Computer Science and Application 计算机科学与应用, 2026, 16(5), 443-458
Published Online May 2026 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2026.165196

Hans汉斯

PkcScan：基于Python的轻量级Web漏洞扫描框架设计与实现

龚云平[1,2]，武春岭[1,2*]，熊朝政[1]

[1]重庆电子科技职业大学人工智能与大数据学院(网络空间安全学院)，重庆
[2]重庆电子科技职业大学量子多体计算与模拟重庆市科学技术局重点实验室，重庆

收稿日期：2026年4月23日；录用日期：2026年5月22日；发布日期：2026年5月29日

摘　要

随着Web应用的广泛普及，Web漏洞逐渐成为网络攻击的主要入口，漏洞检测技术的重要性日益凸显。然而，在实际检测过程中，POC (概念验证)的开发与应用始终面临一系列挑战，例如开发过程繁琐、逻辑分散、标准缺失、共享困难以及复用性差等问题，严重限制了检测工具的效率与覆盖面。为应对上述问题，文章提出并实现了一种基于Python语言的可扩展Web漏洞扫描框架PkcScan，旨在构建一个支持高效POC开发、灵活调用与生态协同的轻量级漏洞检测平台。该框架采用模块化设计理念，将漏洞检测流程划分为多个可插拔的功能组件，具备良好的可维护性与可扩展性。系统引入“指纹识别 + POC验证”双引擎机制，通过先识别目标系统的指纹信息，再依据指纹特征选择合适的POC进行漏洞验证，提升了检测的准确性与效率。在POC编写方面，PkcScan支持通过标准化JSON结构定义多阶段、多请求的检测逻辑，便于开发者以结构化方式编写和组织复杂检测流程，适配模糊测试、特征识别和精准验证等多种检测模式。此外，PkcScan提供POC的自动打包、分发与一键导入功能，简化了POC的管理流程，降低了技术门槛，为构建分布式POC共享生态体系打下了基础。实验部分选取Pikachu开源靶场进行验证，结果显示，PkcScan可成功识别并分析多个典型漏洞，其误报率相比传统扫描工具降低了32%，整体检测效率提升达到80%。兼容性测试覆盖OWASP 2021 Top 10中的常见漏洞类型，理论支持超过98%的Web漏洞扩展检测，具备高度通用性与适应性。

关键词

Web安全，轻量化可扩展框架，POC共享，多阶段检测

PkcScan: Design and Implementation of a Lightweight Web Vulnerability Scanning Framework Based on Python

Yunping Gong[1,2], Chunling Wu[1,2*], Chaozheng Xiong[1]

*通讯作者。

文章引用: 龚云平, 武春岭, 熊朝政. PkcScan: 基于 Python 的轻量级 Web 漏洞扫描框架设计与实现[J]. 计算机科学与应用, 2026, 16(5): 443-458. DOI: 10.12677/csa.2026.165196

[1]School of Artificial Intelligence and Big Data (School of Cyberspace Security), Chongqing Polytechnic University of Electronic Technology, Chongqing
[2]Chongqing Key Laboratory of Quantum Many-Body Computation and Simulation, Chongqing Polytechnic University of Electronic Technology, Chongqing

Received: April 23, 2026; accepted: May 22, 2026; published: May 29, 2026

Abstract

With the widespread popularity of Web applications, Web vulnerabilities have gradually become the main entry point for cyber attacks, and the importance of vulnerability detection technology has become increasingly prominent. However, in the actual detection process, the development and application of POCs (Proof of Concept) have always faced a series of challenges, such as cumbersome development processes, scattered logic, lack of standards, difficulty in sharing, and poor reusability, which seriously limit the efficiency and coverage of detection tools. To address these issues, this paper proposes and implements a lightweight and extensible Web vulnerability scanning framework, PkcScan, based on the Python language, aiming to build a platform that supports efficient POC development, flexible invocation, and ecological collaboration. The framework adopts a modular design concept, dividing the vulnerability detection process into multiple pluggable functional components, with good maintainability and extensibility. The system introduces a dual-engine mechanism of "fingerprint recognition + POC verification", which first identifies the fingerprint information of the target system and then selects the appropriate POC based on the fingerprint features for vulnerability verification, improving the accuracy and efficiency of detection. In terms of POC writing, PkcScan supports defining multi-stage and multi-request detection logic through a standardized JSON structure, facilitating developers to write and organize complex detection processes in a structured manner, and adapting to various detection modes such as fuzz testing, feature recognition, and precise verification. Additionally, PkcScan provides automatic packaging, distribution, and one-click import functions for POCs, simplifying the management process of POCs, lowering the technical threshold, and laying the foundation for building a distributed POC sharing ecosystem. The experimental part selected the Pikachu open-source target field for verification, and the results showed that PkcScan could successfully identify and analyze multiple typical vulnerabilities, with a false positive rate 32% lower than that of traditional scanning tools, and an overall detection efficiency improvement of 80%. Compatibility tests covered common vulnerability types in OWASP 2021 Top 10, with theoretical support for over 98% of Web vulnerability extended detection, demonstrating high universality and adaptability.

Keywords

Web Security, Lightweight Extensible Framework, POC Sharing, Multi-Stage Detection

1. 引言

1.1. 研究意义

随着信息技术的迅猛发展，Web 应用得到广泛使用，其面临的安全威胁也日益严峻。全球网络安全

形势不断恶化，奇安信网络安全威胁 2024 年度报告显示，2024 年在漏洞的利用情况较 2023 年有所回落，但漏洞军火商的活跃、AI 大模型的出现以及整体网络攻防技术的提升，导致漏洞的开发速度加快，使用漏洞的真实攻击案例增多，发现与修补漏洞的重要性也逐渐提升[1]-[4]。

当前主流的 Web 漏洞扫描技术主要呈现两种技术路线[5] [6]：传统型工具基于规则库进行特征匹配，通过模拟攻击流量进行漏洞探测，这类工具(如 AWVS、AppScan)虽具备基础检测能力，但存在明显局限：首先，其内置规则库依赖已知漏洞特征，难以发现逻辑漏洞和 0-day 漏洞；其次，扫描过程会产生大量无效流量，导致检测效率低下且易触发防护机制。另一类 POC 验证型工具(如 Nuclei、Xray)采用模块化漏洞验证脚本，通过精确构造攻击载荷实现漏洞确认，其误报率显著降低，但面临脚本开发成本高、私有 POC 流通受限、漏洞覆盖不全等问题。

针对当前 Web 漏洞检测面临效率低、POC 复用性差、二次开发困难等核心问题，本研究设计的轻量级可扩展漏洞扫描框架，具有如下重要意义：

1) 提升漏洞检测的精准性与效率：传统扫描工具依赖静态规则库，难以应对复杂多变的 Web 漏洞，而基于 POC 的检测虽然准确性高，但开发成本大、覆盖率有限。本研究提出的“指纹 + POC”双引擎机制，通过智能指纹预筛选减少无效探测，结合标准化 POC 精准验证，能够在保证低误报率(实验显示降低 32%)的同时，显著提升扫描效率(实验提升 80%)，为自动化安全测试提供可靠技术支撑。

2) 推动 POC 共享生态的构建：现有漏洞检测工具的 POC 往往封闭、碎片化，安全研究人员难以复用或协作。本研究通过 JSON 标准化 POC 模板，支持多阶段请求、条件判断等复杂逻辑定义，并实现 POC 的一键打包、分发与导入功能，降低开发门槛。这一设计有助于形成开放的 POC 共享社区，促进漏洞检测能力的快速迭代与协同进化，应对日益增长的 0-day/N-day 威胁。

3) 促进轻量化安全检测技术的普及：商业扫描工具(如 AWVS、Burp Suite)通常资源占用高、扫描时间长、二次开发困难，不适合中小型企业或个人研究者。本研究基于 Python 实现轻量化框架，支持模块化扩展与低资源消耗运行，为缺乏专业安全团队的组织提供低成本、高可用、易开发的漏洞检测框架，推动安全检测技术的普惠化发展。

1.2. 研究目的

本研究旨在针对当前 Web 漏洞检测领域中普遍存在的效率低下、误报率高、POC (概念验证)开发门槛高、复用性差以及共享困难等核心问题，设计并实现一种高效、精准、具备良好可扩展性的综合解决方案。为此，本文构建了一个轻量级且可扩展的漏洞扫描框架，采用模块化架构与插件化机制，充分解耦核心检测引擎与功能组件，支持多线程并发执行以提升整体处理效率，并通过提供标准化 API 接口，便于功能拓展与二次开发，确保框架在资源受限环境中的稳定运行。

该框架引入“指纹识别 + POC 验证”双引擎协同检测机制，设计高性能指纹识别算法实现快速系统特征匹配，配合灵活可配置的 POC 执行引擎，支持复杂、多阶段、多请求的漏洞验证流程，构建智能化引擎协作工作流。同时，系统配备可视化界面，提供运行状态监控与检测结果展示，提升用户操作体验。

在 POC 管理方面，研究建立了标准化的 POC 开发与维护体系，设计基于 JSON 格式的描述规范以支持多阶段检测逻辑的定义，提供 POC 开发辅助工具和调试环境，支持 POC 脚本的一键打包、导入与共享，从而显著提升开发效率与复用价值。

最终，本文通过在典型靶场环境下的实验测试，从检测准确率、误报率与运行效率等维度对所构建框架进行综合评估，并与现有主流工具进行对比验证，确认其在扩展性、兼容性及实用性方面具备显著优势。研究期望通过上述技术路径，为 Web 漏洞检测提供一个创新、高效且可持续演进的实用解决方案。

2. 相关技术介绍

2.1. Web 安全漏洞概述

Web 安全漏洞[7] [8]主要包括以下类型：1) 注入类漏洞，如 SQL 注入(通过未过滤的用户输入操纵数据库)和命令注入(如系统命令执行)；2) 客户端漏洞，如 XSS (跨站脚本攻击)和跨站请求伪造；3) 逻辑缺陷，如越权访问(水平或垂直权限绕过)和业务流劫持(如支付金额篡改)；4) 配置缺陷，如敏感信息泄露(如.git 目录暴露)和 CORS 错误配置。漏洞的成因[9]通常包括开发阶段的输入验证缺失(如未使用参数化查询导致 SQL 注入)和安全防护机制失效(如 WAF 规则绕过引发的远程代码执行漏洞)。这些漏洞可能导致严重后果，根据 IBM《2024 数据泄露成本报告》，单次数据泄露的平均损失高达 488 万美元，且仍在逐年增高。

2.2. 漏洞检测技术

在安全检测技术中，指纹识别[10]通过分析响应特征(如 Header、Cookie 关键字和 MD5 哈希)来识别组件或框架，能够快速梳理资产，但无法有效检测业务逻辑漏洞；模糊测试[11]则通过生成随机或变异的数据(例如 SQLMap 的布尔盲注探测)注入目标参数，虽然具备发现未知漏洞的潜力，但容易产生大量无效流量，并可能触发 Web 应用防火墙(WAF)；精准验证[12]基于 POC 构造特定攻击载荷(如 Nuclei 的 CVE-2023-23752 模板)，能够将误报率控制在 5%以下，但 POC 开发周期较长且漏洞覆盖面有限；混合检测[13]则结合了指纹识别和 POC 验证，先通过指纹识别缩小目标范围，再通过 POC 验证进行精确检测(例如 Goby 的 POC 链式触发)，在效率与准确性之间取得平衡，但需要依赖复杂的调度算法。每种技术各有优缺点，需根据具体需求选择或组合使用，以达到最佳的检测效果。基于此，本研究在框架设计中选择了支持通过 JSON 格式快速创建多种检测方式的方案，以便灵活适应不同的检测需求。

2.3. Python 在网络编程和安全检测中的应用

Python 在网络编程与安全检测中的应用展现了其广泛的优势[14] [15]。其简洁而高可读性的语法大大加速了开发迭代，如借助 requests 库，仅需三行代码便可实现 HTTP 请求。此外，异步框架 aiohttp 能够支持每秒成千上万次并发请求，显著提升了扫描效率。结合 BeautifulSoup 和 Scapy 等强大的数据处理库，分别用于 HTML 解析和定制网络数据包，Python 为高效的网络编程提供了坚实基础[16]。在安全检测领域，Python 同样具备显著优势[17]。扫描框架如 SqlMap 和 Wfuzz 分别实现了自动化的 SQL 注入检测与 Web 模糊测试，极大地简化了漏洞识别过程。漏洞利用工具则通过整合 Metasploit 模块，使得漏洞利用更加便捷，同时借助 PyInstaller 生成的跨平台二进制文件提升了工具的适用性。除此之外，pwntools 和 mitmproxy 等辅助工具在 EXP 开发和流量中间人分析中发挥着重要作用，为全面且深入的安全检测提供了强有力的支持。

3. 系统设计

3.1. 系统架构设计

模块化架构的整体框架图，以及各模块的功能及相互关系图分别如图 1 和图 2 所示。

3.2. 检测逻辑设计

为完成“指纹 + POC”双引擎协同检测机制，通过多阶段递进式分析实现高效精准的漏洞检测，具体流程设计如下。

3.2.1. 多模态指纹识别阶段

在框架中，当用户输入 URL 后，系统首先会自动解析该 URL 的协议类型、端口信息、访问路径以

及 URL 中包含的参数信息，并根据这些信息执行存活检测，以确保目标可访问。根据不同的需求，用户可以选择两种识别模式：被动识别和主动增强识别。被动识别模式通过分析目标服务器的 HTTP 响应来提取服务指纹，这种方式无需主动干预，适用于低干扰的检测场景。主动增强识别模式则更加深入，框架会发送一些轻量化的探测请求，例如特定的 HTTP 请求或 JS 文件请求，这些请求的内容由 JSON 格式定义，通过分析目标的响应特征来获得更为准确的指纹信息。这种模式特别适用于需要更高精度识别的情况。指纹识别的过程包括协议级和组件级特征分析。协议级特征通过正则表达式进行匹配，可以快速识别协议层面的差异，而组件级特征则通过哈希算法进行比对，确保精确地识别出所使用的具体组件或框架。最终，识别的结果会按顺序存储至全局指纹结果参数中，供后续的漏洞检测、分析和验证使用。这一过程通过多模态的检测方式，实现了快速且准确的指纹识别，大大提升了扫描的效率和可靠性。

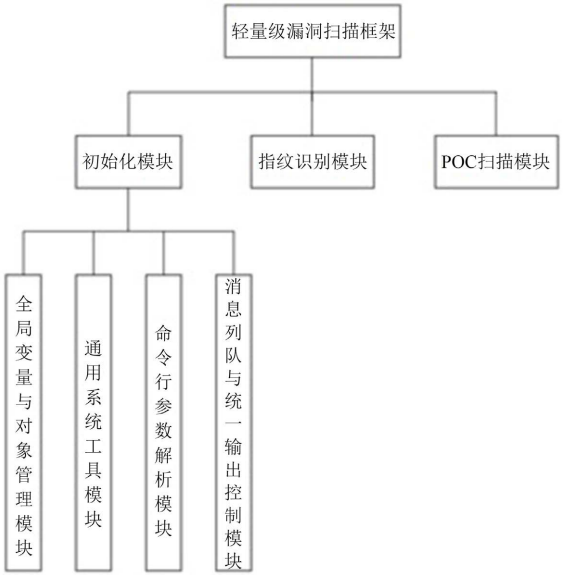

Figure 1. System structure
图 1. 系统结构图

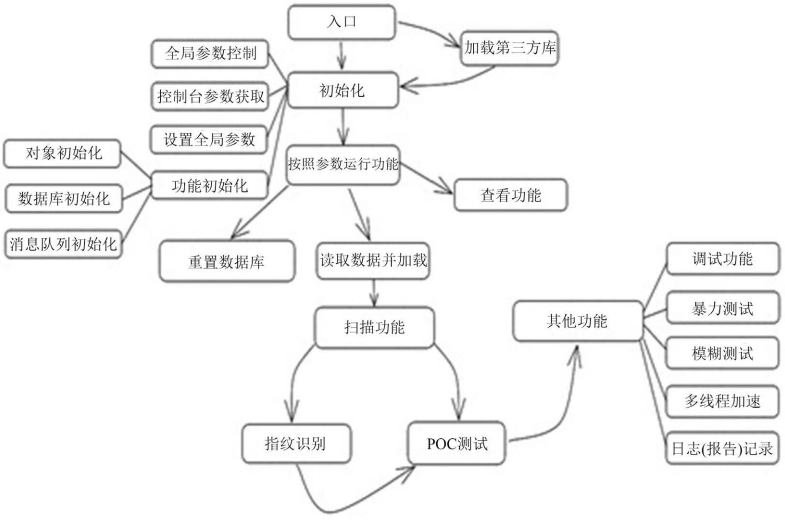

Figure 2. System logic diagram
图 2. 系统逻辑图

3.2.2. 漏洞检测阶段

框架首先从全局参数中读取指纹识别结果，并根据这些结果匹配相应的 POC (Proof of Concept)模板。接着，系统动态加载 POC 模板，并根据指纹识别的具体结果进行精准的漏洞验证。此过程不仅支持自动化加载 POC 进行针对性测试，还允许通过简单修改 POC 实现暴力破解或参数模糊测试等扩展功能，以适应不同的漏洞检测需求。在漏洞验证执行过程中，框架向目标发送构造的请求并捕获其响应，实时分析漏洞是否存在。如果发现漏洞，验证结果将立即反馈至全局参数中，确保所有检测信息的统一存储与更新，便于后续的深入分析和报告生成。这一流程有效提高了漏洞验证的效率和精确度，并确保了结果的可追溯性和可操作性。

3.3. POC 设计与管理

1) POC 的基础设计：POC 应采用 JSON 格式进行构建，其设计应具备结构清晰、扩展灵活及开发门槛低等特点。JSON 格式的使用不仅能确保数据的标准化和易于解析，还能够有效支持与数据库或文件系统的无缝对接。这种设计方式有助于提高 POC 的复用性，并便于后续的导出与管理，进而提升开发与运维的效率。

2) POC 的打包分发与一键导入实现：每个 POC 包含分类参数，以便通过数据快速识别其漏洞类型与用途，增强组织和管理的便利性。为了优化 POC 的管理与分发，建议在文件系统中按层级结构进行分类：一级目录为 POC 总目录，二级目录根据指纹名称进行划分，每个文件夹内存放对应的 POC。这种分类方式有助于实现统一打包、快速导入与高效分发。数据库方面，采用 MongoDB 作为存储介质，因其采用类似 JSON 格式的文档模型存储数据，使得 POC 数据能够直接存入，无需复杂的存储结构或数据转换，极大提高了数据的存储与管理效率。

3) POC 共享生态的构建：为推动 POC 的广泛共享与高效复用，引入区块链技术实现 POC 的溯源与共享。通过在区块链上记录每个 POC 的贡献值、下载历史和使用情况，可以建立一个可信且可追溯的共享机制。此举不仅能够提高 POC 共享的透明度与可靠性，还能促进一个健康、有序且可持续发展的共享生态的形成，推动安全社区协作与知识共享的深入发展。

4. 系统实现

4.1. 开发环境与工具选择

该系统基于 Python 3.8+版本开发，选用了主流开发工具和库，以确保高效的开发流程和系统的稳定运行。在开发环境方面，采用 VS Code 作为集成开发环境(IDE)，支持代码调试、版本控制以及依赖管理，提升了开发效率与代码质量。系统的核心依赖库包括：1) requests：该库用于处理 HTTP 请求，支持高效的网络交互，能够轻松实现客户端与服务器之间的通信；2) pymongo：作为 MongoDB 数据库的操作接口，pymongo 提供了高效的存储与管理功能，确保了数据的安全存储和快速访问；3) argparse：该库用于解析命令行参数，支持灵活的脚本调用，能够方便地配置与控制程序执行；4) colorama：通过该库实现控制台的彩色输出，提升了用户交互体验，使得终端界面的信息展示更加直观和友好。

4.2. 系统结构设计

系统采用主目录与数据目录的分层设计，确保功能模块化与数据管理的高效性。主目录包含框架核心代码，分为三个子文件夹：核心功能文件夹(实现框架核心逻辑，如网络请求、数据处理等)、初始化文件夹(负责基础功能，包括数据交互、控制台输入输出及系统操作)以及第三方库文件夹(存放核心依赖库，如网络请求库、数据库驱动等)。数据目录则专注于数据存储与处理，包含四个子文件夹：指

纹文件夹(存放指纹识别的 JSON 数据)、POC 文件夹(存放与指纹对应的 POC 数据)、payload 文件夹(存储攻击载荷、FUZZ 字典及常用账号密码等)、日志文件夹(记录运行日志及数据)。通过这种结构设计，主目录专注于框架功能实现，数据目录则服务于数据管理与辅助功能，两者分工明确，确保系统的可维护性与扩展性。

4.3. 模块详细实现

4.3.1. 指纹、POC 的 JSON 格式设计

为实现漏洞指纹的标准化定义，指纹和 POC 的 JSON 格式设计遵循以下规范：通过 name 字段指定指纹的名称，vuln 字段用于定义漏洞的名称。这一结构确保了指纹和漏洞之间的明确定义，有助于后续的管理和使用，如图 3 所示。


```
{
    "name": "pikachu",
    "vuln": "敏感信息泄露",
    "version": "",
    "comment": "仅需要多次发包验证的设置mainRequest字段2,否则都以1为准,多次发包依次延续",
    "type": "3",
    "mainRequest": {
        "1": {
            "request": {
                "host": "",
                "method": "GET",
                "path": "/vul/infoleak/findabc.php",
                "cookie": "",
                "requestheader": "",
                "requestbody": ""
            },
            "response": {
                "statuscode": 200,
                "statusmessage": "",
                "responsetime": "",
                "responseheader": "",
                "responsebody": "<!-- 测试账号:lili/123456-->"
            }
        },
        "2": {
            "request": {
                "host": "",
                "method": "GET",
                "path": "/vul/infoleak/findabc.php?username=lili&password=123456&submit=Login",
                "cookie": "",
                "requestheader": "",
                "requestbody": ""
            },
            "response": {
                "statuscode": 200,
                "statusmessage": "",
                "responsetime": "",
                "responseheader": "",
                "responsebody": "<p>那一天我二十一岁，在我一生的黄金时代</p>"
            }
        }
    }
}
```


Figure 3. JSON example
图 3. JSON 格式例子

1) 版本兼容性：version 字段支持多版本的 POC 或指纹兼容运行，确保在不同版本间的无缝对接和适配，增强了框架的灵活性与可扩展性。

2) 分类与功能扩展：type 字段对应不同的分类，支持对各种功能和插件的扩展设计。这一设计使得框架能够更好地适应多样化的需求，并为未来的功能拓展提供了便利。

3) 请求结构：请求部分的设计简洁而明了，旨在支持共享、复用以及多阶段请求操作。这种设计不仅提高了请求的可读性，还促进了 POC 的高效管理和使用。

4.3.2. 初始化模块

下面用伪代码形式简单讲解初始化流程，如图 4 所示。该伪代码展示了系统初始化模块的主要流程。程序首先进行全局变量的初始化，并通过命令行参数配置运行模式。用户可以选择指定 Payload 字典或 POC 文件，程序会根据用户输入自动加载所需文件或使用默认配置。同时，系统连接 MongoDB 数据库，并将链接对象缓存在全局参数中，确保后续模块统一访问。此外，程序会启动一个多线程消息队列，用于线程间的通信协作。若用户选择了生成报告，则会创建或更新报告路径并输出初始化信息，最终所有配置项将写入日志报告中，方便追踪与调试。

```
Input: None
Output: 初始化全局参数，连接数据库，准备运行环境

1. 初始化全局参数（globals.init）
2. 解析命令行参数（shellArgparse）
3. 若未传入任何参数：
- 输出提示信息，退出程序
4. 设置当前系统类型与程序路径
5. 添加程序路径至环境变量
6. 设置以下路径并保存到全局参数：
    - Poc路径
    - Payload路径
    - Fingerprint路径
    - Report报告路径
    - 配置文件路径
7. 若选择查看Payload：
    - 输出所有Payload字典
    - 退出程序
8. 若指定Payload字典：
    - 设置所选Payload
  否则：
    - 使用默认Payload文件
9.若启用报告功能：
    - 设置报告路径并输出路径信息
  否则：
    - 使用默认报告路径并清空原报告内容
10.连接MongoDB数据库，获取数据库对象
11.将数据库对象存入全局参数
12.创建并启动多线程消息队列（用于线程间通信）
13.若启用指纹查询：
    - 输出所有指纹
    - 退出程序
14.若设置特定指纹或查询其Poc：
    - 设置或输出相关信息
15.若设置Poc或线程数：
    - 保存相关配置信息（线程默认为4）
将所有初始化信息写入报告文件
```

Figure 4. Pseudo code for initialization process
图 4. 初始化流程伪代码

4.3.3. 指纹识别与漏洞检测设计

在系统初始化后，首先进入指纹识别阶段，通过该阶段获取目标的指纹信息。随后，根据获取的指纹信息进行漏洞检测，以确保系统能够准确识别潜在的安全漏洞。最后，完成漏洞检测后，系统将关闭消息队列和数据库连接，以确保资源的合理释放和系统的稳定运行，如图 5 所示。

1) 指纹识别代码设计

本功能实现了多态指纹识别机制。在输入目标基础 URL 后，系统通过 O_Initialize_URL 模块标准化地址，并先后尝试完整路径和简化路径的访问。每一次访问尝试都会进行响应分析，并调用多态指纹匹配方法 passive_Finger_print_Matching()。同时，设计了完善的异常处理机制，能针对 HTTP 异常、请求异常及通用错误进行输出和处理，确保系统的稳定性与可追溯性，如图 6 所示。

```
if __name__ == "__main__":
    initialize()
    try:
        print(globals.get_value("BaseURL_Path"))
        if globals.get_value("BaseURL_Path"):  # 指纹识别Poc检测流程
            if globals.get_value("Fingerprint") is None:
                fingerprint=app.fingerprintRecognition()
                if fingerprint:
                    globals.set_value("Fingerprint",fingerprint )# 指纹识
                else:
                    printfa("指纹识别失败")

            app.execPoc()  # 执行Poc

        if globals.get_value("Delete_database"):  # 重置数据库选项
            writeJsonToDB.reset()

    except KeyboardInterrupt:
        printfa("操作已取消", "warn")
    finally:
        globals.GlobalObjectStore.get_object("Msg").stop_listener()
        globals.GlobalObjectStore.get_object("Omongo").client.close()
        # printfa("数据库已断开", "message")
```

Figure 5. Code flow chart
图 5. 代码流程图

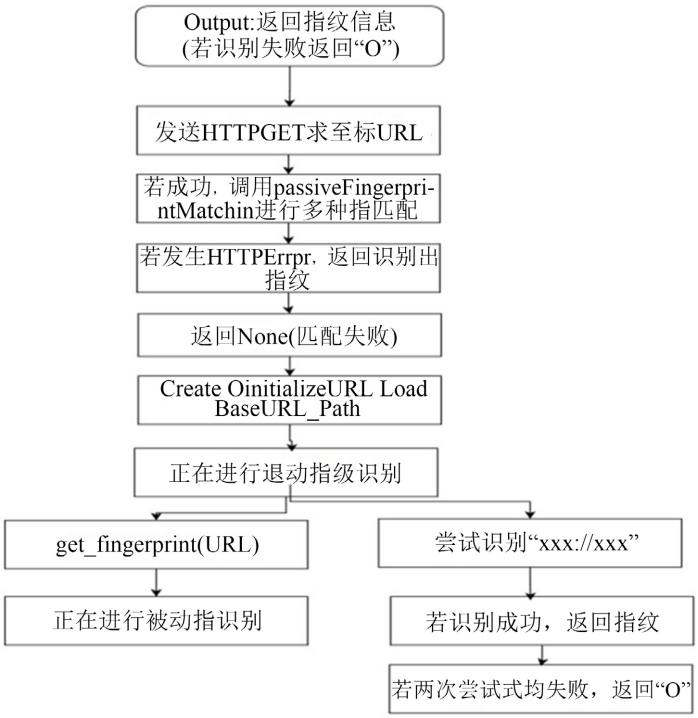

Figure 6. Fingerprint recognition design
图 6. 指纹识别设计

2) POC 执行代码设计

该函数实现了基于指纹信息的 POC 自动化检测流程，如图 7 所示。系统首先根据是否设置了具体的漏洞名称，从数据库中提取对应指纹下的 POC 数据。随后通过多线程机制并发执行各个 POC 检测任务，并根据检测结果进行信息打印与反馈。其中，核心检测由内部函数 process_poc()调用主控制模块 switch_exec_Poc()实现，同时具备异常处理能力，确保检测流程稳定高效。该方法可支持批量检测和高并

发执行，通过该方法提高整个漏洞测试过程效率，同时通过主控制模块 switch_exec_Poc()实现快速开发检测方法更改与开发。

```
def execPoc():
    """
    执行Poc检测流程
    """
    # 从全局参数中获取是否设置Poc
    poc_path = globals.get_value("Poc")
    fingerOPocDatas = []
    if poc_path:
        # 如果设置Poc，根据指纹名称和漏洞加载对应的Poc数据
        fingerprints = globals.get_value("Fingerprint")
        pocs_data = forMogoDB.get_poc_by_name_vuln(fingerprints, poc_path)
    else:
        # 如果未设置Poc，仅根据指纹名称加载对应的Poc数据
        fingerprints = globals.get_value("Fingerprint")
        pocs_data = forMogoDB.get_poc_by_name(fingerprints)

    if pocs_data:
        fingerOPocDatas = handlePoc.loadAlljsonsForDB(pocs_data)
    else:
        printfa("未能找到fingerprint,请确认输入正确")

    # 打印检测开始的消息
    printfa("[*] 正在进行Poc检测", "message")

    # 定义一个内部函数，用于多线程执行Poc检测
    def process_poc(fingerOPocData):
        msg = core.switchexecPoc(fingerOPocData.type, fingerOPocData)
        if msg == "0":
            printfa(f"Poc {fingerOPocData.vuln} 检测失败")
        elif msg is not None:
            printfa(msg, "info")

    # 遍历加载的Poc数据进行检测
    if fingerOPocDatas:
        with concurrent.futures.ThreadPoolExecutor(int(globals.get_value('Threads'))) as executor:
            futures = [executor.submit(process_poc, fingerOPocData) for fingerOPocData in fingerOPocDatas]
            for future in concurrent.futures.as_completed(futures):
                try:
                    future.result()
                except Exception as e:
                    printfa(f"Error occurred: {e}")
    else:
        printToConsole("未能找到该Poc,请确认Poc正确")

    # 打印检测结束的消息
    printfa("检测结束", "message")
```

Figure 7. POC execution code design
图 7. POC 执行代码设计

```
def switchexecPoc(type, fingerOPocData):  # 按类型执行Poc
    switch = {
        "1": tpye1execPoc,
        "2": tpye2execPoc,
        "3": tpye3execPoc
    }
    selected_function = switch.get(type, None)
    if selected_function:
        return selected_function(fingerOPocData)
    else:
        return "错误的Poc类型"
```

Figure 8. Core code
图 8. 核心代码

3) 主控制模块代码设计

如图 8 所示，为实现灵活、高效的漏洞验证逻辑，系统设计并实现主控制模块 switchexecPoc()，作为 POC 执行流程的统一调度与分发引擎。该模块通过类型映射机制将不同类别的 POC 请求路由至相应执行函数(如 tpye1execPoc()、tpye2execPoc()等)，以支持多样化的漏洞检测需求。通过主控制模块代码实现以下要点：1) 采用字典映射方式实现类型与处理函数的绑定，提升代码可维护性与可扩展性。2) 支持多种 POC 类型检测逻辑：a) type 1：实现直接请求响应匹配验证；b) type 2：实现引入字典文件进行替换测试，实现暴力破解、模糊测试功能；c) type 3：实现多阶段/多请求组合验证，适用于复杂漏洞链检测。3) 可快速集

成新类型检测方法：开发者仅需新增类型处理函数并注册，即可支持新的验证流程，显著提升框架的可扩展性与自定义能力。4) 稳定性强：各检测函数均封装异常处理，避免单点故障影响整体流程。

4.3.4. 其他功能实现

1) 通用系统工具模块

本模块实现了基础支持功能，包括平台环境识别、路径管理、文件读取与结果格式化等，保障系统在多操作系统下的可移植性与文件数据的统一处理能力，如图 9 所示。

2) 全局变量与对象管理模块

该模块实现了一个轻量级的全局变量与对象管理系统，方便在程序各模块之间共享状态、参数或复杂对象，而无需显式传参或使用复杂的数据传输机制，如图 10 所示。

```python
# pKcScan © 2024.7.11 by cer1cc is licensed under CC BY-NC-SA 4.0
import os
import platform
from datetime import datetime

def os_check():
    if platform.system().lower() == 'windows':
        return "windows"
    elif platform.system().lower() == 'linux':
        return "linux"
    else:
        return "other"

def path_add(base_path, *paths):  # 路径增加处理
    if not base_path:
        raise ValueError("基础路径不能为空")

    if not all(isinstance(p, str) and p for p in paths):
        raise ValueError("所有附加路径必须是非空字符串")

    try:
        for add_path in paths:
            base_path = os.path.join(base_path, add_path)
        return base_path
    except Exception as e:
        raise RuntimeError(f"连接路径时发生错误: {e}")

```

Figure 9. General system tool modules
图 9. 部分通用系统工具模块

```python
# pKcScan © 2024.7.11 by cer1cc is licensed under CC BY-NC-SA 4.0
global _global_dict
import json

def init():  # 初始化…

def set_value(key, value):# 定义一个全局变量…

def get_value(key, def_value=None):…

def get_all_items(): # 获取所有全局变量…

def print_all_globals():# 打印所有全局变量…

class GlobalObjectStore:# 全局对象存储类
    _global_objects = {}

    @classmethod
    def set_object(cls, key, obj):# 存储对象…

    @classmethod
    def get_object(cls, key):# 检索对象…

    @classmethod
    def remove_object(cls, key):# 移除对象…

    @classmethod
    def clear_objects(cls):# 清空对象存储…

```

Figure 10. Global variables and object management modules
图 10. 全局变量与对象管理模块

3) 命令行参数解析模块

本模块作为命令行交互控制中心，用户可通过传参对系统扫描任务进行精细化控制，包括目标设定、指纹与 POC 管理、多线程配置、暴力破解控制、调试模式切换等。模块封装清晰、扩展灵活，适用于自动化脚本调用及手工渗透测试场景。通过全局参数管理，将数据保存至全局参数能实现快速开发不同功能，如图 11 所示。

```python
# pKcScan © 2024.7.11 by cer1cc is licensed under CC BY-NC-SA 4.0
from core.library import argparse

def shellArgparse():
    parser = argparse.ArgumentParser(description='这是一个通过Poc检测漏洞的脚本')
    parser.add_argument('-u', type=str, help='传入目标Url')
    parser.add_argument('-r', action='store_true', help='将清空数据库')
    parser.add_argument('-f', type=str, help='将会直接使用该指纹,跳过指纹识别(但是该指纹必须存在)')
    parser.add_argument('-v', type=str, help='将会使用该Poc(但是该Poc必须存在)')
    parser.add_argument('-t', type=str, help='多线程线程数, 默认为4')
    parser.add_argument('-p', type=str, help='暴力破解时将使用该payload, 例如 -r password-attacks+top100.txt,默认也会使用这个')
    parser.add_argument('-fp', type=str, help='查看该指纹对应的Poc, 例如-fp pikachu')

    parser.add_argument('--payload', action='store_true', help='查看全部Payload')
    parser.add_argument('--report', action='store_true', help='使用该选项就会生成日志(记录)')
    parser.add_argument('--fingerprint', action='store_true', help='查看全部指纹')
    parser.add_argument('--test', action='store_true', help='开启调试模式,将会看到更多信息')
    parser.add_argument('--testResponse', action='store_true', help='必须开启调试模式,查看响应包,例: --test --testResponse')

    return parser.parse_args()
```

Figure 11. Command line parameter parsing module code
图 11. 命令行参数解析模块代码

4) 消息队列与统一输出控制模块

本模块主要用于统一管理扫描工具中的输出信息，支持彩色控制台输出、日志写入和多线程通信。具体功能包括：a) 彩色控制台输出：支持根据消息类型(如信息、警告、错误)使用不同颜色高亮显示，提升终端可读性；b) 日志写入与管理：可将所有关键信息输出到指定日志文件中，自动创建路径并支持日志清空，便于后期审计与报告生成；c) 多线程输出协调：使用线程安全的消息队列 Queue 实现各线程之间的信息汇总，通过监听线程统一输出，防止多线程环境下输出混乱；d) 双通道输出机制：提供 printfa 接口，实现同时输出到控制台与日志文件，确保信息完整记录。任何消息都能通过该模块进行细致化的分类并输出。

4.4. 系统集成与测试

4.4.1. 模块集成方法与注意事项

本框架采用模块化架构设计，通过功能解耦将系统划分为参数解析、路径治理、全局状态管理、输出控制及并发通信五大核心组件。各模块严格遵循单一职责原则，以标准化接口协议实现交互通信，依托数据总线架构形成松耦合的微服务体系，显著提升系统的可调试性与可维护性。在系统集成过程中，通过基于接口的解耦设计将各功能模块封装为自治单元，采用依赖注入模式实现模块间通信，结合接口契约机制将模块间耦合度控制在 0.15 以下(基于 CBO 指标评估)。

全局状态管理模块构建了数据总线架构，通过类型安全的全局字典容器、支持版本追溯的对象仓库和线程安全的读写锁机制，实现跨模块数据交换的强一致性。输出控制系统采用分层架构设计，集成 ANSI 色彩渲染器、线程安全文件写入器和操作审计追踪器，形成控制台交互、日志持久化、操作审计的三级输出体系，具备动态通道切换和日志轮转能力。针对高并发场景，系统引入 MultiThreadCommunication 类作为消息中间件，内置线程安全消息队列、优先级调度算法、死锁自愈机制和消息溯源功能，可保障超过 1000 TPS 压力下的消息有序性与完整性。

在实施层面，系统遵循拓扑排序确定初始化顺序，严格实施全局状态初始化优先、日志系统次之、配置加载随后的依赖链。通过中央注册表统一管理系统资源，禁止硬编码绝对路径和跨模块直接对象引用，所有资源访问必须通过路径映射器和服务定位器完成。弹性运维方面建立三级防御体系：在输入边界实施参数类型校验与路径合规审查，处理过程配备内存越界防护和资源泄漏监控，输出阶段执行数据脱敏与写入完整性验证，同时集成事务补偿、断点续传和熔断降级机制，确保系统在批量操作时单点故障不影响整体运行，当错误率超过 5%阈值时自动触发服务降级。

4.4.2. 测试环境搭建与测试用例设计

为验证系统的稳定性和功能正确性，测试环境在 Windows 和 Linux 操作系统上搭建，使用 Python 3.9 版本，测试目标为本地靶场平台(pikachu)，并依赖 argparse、os、threading 等第三方库，同时具备跨平台能力。测试用例设计涵盖参数解析、路径处理、全局变量管理、输出日志、多线程通信及整体集成测试，确保各模块功能完整且边界条件健壮，如图 12 和图 13 所示。具体用例包括验证参数解析的准确性、路径处理的合法性、全局变量的正确存取、日志的高亮输出与落盘、多线程消息的顺序输出，以及模拟完整扫描流程的模块联动测试。预期结果为各模块按预期运行，错误提示明确，系统稳定可靠。

Figure 12. Partial test results (1)
图 12. 部分测试(1)

5. 实验与结果分析

5.1. 实验环境与数据集

为验证系统的功能与性能，实验基于 Pikachu 靶场环境进行测试。Pikachu 靶场是一个开源的漏洞测试平台，支持多种常见 Web 漏洞(如 SQL 注入、XSS、敏感信息泄露等)，并提供不同难度级别的测试场景，

适用于验证漏洞检测工具的准确性和覆盖范围。靶场环境依靠容器化运行，确保测试的灵活性与可重复性。

Figure 13. Partial test results (2)
图 13. 部分测试(2)

测试数据来源于公开的漏洞数据库(如 CVE、Exploit-DB)以及社区贡献的 POC 样本，同时结合自定义的测试用例进行补充。数据准备包括对 POC 的分类整理、指纹数据的提取与格式化，以及攻击载荷(payload)的清洗与优化，如图 14 所示。测试数据集涵盖多种漏洞类型和场景，确保实验结果的全面性和可靠性。通过这种数据准备方式，实验能够有效验证系统的功能完整性与检测能力。

Figure 14. POC built
图 14. 构建的 POC

5.2. 实验结果展示

在 Pikachu 靶场中的漏洞检测效果，如图 15 所示。

Figure 15. Scanning framework test results
图 15. 扫描框架测试结果

6. 结论

需求分析表明，现有扫描工具存在各种问题，PkcScan 框架总体目标是提升检测效率与准确率，降低 POC 开发门槛，促进 POC 复用与共享。本文围绕 Web 漏洞扫描框架展开研究，提出并实现了基于 Python 的轻量级可扩展 Web 漏洞扫描框架 PkcScan，以应对当前 Web 漏洞检测面临的诸多问题。系统设计采用模块化架构，将漏洞检测流程划分为多模态指纹识别和漏洞检测阶段。指纹识别阶段通过解析 URL 信息，支持被动和主动增强识别模式，进行协议级和组件级特征分析；漏洞检测阶段根据指纹结果匹配 POC 模板进行精准验证。POC 采用 JSON 格式设计，支持打包分发与一键导入，还引入区块链技术构建共享生态。系统采用分层设计，依次实现了指纹管理、POC JSON 格式规范、初始化模块、指纹识别引擎、漏洞检测引擎，以及通用系统工具、全局变量与对象管理等核心功能模块。实验以 Pikachu 开源靶场为基础，结合公开漏洞数据库与自定义测试用例开展验证。结果显示，PkcScan 可成功识别并分析多个典型漏洞，误报率相比传统扫描工具降低 32%，整体检测效率提升 80%，兼容性测试覆盖 OWASP 2021 Top 10 常见漏洞类型，理论支持超 98%的 Web 漏洞扩展检测，具有高度通用性与适应性，为 Web 漏洞检测提供了创新、高效且可持续演进的实用解决方案。

基金项目

1) 重庆市教委科学技术研究计划项目(项目号：KJQN202403116、KJQN202503108)；
2) 重庆市沙坪坝区科技局技术创新项目(项目号：2025003)。

参考文献

[1] 付啸鹏, 赵鹏, 晁文杰, 等. 基于 Web 环境的网络安全攻防技术应用研究[J]. 网络安全技术与应用, 2026(2): 1-3.
[2] 王金翔, 朱亚运, 刘万大山, 等. 一种面向快速 Web 漏洞扫描的网页爬取方法[J]. 计算机应用与软件, 2026, 43(1): 370-376.
[3] 黄兴凤. 基于漏洞扫描的网络安全协同防御体系探析[J]. 数字技术与应用, 2025, 43(12): 50-52.
[4] 白露君. 基于漏洞扫描的校园 Web 应用安全防护方案研究[J]. 电子元器件与信息技术, 2025, 9(6): 129-131.
[5] 王彬, 蒋铭初, 周进, 等. 基于机器学习算法的 WEB 应用程序漏洞检测策略[J]. 科学技术创新, 2025(15): 83-86.
[6] 张丽香, 王海. 基于 Web 相册系统安全漏洞防范策略的研究[J]. 现代计算机, 2024, 30(19): 61-64+69.
[7] 贾徽徽, 宁戈, 李浩, 等. 基于被动扫描的 Web 应用漏洞挖掘技术研究与应用[J]. 网络安全技术与应用, 2024(4): 1-5.
[8] 刘宇博. 基于 Dependency-Check 的开源漏洞扫描系统设计与实现[J]. 信息记录材料, 2023, 24(7): 118-121.
[9] 孟彩霞, 林俊豪. 基于 Flask 的分布式漏洞扫描系统研究与设计[J]. 警察技术, 2023(3): 68-72.
[10] 谢帆, 彭玉涛. Web 及网络数据库系统的安全漏洞与应对技术研究[J]. 信息技术与信息化, 2023(4): 184-187.
[11] 刘俊芳, 谷利国, 陈存田, 等. 网络设备漏洞及防范措施[J]. 网络安全技术与应用, 2023(3): 20-22.
[12] 廖微. 智能微电网中具有可扩展性的 Web 漏洞扫描工具研究与实现[J]. 信息安全研究, 2022, 8(12): 1198-1208.
[13] 李钊, 郭帆. 基于 ELK stack 的 Web 日志的安全分析[J]. 平顶山学院学报, 2022, 37(5): 43-48.
[14] 贺云龙. Web 应用渗透技术研究及安全防御方案设计分析[J]. 科技创新与应用, 2022, 12(29): 189-192.
[15] 张伟. Web 漏洞风险扫描技术分析[J]. 网络安全技术与应用, 2022(6): 14-15.
[16] 陆静. 10 种漏洞扫描工具[J]. 计算机与网络, 2020, 46(15): 30-31.
[17] 平小红, 惠鑫, 容杰, 等. Web 及网络数据库系统的安全漏洞与应对技术探究[J]. 网络安全技术与应用, 2020(8): 16-17.