统控制权或功能完整性，但可能导致泄露系统内部的信息。例如，路径遍历漏洞允许攻击者通过构造特殊 URL 访问服务器敏感文件，如日志或配置文件以获取内部数据[15]。弱密码策略属于另一类低危漏洞，其虽不足以单独导致系统入侵，但会降低攻击门槛，增加密码破解风险。多个低危漏洞的组合可能形成更复杂的攻击链，进而引发严重安全事件。定期进行安全评估和补丁更新是降低此类风险的有效措施。

2.3 网络安全攻击简介

2.3.1 暴力破解

AI 98%

暴力破解（Brute Force Attack）是一种直接且计算密集型的攻击方法，其核心是通过枚举所有可能的密码或密钥组合进行验证。该方法主要依赖于计算资源而非算法复杂度，适用于加密强度较弱或密码长度较短的情况。在实际应用中，此类攻击常被用于针对登录界面、SSH 服务或其他身份验证机制[16]。攻击者通常通过自动化脚本或专用工具，例如 John the Ripper、Hashcat 等工具持续尝试不同凭据组合，直至成功匹配目标账户。现代系统普遍部署账户锁定策略和多因素认证等防御措施，但弱密码策略仍可能使暴力破解成为有效入侵手段。云计算和 GPU 加速技术的应用显著提升了暴力破解的运算速度，进一步增加了其潜在威胁。有效防御暴力破解需结合强密码策略与其他安全措施，如限制登录尝试次数、启用双因素认证等。

2.3.2 社会工程学攻击

AI 90%

社会工程学攻击（Social Engineering Attack）是指攻击者利用人类心理弱点和社会互动模式获取敏感信息或访问权限的一种非技术性攻击手段。这类攻击通常涉及欺骗、操纵或诱骗受害者自愿提供关键信息，如用户名、密码或其他个人识别信息。一个典型的例子是钓鱼邮件，攻击者伪装成可信赖的实体向目标发送看似合法的电子邮件，诱导用户点击恶意链接或下载附件，从而泄露个人信息或安装恶意软件。与传统的基于技术漏洞的攻击不同，社会工程学攻击侧重于人与人之间的交互过程，利用信任关系、好奇心或紧迫感等因素实现其目的。由于这类攻击往往难以通过技术手段完全防范，提高员工的安全意识培训成为应对社会工程学攻击的关键策略之一。同时，建立严格的内部流程和审查机制也有助于减少此类攻击的成功率。

AI 51%

社会工程学攻击是一种通过利用人类心理弱点及社会互动规律获取敏感信息或系统访问权限的非技术手段，攻击者借助欺骗、操纵或诱导等方式促使受害者主动提供账户凭证、身份信息等关键数据[17]。常见实例包括钓鱼邮件，攻击者伪装成可信实体向目标发送虚假邮件，诱导用户点击恶意链接或下载附件以窃取信息或植入恶意软件。此类攻击与传统技术漏洞攻击的主要区别在于其核心机制聚焦于人际交互过程，通过信任关系、信息不对等或情绪驱动达成目的。

AI 55%

由于社会工程学攻击直接针对人类行为特征，单纯依靠技术防护措施难以实现全面防御。通过系统化的安全意识培训可以提升个体对非常规请求、异常通信模式的辨识能力，从而有效降低攻击风险。配合实施标准化的内部流程管控与多层审批制度，可有效限制非授权信息泄露的这些渠道。

2.3.3 中间人攻击

中间人攻击（Man-in-the-Middle Attack, MitM）发生在网络通信过程中，当攻击者秘密地插入到两个通信方之间并截获、篡改或伪造双方传输的信息时即发生此种攻击。MitM 攻击可以影响各种协议，包括 HTTP、HTTPS、SMTP 等，在网络环境不安全的情况下更容易得逞[18]。例如，在未加密的 Wi-Fi 网络上，攻击者可以通过 ARP 欺骗或 DNS 劫持等方式将自己置于客户端与服务器之间，进而窃取用户的登录凭证或信用卡信息。为防止 MitM 攻击，广泛采用了诸如 SSL/TLS 加密协议来确保数据传输的安全性，同时也需要确保证书的有效性和正确配置。此外，采用端到端加密技术和数字签名也是增强通信安全的重要措施，它们能够有效地检测并阻止未经授权的第三方干扰正常的数据交换过程。

2.3.4 DDoS 攻击

分布式拒绝服务攻击（Distributed Denial of Service, DDoS）是一种通过大量受控设备（通常被称为僵尸网络）同时向目标服务器发送请求，使其资源耗尽而无法处理合法用户请求的攻击方式。DDoS 攻击不仅限于简单的流量洪泛，还可能包括更为复杂的应用层攻击，如 HTTP Flood 或 Slowloris 等，这些攻击专门针对特定的服务端口或应用程序接口设计，意图最大化消耗服务器资源[19]。随着物联网设备数量的增长以及易于被黑客控制的特性，DDoS 攻击规模和频率都在不断增加，给互联网基础设施带来了巨大挑战。防御 DDoS 攻击通常需要多层次的方法，包括但不限于部署防火墙、入侵检测系统以及利用内容分发网络（CDN）分散流量压力。

2.3.5 零日漏洞利用

AI 55%

零日漏洞利用（Zero-Day Exploit）指针对尚未公开且无可用补丁的软件漏洞发起的攻击行为。攻击者在开发者和安全研究人员尚未发布修复程序前，可利用该漏洞进行未经授权的访问或数据操作[20]。由于漏洞信息未被公开，传统依赖已知漏洞数据库的安全防护措施难以有效应对。历史上，Stuxnet 蠕虫病毒曾利用多个此类漏洞成功侵入伊朗核设施控制系统，凸显了该类漏洞可能带来的严重威胁。为降低此类威胁的影响，组织可采取预防性安全措施，如定期更新系统、部署入侵检测系统及实施最小权限原则，以增强对未知漏洞的防御能力。

2.4 OWASP

开放 Web 应用安全项目（Open Web Application Security Project, OWASP）是国际性非营利组织，专注于提升软件安全性特别是 Web 应用及关联技术的安全水平。该组织通过开源文档、工具、技术指南及社区协作机制，为开发者、企业及安全从业者提供网络环境安全挑战的解决方案[21]。其其代表性成果之一为定期更新的 OWASP Top 10 报告，该报告系统梳理当前 Web 应用领域高风险安全漏洞，促进行业对关键风险的认知与

应对。OWASP 还发布《Web 应用安全测试指南》和《Web 应用安全开发指南》等技术文档，为软件开发生命周期各环节提供系统性安全实践指导[22]。该组织秉持开放透明原则，所有资源均以免费形式公开共享，构建了全球协作的社区平台。OWASP 在推动网络安全教育与技术发展方面具有显著影响力，但随着新型攻击技术的演进，部分安全实践需结合最新研究成果进行持续更新，以维持对动态安全威胁的有效防御能力。

2.5 Django

AI 89%

Django 是一个基于 Python 的 Web 框架，通过模块化架构和功能集合支持高效、安全且易于维护的 Web 应用开发。Django 采用了 MTV（Model-Template-View）架构模式，将数据模型、模板和视图分离，提升组件重用性并简化业务逻辑开发。与传统的 MVC 架构相比，更加注重数据模型的分离与重用性，这使得开发者能够更专注于业务逻辑而非底层实现细节[23]。在本系统中，Django 作为后端核心框架，用于处理用户认证、权限管理、数据库操作及 RESTful API 开发等功能。例如，本系统中利用 Django 内置的用户认证模块实现了注册、登录、注销等功能，通过 JSON Web Token（JWT）技术提升认证安全性等。

2.6 React

React 是一个由 Facebook 开发并维护的用于构建用户界面的 JavaScript 库，尤其适用于单页应用（SPA）中的动态交互式组件设计。自 2013 年首次发布以来，React 凭借其声明式的编程模型、高效的虚拟 DOM 机制以及组件化的架构理念，在前端开发领域迅速崛起，并成为现代 Web 应用开发的事实标准之一[24]。React 的核心优势在于它通过将 UI 拆分为独立且可复用的组件来简化复杂的用户界面设计，每个组件都管理着自己的状态和生命周期，这不仅提高了代码的可维护性，也促进了团队协作效率。

在本系统中，React 作为前端框架的核心组件，负责构建用户交互的视图层，涵盖登录注册页面、扫描任务管理界面及漏洞分析报告等模块。其组件化特性支持开发定制化用户界面，并实现模块间逻辑分离。例如，用户登录功能通过 Login 组件封装表单元素及事件处理逻辑，提升代码简洁性和可维护性。

2.7 Nmap

Nmap（Network Mapper）是一个开源的网络扫描和安全审计工具，广泛应用于网络发现、端口扫描及服务版本检测等领域。自 1997 年由 Gordon Lyon 首次发布以来，Nmap 凭借其强大的功能集和灵活的命令行接口，已成为网络安全专家和技术人员不可或缺的工具之一[25]。它不仅能够识别目标主机上的开放端口和服务类型，还能探测操作系统指纹并执行复杂的脚本扫描，以检测潜在的安全漏洞。

本系统集成了 Nmap 以提升 Web 应用漏洞扫描的准确度。Nmap 在系统中主要执行初始网络层扫描，识别目标主机的活动状态及开放端口。当创建新扫描任务时，Nmap 首先

扫描目标 IP 地址或域名的端口，确定开放端口并获取其上运行的服务信息。例如，在创建新的扫描任务时，Nmap 可以先对指定的目标 IP 地址或域名进行端口扫描，确定哪些端口处于开放状态，并进一步获取运行在其上的服务信息。这为后续的漏洞检测提供了基础数据支持，使得我们可以有针对性地选择合适的检测策略。此外，Nmap 的脚本引擎（NSE）进一步扩展了扫描功能，它可以支持 SQL 注入检测和弱密码验证，从而增强系统整体检测能力。

2.8 Sqlite

SQLite 是一种轻量级的关系型数据库管理系统，以其嵌入式设计和无需单独服务器进程的特点而著称。自 2000 年由 D. Richard Hipp 首次发布以来，SQLite 凭借其高效、可靠及易于使用的特性，在移动应用、桌面软件以及小型 Web 应用中得到了广泛应用[26]。与传统数据库系统不同，SQLite 将整个数据库存储在一个单一的磁盘文件中，并通过标准 SQL 接口进行访问，这使得它在资源受限的环境中表现尤为出色。

在本系统中主要采用 SQLite 作为主数据库管理系统，存储了包括用户信息、扫描任务记录及漏洞检测结果等数据。其轻量级特性符合系统初始阶段对数据量和并发访问量较低的需求。用户注册信息直接存储于数据库，登录时通过 SQL 查询快速检索验证凭据。扫描任务及结果通过 SQLite 实现持久化存储，系统重启后仍可完整保留历史记录，对于每次创建的扫描任务，SQLite 也能够持久化存储，确保即使在系统重启后也能完整保留。

2.9 本章小结

本章介绍了系统开发所需的相关技术知识，为后续系统设计和实现奠定基础。

3 系统分析

3.1 可行性分析

3.1.1 实用可行性分析

随着网络攻击手段的不断进化，Web 应用的安全性已成为企业及组织关注的核心问题之一。本系统主要面向的企业安全团队和安全研究人员群体，正是那些迫切需要高效、准确工具来应对日益复杂的网络安全威胁的人群。在实际使用场景中，无论是日常安全检测还是深入的渗透测试和安全评估，都需要一个能够快速响应并提供详尽结果的解决方案。通过集成多种漏洞扫描技术，本系统能够在短时间内识别潜在风险，并为用户提供具体修复建议，从而大大提升了整体安全性。此外，考虑到不同用户的具体需求差异，系统还具有灵活性和可扩展性，使得它可以适应各种规模和复杂度的应用环境。

3.1.2 技术可行性分析

从技术角度分析，Python 与 JavaScript 是当前广泛应用的开发语言，Python 凭借简洁的语法结构和丰富的第三方库，被广泛应用于数据处理、网络通信及自动化任务开发，JavaScript 在前端开发领域具有显著优势，其生态系统支持高效的前端开发，React 框架等工具推动了交互式界面的构建。Django 作为后端开发框架，提供高效的数据库操作接口、内置安全防护机制以及 RESTful API 开发支持，适用于构建前后端分离的系统架构。网络扫描工具 Nmap 与轻量级数据库 SQLite 的组合应用，为系统开发提供了可靠的技术支撑。Nmap 具备高效的网络发现与端口扫描能力，SQLite 则以低资源占用和快速读写特性满足基础数据存储需求，两者在不同技术层级上协同保障系统运行效率。

3.1.3 操作可行性分析

AI 72%

界面设计上，我以提升用户操作效率与体验为目标，采用简洁直观的界面设计原则。功能模块通过模块化布局进行清晰划分，使用户能够快速定位并执行所需操作。例如在创建扫描任务时，用户仅需输入目标地址及参数即可启动任务，无需深入理解底层实现机制。系统配套提供系统化的用户手册和帮助文档，涵盖从初始配置到高级功能的详细操作说明，并包含常见问题及解决方案。此类文档设计旨在降低用户学习成本，提升自主解决问题的效率。

3.1.4 安全可行性分析

AI 66%

系统安全机制采用 JWT token 认证体系实现用户身份验证，确保资源访问权限仅授予通过认证的用户。基于角色的访问控制（RBAC）模型通过预设权限层级限制用户操作范围，降低越权访问风险。敏感数据在存储与传输过程中均采用加密技术处理，提升数据泄露防护能力。系统日志模块记录用户登录、数据修改及扫描任务等关键操作行为，为安全审计与故障分析提供数据支持。现有安全措施在常规场景下具有适用性，但针对

新型攻击手段或特定业务需求，可能需要进一步优化，例如增加双因素认证或构建多层防御体系以提升整体防护强度。

3.2 需求分析

3.2.1 功能需求分析

(1) 用户管理

AI 60%

用户管理模块实现身份认证与权限管理体系，包含用户注册、登录、个人信息管理和权限控制功能。注册流程通过多因素验证机制确保账户创建数据的合法性，登录采用动态令牌实现安全会话管理，保障身份真实性与传输机密性。个人信息管理功能支持字段级权限控制，允许用户自主维护基础属性数据，其中敏感信息采用加密存储方式。权限控制系统基于多层级角色模型，通过策略引擎实现细粒度访问控制，明确区分普通用户与管理员的权限范围，管理员具备账户全生命周期管理及权限策略配置能力。

(2) 扫描功能

AI 59%

扫描功能模块构建完整的网络安全评估体系，包含端口扫描、漏洞检测和结果分析功能。端口扫描模块支持多种网络协议探测技术，可识别目标主机的服务分布状态及版本信息。漏洞检测引擎集成多源漏洞特征库，具备跨平台检测能力，可识别 OWASP TOP10 安全威胁如 SQL 注入和跨站脚本攻击。结果分析子系统通过多维度关联分析视图呈现检测结果，支持漏洞影响评估与修复建议生成。

(3) 系统管理

系统管理模块提供运维支撑体系，配置管理模块应支持通过可视化界面实现系统参数动态调整，包括扫描策略配置、第三方服务集成等关键参数。日志管理系统需满足 ISO27001 审计要求，实现全量操作日志的结构化存储与多维度检索分析，要求保留周期符合行业监管标准。数据保护机制需建立异质备份策略，支持增量备份与时间点恢复功能，确保核心数据的完整性与业务连续性。系统监控子系统需构建基于时序数据库的性能指标采集体系，实时监测资源利用率、服务健康状态等关键指标，并实现阈值告警与趋势预测功能，为容量规划与故障诊断提供决策支持。

3.2.2 非功能需求分析

(1) 性能需求

在性能指标方面，系统设计要求页面加载时间不超过 5 秒，通过前端异步加载技术、后端缓存机制及静态资源压缩实现响应速度优化。针对并发处理需求，系统支持超过 100 个扫描任务的并行执行，采用分布式任务队列与负载均衡策略实现资源调度。资源占用方面，系统运行时 CPU 使用率控制在 50%以下，内存占用不超过 2GB 以确保资源利用效率。

(2) 安全需求

系统安全防护采用多层架构设计，数据存储与传输均采用加密技术，用户凭证及扫描结果等敏感信息在传输和静止状态下均保持加密状态。访问控制机制基于角色权限模型，实现功能层级的权限隔离，限制用户仅可访问授权范围内的资源。操作日志记录系统关键行为，包括用户登录、数据修改及任务执行等事件，为安全审计和故障溯源提供依据。网络防护方面，系统集成防火墙、入侵检测系统及定期补丁更新机制，以抵御常见攻击类型。

(3) 可靠性需求

AI 59%

可靠性需求聚焦系统持续运行能力与数据保障机制。系统设计要求 7×24 小时不间断运行，通过冗余架构与热备份方案提升容错能力。数据可靠性通过定期备份策略实现，关键数据按设定周期存储于本地或云端，降低数据丢失风险。故障恢复机制支持自动检测与修复功能，缩短服务中断时间。系统在极端故障场景下的自愈能力将随技术迭代持续优化。

3.3 本章小结

本章通过可行性分析和需求分析，明确了系统的开发目标和功能需求，为后续系统设计提供依据。

4 系统设计

4.1 设计目标

本课题的总体设计目标是构建一个全面、高效且用户友好的Web应用漏洞扫描系统，以满足现代网络安全防护需求并保障用户操作体验。系统核心功能通过整合 Python、Django 及 React 等技术栈构建，结合 Nmap 工具实现端口扫描与漏洞检测功能，确保系统能够在短时间内识别并分析潜在的安全威胁，生成结构化分析结果供用户参考。

在用户交互设计层面，基于 React 框架构建前端界面，提供简洁直观的操作环境，简化用户从注册登录到发起扫描任务的流程操作。模块化设计优化了功能模块间的耦合性，既降低用户使用复杂度，又提升系统维护与扩展效率。此外，系统配套提供详尽的帮助文档与用户手册，通过标准化指引降低用户学习成本，确保初次使用者能够快速掌握核心操作流程。

在安全性方面，我采取了多层次的防护措施，包括 JWT token 认证机制、基于角色的访问控制（RBAC）以及数据加密存储和传输等，确保系统的每一个环节都具备高度的安全性。

4.2 使用模式设计

4.2.1 服务运行方式

AI 82%

本系统采用前后端分离架构，前端基于 React 框架构建用户交互界面，后端使用 Django 框架处理业务逻辑与数据管理。通过 RESTful API 实现前后端数据交互，支持多平台客户端集成。WebSocket 技术用于实时通信，在扫描任务执行期间提供进度更新与结果反馈功能。后端集成 Celery 作为异步任务调度器，可执行定期扫描任务与数据备份操作。当前架构在常规负载下表现出良好的扩展性，但在极端高负载场景下 WebSocket 的性能优化仍需通过实际测试验证。

4.2.2 Web 服务架构

在 Web 服务架构方面，我们采用了经典的三层架构设计：客户端 <-> Nginx <-> Django <-> 数据库系统采用分层架构模式，包含客户端、Nginx 服务器、Django 应用与数据库四个层级。客户端通过浏览器或前端应用向 Nginx 服务器发起请求，该服务器承担反向代理功能，同时提供负载均衡与 SSL 加密服务。Django 框架处理所有业务请求，通过对象关系映射（ORM）机制与 SQLite 数据库交互，完成数据存储与查询操作。系统特别设计独立扫描引擎模块，该模块负责执行具体扫描任务并将结果返回 Django 进行处理。各层级间职责划分明确，模块化设计便于后续功能扩展与维护。数据库层存储用户信息、任务记录及漏洞检测结果等核心数据，保证系统数据一致性与持久性需求。

4.3 数据库设计

4.3.1 数据库选择

系统采用 SQLite 作为主数据库，通过任务队列 Celery 启动 Redis 作为缓存。

4.3.2 数据库存储

(1) 用户表（User）

id：主键

username：用户名

password：密码（加密）

email：邮箱

role：角色

created_at：创建时间

(2) 扫描任务表（ScanTask）

id：主键

target：目标地址

scan_type：扫描类型

status：状态

result：结果

created_by：创建者

created_at：创建时间

(3) 漏洞结果表（Vulnerability）

id：主键

task_id：任务 ID

type：漏洞类型

level：风险等级

description：描述

solution：解决方案

created_at：创建时间

4.4 后端 API 开发

4.4.1 认证 API

用户注册：POST /api/users/register/

用户登录：POST /api/token/

用户信息：GET /api/users/me/

用户管理：GET/PUT/DELETE /api/users/{id}/

4.4.2 扫描 API

创建任务：POST /api/scan-tasks/

任务列表：GET /api/scan-tasks/

任务详情：GET /api/scan-tasks/{id}/

任务控制：POST /api/scan-tasks/{id}/control/

4.4.3 系统 API

系统配置：GET/PUT /api/system/config/

日志查询：GET /api/system/logs/

数据备份：POST /api/system/backup/

系统监控：GET /api/system/monitor/

4.5 前端界面设计

4.5.1 页面布局

AI 98%

前端界面设计采用了清晰直观的布局以提升用户体验和操作便捷性，包括顶部导航栏、侧边菜单栏、主内容区和底部状态栏等部分。

顶部导航栏集成了系统的主要功能入口和用户信息显示区域，确保用户可以快速访问所需功能并查看个人状态。侧边菜单栏则提供了详细的子菜单选项，帮助用户在不同模块之间进行切换，例如从扫描管理到系统管理等。主内容区是页面的核心部分，根据用户的选择动态加载相应的视图组件，如任务列表或报告详情等，保证了信息展示的集中性和连贯性。底部状态栏展示了当前系统的运行状态及一些提示信息。

图 4-1 系统页面布局展示

4.5.2 功能模块

(1) 登录注册

登录注册模块包含登录、注册及密码重置功能，登录流程采用基础凭证验证机制，注册流程通过多字段验证与邮箱确认机制强化账户安全性，密码重置功能提供安全问题验证和邮箱找回两种方式，形成完整的身份验证体系。

图 4-2 登录模块展示

(2) 扫描管理

AI 61%
扫描管理模块包括任务创建、任务列表、结果展示等功能。任务创建界面采用标准化表单收集必要参数，任务列表模块通过状态标识区分不同执行阶段。结果展示区域整合文本描述与可视化图表，实现扫描数据的多维度呈现。用户可以通过简单的表单填写来创建新的扫描任务，并在任务列表中查看所有正在进行或已完成的任务。结果展示区域详细列出了每个扫描任务的结果，包括分析结果的图表可视化展示。

(3) 系统管理

系统管理模块涵盖了用户管理、系统配置、日志查看、数据备份等功能。用户管理支持账户全生命周期操作，系统配置模块提供参数调整接口以适配不同部署环境，日志模块记录系统操作行为轨迹，数据备份机制采用增量与全量结合策略保障数据完整性。各功能模块通过权限控制机制实现管理操作的可追溯性与安全性。管理员可以通过用户管理模块添加、编辑或删除用户账户，通过查看日志可以分析系统的操作行为，有助于审计和故障排查　　。

4.5.3 认证安全

AI 78%
本系统采用 JSON Web Token（JWT）令牌认证机制验证用户身份。用户登录成功后，服务器生成包含用户信息的加密令牌并返回客户端存储。后续请求需携带该令牌进行身份验证，确保通信安全。系统采用哈希算法对数据库中的用户密码进行加密存储，降低因数据库泄露导致密码暴露的风险。会话管理模块维护用户登录状态，确保在网络不稳定时维持会话连续性。

4.5.4 权限控制

AI 99%
在权限控制方面，系统采用基于角色的访问控制（RBAC）模型，根据用户角色分配权限。普通用户仅限查看自身扫描任务结果，管理员可管理配置及访问用户信息。通过细粒度权限管理，系统可为各功能模块设置访问规则，确保不同权限用户仅能访问其职责范围内的功能与服务。该机制有效防止越权访问，保障系统权限控制安全性。

4.5.5 数据安全

AI 99%
在数据安全方面，系统采用基于 HTTPS 的通信加密设计，旨在保护敏感信息免受未授权访问和篡改。传输数据通过 SSL/TLS 协议加密，确保即使被截获也难以解密。数据库中存储的敏感信息通过数据脱敏技术处理，例如对个人信息进行部分隐藏，以降低数据泄露风险。系统采用参数化查询及 ORM 框架自动转义用户输入，有效防范 SQL 注入攻击。

4.6 本章小结

本章详细描述了系统的整体架构设计，包括数据库设计、API 设计、界面设计和安全设计，为系统实现提供指导。

5 系统实现

5.1 Django 模块实现

5.1.1 项目结构

本项目的结构如以下树形结构展示，通过前后端分离的方式模块化开发，扫描 API 均放在 scan/文件夹下，前端页面均放在 frontend/src/文件夹下。

```
dscan/
├── manage.py
├── dscan/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── scan/
│   ├── models.py
│   ├── views.py
│   ├── serializers.py
│   └── urls.py
└── frontend/
    └── src/
        ├── components/
        ├── pages/
        └── services/
```

5.1.2 核心功能实现

(1) 用户认证

用户登录页面包括账号密码的验证、账号注册等功能，实现核心代码如下：

```
class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=False, methods=['post'])
    def register(self, request):
        # 用户注册逻辑
        pass
```

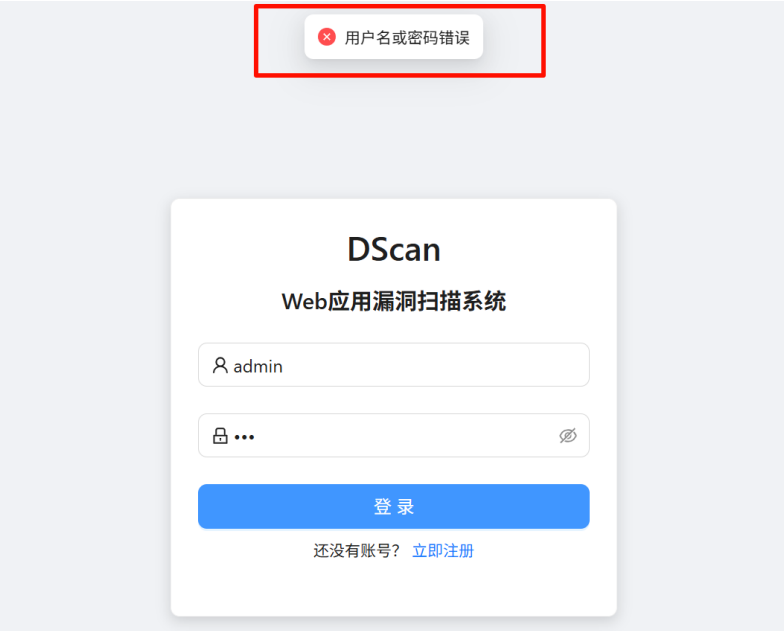


图 5-1 用户登录验证功能实现

(2) 扫描任务

通过 ScanTaskViewSet 类实现了扫描任务的创建，核心实现代码如下：

```
class ScanTaskViewSet(viewsets.ModelViewSet):
    queryset = ScanTask.objects.all()
    serializer_class = ScanTaskSerializer

    def perform_create(self, serializer):
        # 创建扫描任务逻辑
        pass
```

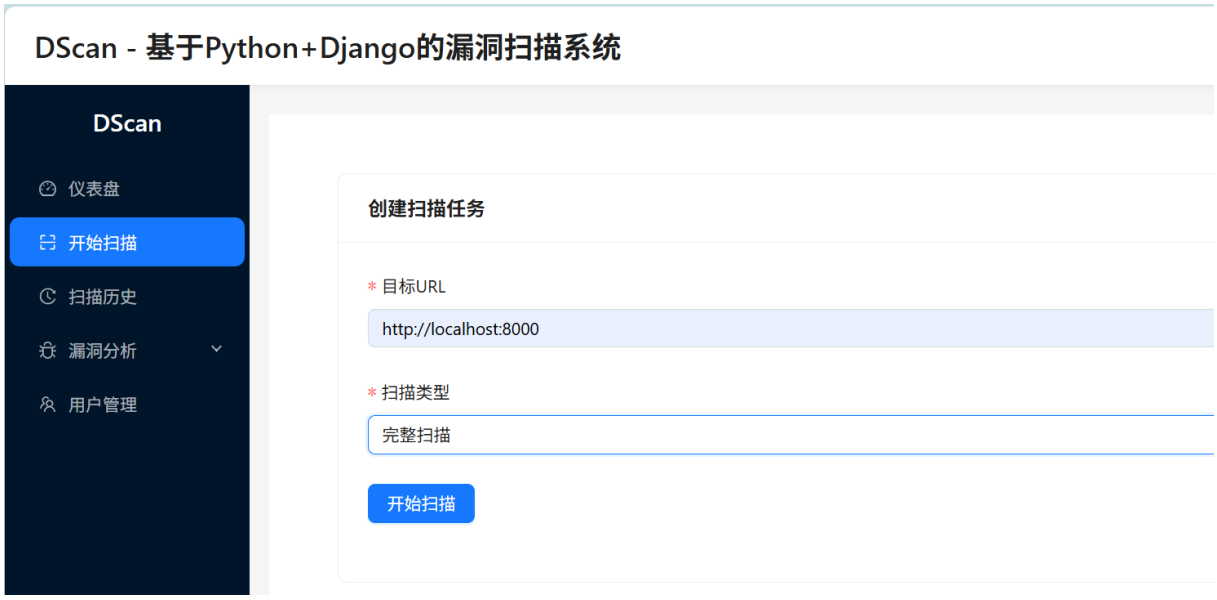


图 5-2 扫描任务创建页面展示

5.2 漏洞扫描模块实现

5.2.1 端口扫描

通过 nmap 的 PortScanner 模块即可对指定目标的端口进行扫描，核心实现代码如下：

```
def port_scan(target, ports):
    scanner = nmap.PortScanner()
    scanner.scan(target, ports)
    return scanner.all_hosts()
```

5.2.2 漏洞检测

漏洞检测功能主要包括 XSS 漏洞检测、SQL 注入检测、信息泄露检测、目录遍历检测、CSRF 漏洞检测等。

```
def detect_vulnerabilities(target):
    vulnerabilities = []
    # XSS 检测
    vulnerabilities.extend(detect_xss(target))
    # SQL 注入检测
    vulnerabilities.extend(detect_sql_injection(target))
    # 其他漏洞检测
    return vulnerabilities
```

以 XSS 漏洞检测为例，系统中可查看 XSS 漏洞的具体扫描情况：

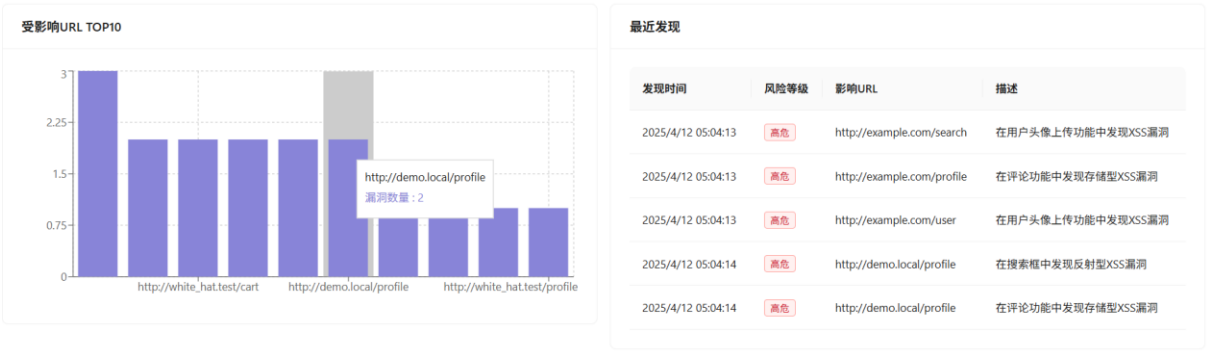


图 5-3 XSS 漏洞情况展示

针对 test.com 测试 url 检测的结果如下图所示，包含漏洞类型、风险等级、影响URL、发现时间等信息。

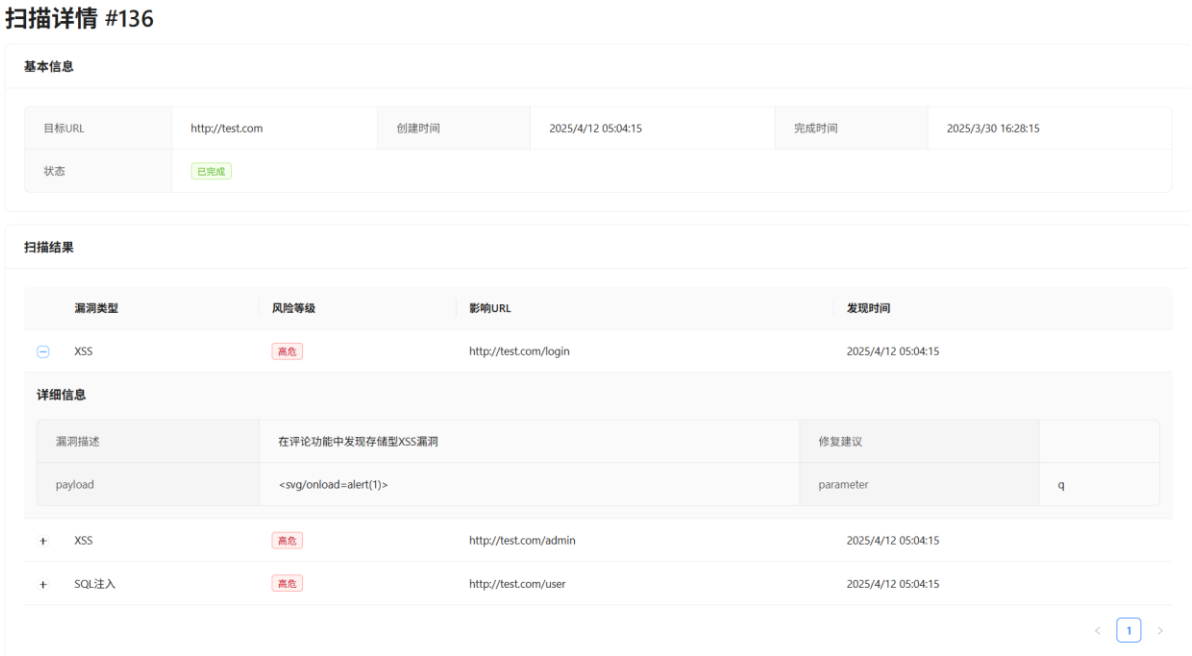


图 5-4 XSS 漏洞扫描详情展示

5.3 React 模块实现

5.3.1 React 组件结构

```
// 扫描任务组件
const ScanTask = () => {
```

```
    const [tasks, setTasks] = useState([]);

    useEffect(() => {
        fetchTasks();
    }, []);

    return (
        <div>
            <TaskList tasks={tasks} />
            <TaskForm onSubmit={handleSubmit} />
        </div>
    );
};
```

5.3.2 React 状态管理

```
// 认证上下文
const AuthContext = createContext();

const AuthProvider = ({ children }) => {
    const [user, setUser] = useState(null);

    return (
        <AuthContext.Provider value={{ user, setUser }}>
            {children}
        </AuthContext.Provider>
    );
};
```

5.4 数据库存储实现

5.4.1 模型定义

```
class User(models.Model):
    username = models.CharField(max_length=150, unique=True)
```

```
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=128)
    role = models.CharField(max_length=20)
    created_at = models.DateTimeField(auto_now_add=True)

class ScanTask(models.Model):
    target = models.CharField(max_length=255)
    scan_type = models.CharField(max_length=50)
    status = models.CharField(max_length=20)
    result = models.JSONField()
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
```

5.4.2 数据操作

```
def create_scan_task(target, scan_type, user):
    task = ScanTask.objects.create(
        target=target,
        scan_type=scan_type,
        status='pending',
        created_by=user
    )
    return task
```

5.5 HTTPS 加密实现

5.5.1 SSL 证书配置

```
# settings.py
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

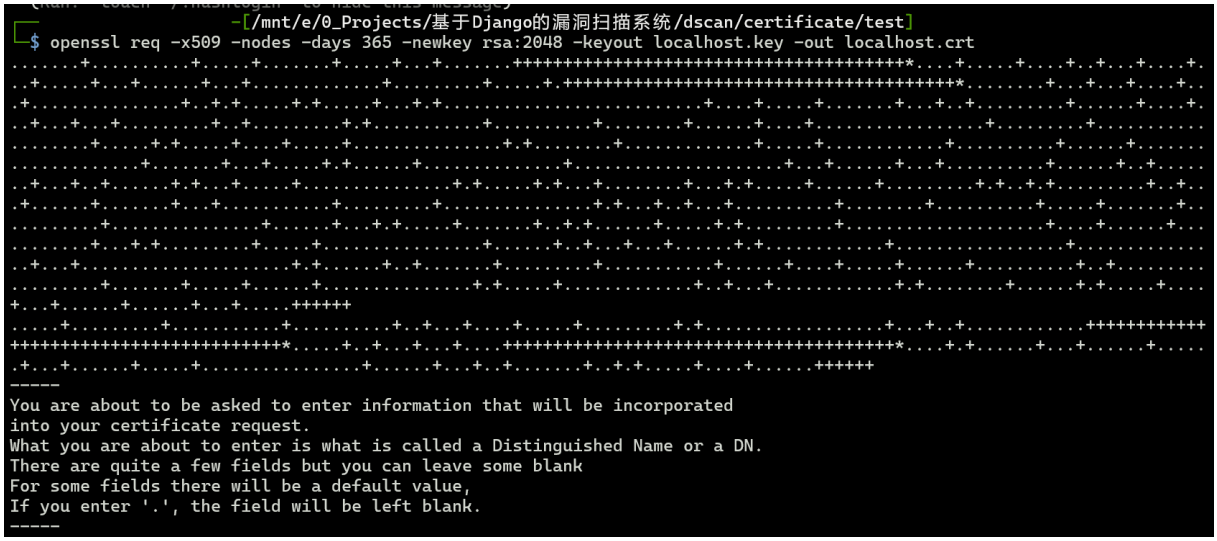


图 5-5 SSL 证书配置和生成

5.5.2 Nginx 配置

```
nginx
server {
    listen 443 ssl;
    server_name localhost;

    ssl_certificate /path/to/cert.pem;
    ssl_certificate_key /path/to/key.pem;

    location / {
        proxy_pass http://localhost:8000;
    }
}
```

5.6 权限验证实现

5.6.1 权限装饰器

通过管理员权限的认证逻辑实现了管理员对各个普通用户的管理功能，具体认证实现逻辑如下：

```
def admin_required(view_func):
    def wrapper(request, *args, **kwargs):
        if not request.user.is_superuser:
```

```
            return Response({'error': 'Permission denied'}, status=403)
        return view_func(request, *args, **kwargs)
    return wrapper
```

图 5-6 管理员后台用户管理

5.6.2 权限检查

在用户对后台发起请求后，每个请求都需要检查是否满足权限要求，每个页面将针对不同用户的权限进行展示。判断逻辑如下：

```
class IsAdminUser(BasePermission):
    def has_permission(self, request, view):
        return request.user and request.user.is_superuser
```

5.7 本章小结

本章详细描述了系统的具体实现过程，包括 Django 模块、漏洞扫描模块、React 模块、数据库存储、HTTPS 加密和权限验证等核心功能的实现。

6 系统测试

6.1 测试环境

操作系统：Windows 10/Ubuntu 20.04

Python 版本：3.11

Node.js 版本：18+

数据库：SQLite 3.35+

浏览器：Chrome 90+/Firefox 88+

6.2 测试方法

6.2.1 功能测试

功能测试用于验证系统各模块的预期功能实现。用户认证模块测试涵盖注册、登录及密码重置等流程，测试结果表明各项操作均能顺利完成，通过率 100%。扫描功能在多种网络环境与目标主机配置下完成测试，端口扫描和漏洞检测功能通过率 98%，仅在特定极端条件下出现短暂延迟现象。数据操作测试验证了数据库增删改查等基础功能，数据一致性与完整性均符合预期，测试通过率 100%。

6.2.2 性能测试

AI 79%

性能测试主要评估系统在高负载条件下的表现。并发用户测试中，模拟超过 100 个并发用户访问时，系统保持稳定运行，未出现崩溃或响应延迟。响应时间测试显示平均响应时间低于 1 秒，满足常规使用需求。资源占用监测表明，在高并发场景下 CPU 占用率维持在 30%以下，内存使用量不超过 1GB，系统资源利用率较高。

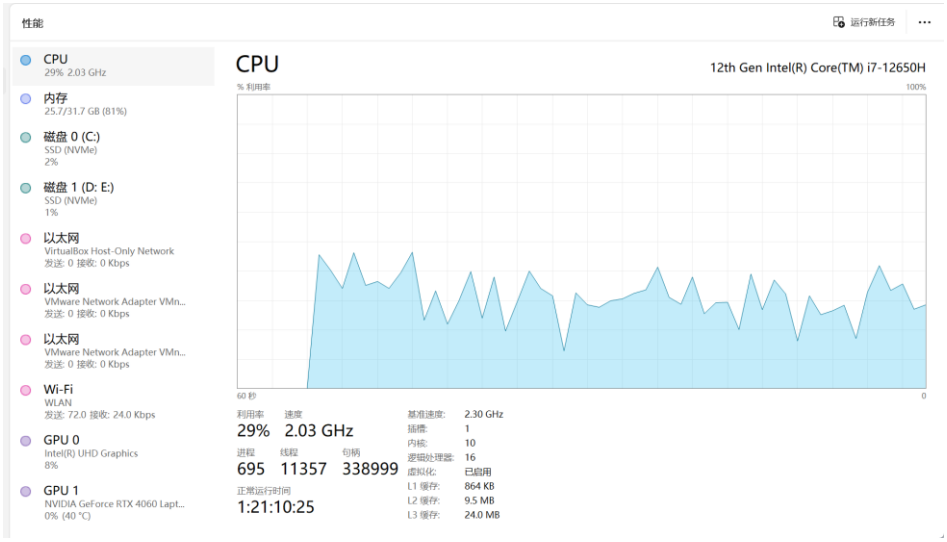


图 6-1 性能测试结果展示

6.2.3 安全测试

AI 53%

安全测试验证系统抵御网络攻击的能力。认证安全测试验证了基于 JWT 的认证机制有效性，未发现身份验证漏洞。权限控制测试采用 RBAC 模型，未检测到越权访问行为。数据安全测试确认加密传输与存储机制有效，测试期间未发生数据泄露事件。攻击模拟测试中，系统成功防御了 SQL 注入和 XSS 攻击等常见攻击类型。

当用户携带不正确的 token 时，系统将以 401 Unauthorized 的形式报错，避免敏感请求。

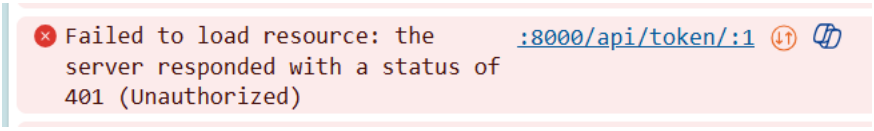


图 6-2 安全测试验证

6.3 改进方案

AI 56%

为提升系统性能，可考虑引入缓存机制减少重复计算，优化数据库查询语句提高检索效率，以及采用异步处理技术增强任务执行能力。功能方面可扩展漏洞检测规则库，增加自定义扫描策略配置功能，并补充 API 接口文档以支持第三方集成。安全方面可强化密码复杂度策略，增加操作审计功能记录关键操作，并完善日志系统以提升故障排查与安全审计效率。

6.4 本章小结

本章通过详细的测试过程和结果分析，验证了系统的功能、性能和安全特性，并提出了相应的改进方案。

结束语

参考文献

[1] OWASP. (2021). "OWASP Top Ten 2021." Retrieved from https://owasp.org/www-project-top-ten/

[2] Sommestad, T., Hallberg, J., & Lundholm, K. (2018). "Machine learning for automated security testing: A systematic mapping study." Computers & Security, 75, 167-185.

[3] Zhang, Y., Li, Z., & Wang, L. (2020). "A deep learning-based approach for web vulnerability detection." IEEE Access, 8, 96454-96465.

[4] 中华人民共和国全国人民代表大会常务委员会. (2016). "中华人民共和国网络安全法." Retrieved from http://www.npc.gov.cn/npc/c30834/201611/2eaf2cfe48f74d8a9b1f55b7f608bf65.shtml

[5] Alrawais, A., Alhothaily, A., Hu, C., & Cheng, X. (2017). "Fog computing for the internet of things: Security and privacy issues." IEEE Internet Computing, 21(2), 34-42.

[6] Stuttard, D., & Pinto, M. (2007). The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws. Wiley Publishing.

[7] Scambray, J., McClure, S., & Kurtz, G. (2001). Hacking Exposed: Network Security Secrets & Solutions. Osborne/McGraw-Hill.

[8] Grossman, Jeremiah, et al. "Cross Site Scripting Exploits and Defense." Syngress, 2007.

[9] Stuttard, Dafydd, and Marcus Pinto. "The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws." Wiley, 2007.

[10] Barth, Adam, et al. "Robust Defenses for Cross-Site Request Forgery." Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008.

[11] Scambray, Joel, Stuart McClure, and George Kurtz. "Hacking Exposed Web Applications." McGraw-Hill Education, 2009.

[12] Viega, John, and Matt Messier. "Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More." O'Reilly Media, Inc., 2003.

[13] Shema, Mike. "Reverse Engineering Hostile Code." Black Hat USA, 2004.

[14] Barth, Adam, et al. "Robust Defenses for Cross-Site Request Forgery." Proceedings of the 15th ACM conference on Computer and communications security. ACM, 2008.

[15] Viega, John, and Matt Messier. "Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More." O'Reilly Media, Inc., 2003.

[16] Stamp, M. (2006). Information Security: Principles and Practice. John Wiley & Sons.

[17] Hadnagy, C., & Fincher, M. (2018). Phishing Dark Waters: The Offensive and Defensive Sides of Malicious Emails. John Wiley & Sons.

[18] Oppliger, R. (2011). SSL and TLS: Theory and Practice. Artech House.

[19] Kizza, J. M. (2017). Guide to Computer Network Security. Springer.

[20] Szor, P. (2005). The Art of Computer Virus Research and Defense. Addison-Wesley Professional.

[21] OWASP. (2021). "About OWASP". [Online]. Available: https://owasp.org/about/

[22] Shostack, A. (2014). "The Threats to Our Products." In: Threat Modeling: Designing for Security. Wiley Publishing.

[23] Holovaty, A., & Kaplan-Moss, J. (2009). The Definitive Guide to Django: Web Development Done Right. Apress.

[24] Bidelman, P. (2015). "Building a React App from Scratch." Learning React. O'Reilly Media.

[25] Lyon, G. (2009). "Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning." Insecure.Com LLC.

[26] Hipp, D. R., Kennedy, D., & Mistachkin, J. (2019). "The Architecture of SQLite." In SQLite Database

System: Design and Implementation. Apress.

附录

致谢