Computer Science and Application 计算机科学与应用, 2026, 16(3), 771-781
Published Online March 2026 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2026.163102

Hans汉斯

基于污点分析的PHP应用威胁检测平台

王　伟，于　越，张运玮，陈　聪，姬懿轩，康晓凤

徐州工程学院信息工程学院，江苏　徐州

收稿日期：2026年2月15日；录用日期：2026年3月13日；发布日期：2026年3月24日

摘　要

本文针对PHP应用中由于序列化滥用、输入校验缺失、路径解析歧义而引起的高危安全威胁，将快速预筛、基于AST/CFG的深度语义分析、跨函数关联溯源三种模式有机结合起来，构造从受污染源到触发点的可视化证据链，并设计实现了一套多模态污点分析检测框架，解决了传统审计工具在跨文件数据流追踪、消毒器识别、路径归一化等方面的不足，同时引入AI辅助技术对漏洞片段做智能摘要及自动化验证，提高了人工复核效率。通过在DVWA靶场及若干真实开源项目上的对比实验，验证了所提方法在复杂漏洞挖掘中的可靠性及所生成报告的可信性。

关键词

污点分析，跨文件数据流，路径归一化，PHP安全，静态分析

A PHP Threat Detection Platform Based on Tag Analysis

Wei Wang, Yue Yu, Yunwei Zhang, Cong Chen, Yixuan Ji, Xiaofeng Kang

School of Information Engineering, Xuzhou University of Technology, Xuzhou Jiangsu

Received: February 15, 2026; accepted: March 13, 2026; published: March 24, 2026

Abstract

This paper addresses high-severity security threats in PHP applications caused by the misuse of serialization, missing input validation, and ambiguities in path resolution. It integrates three complementary modes—rapid pre-screening, deep semantic analysis based on AST/CFG, and cross-function correlation tracing—to construct a visual evidence chain from taint sources to vulnerability trigger points. We design and implement a multimodal taint-analysis detection framework that overcomes limitations of traditional auditing tools in cross-file data-flow tracking, sanitizer identification, and path normalization. In addition, we introduce AI-assisted techniques to generate

文章引用: 王伟, 于越, 张运玮, 陈聪, 姬懿轩, 康晓凤. 基于污点分析的 PHP 应用威胁检测平台[J]. 计算机科学与应用, 2026, 16(3): 771-781. DOI: 10.12677/csa.2026.163102

intelligent summaries of vulnerable code snippets and automate verification, thereby improving the efficiency of manual review. Comparative experiments on the DVWA testbed and several real-world open-source projects demonstrate the reliability of the proposed approach for discovering complex vulnerabilities and the credibility of the generated reports.

Keywords

Taint Analysis, Cross-File Data Streams, Path Normalization, PHP Security, Static Analysis

1. 引言

PHP 语言在网站服务端使用广泛，其开发高效、部署便捷，在内容管理系统、电商平台以及企业级业务系统中占据了举足轻重的位置[1]。由于固有的脆弱性：动态类型、弱约束处理、字符拼接、历史遗留 API 等特性导致注入型漏洞频发。OWASP Top 10 报告长期把注入类风险列为高优先级威胁之一[2]，此类漏洞在实际业务中常与鉴权缺失、配置错误、依赖风险彼此组合，形成危险的复合攻击链。现有主要的检测手段存在明显短板：黑盒扫描难以覆盖复杂逻辑和深层路径，漏报率高；人工审计虽准确，但在大规模、高迭代的开发模式下，效率低、成本高，难以持续。针对上述问题，从理论层面对静态污点分析与符号执行方法进行梳理，分析主流工具在数据流建模、路径可达性验证及动态语义处理方面的技术特点与局限性，为后续平台的实现进行理论的分析。

2. 相关研究

2.1. 静态污点分析与变量回溯

污点分析的核心思想是把外部输入标记为污点源，追踪其传播路径直到达危险操作(汇点)。曹凯等提出基于 AST 生成 CFG 并对敏感参数做变量回溯的污点分析算法。生成 CFG 时发现了敏感函数的调用，则获取其中包含的危险参数，通过变量回溯的方法进行向上追踪，判断该参数是否来源于污点源中，并追踪其是否经过了净化处理来判断是否会产生漏洞。基于 PHP 语法分析器 PHP-Parser 对 PHP 进行词法和语法分析，产生抽象语法树，再由其产生对应的控制流图，然后在控制流图上实行污点分析[3]。该方法依赖成熟的语法解析库实现对 PHP 语法特性的支持。但只支持面向过程的程序，对于面向对象的分析方法还需要进一步的研究分析。当前支持检测的漏洞类型也较少。

2.2. 中间表示与前向数据流迭代

为提高语义规约及分析的可实现性，部分研究采用三地址码或 SSA 等中间表示。王国峰等以三地址码作为 IR，自然、设计污点数据流值及传播规则，在 CFG 上迭代执行数据流算法，且每步迭代都做安全检查以定位含污点数据的 sink 点集合[4]。值得指出的是，此类方法对 SQL 注入等字符串拼接场景很敏感，而 IR 构建及语言特性支持仍是工程化中十分重要的问题。

2.3. 污点分析与符号执行/动态验证融合

纯静态污点分析在分支可达判断和对象状态分析上容易产生误报。刘行波等提出通过生成更精确的

对象状态记录，并精准追踪污点与 sink 的传递路径，有效减少过污染和欠污染。同时引入符号执行验证漏洞可达性，排除虚假告警，进一步降低误报漏报[5]。但该方法仍面临符号执行路径爆炸、外部依赖建模、与静态分析结果对齐等挑战。

2.4. 工具与平台实践

Pixy 是早期 PHP 污点分析的代表性工具，采用静态数据流分析检测 XSS 等漏洞，通过构建基础控制流图进行数据流分析来实现[6]。RIPS 利用了 PHP 内置的词法分析函数 token_get_all 来提取源代码中的全部符号序列以控制流建模，用块/函数摘实现高精度污点分析，在工业界得到广泛应用[7]。近年来，针对 PHP 的基准评测工作从规则覆盖及误报/漏报平衡角度提出了系统化的评估方法[8]。上述工具与方法可作为本文的对比基线。

3. 平台的设计与实现

3.1. 总体架构

平台采用分层架构如图 1 所示，主要分为四个层次：接口层设计为 CLI 和 Flask Web，引擎层用 AST 精扫及跨文件污点追踪，知识库层负责源/汇/消毒器及规则的管理，呈现层给出表格、JSON/HTML 报告及可视化各种结果。该架构可分析算法与工程能力(缓存、分页、权限、报告)，使其彼此解耦，利于系统迭代与横向扩展。统一数据模型设计，将快速扫描、深度扫描、函数摘要这三种扫描模式的输出规范为一致的结构，包括漏洞发现(findings)、污点传播路径(taint_paths)及可视化数据(visualization)。这种设计有效降低了前端与报告层的开发复杂度。

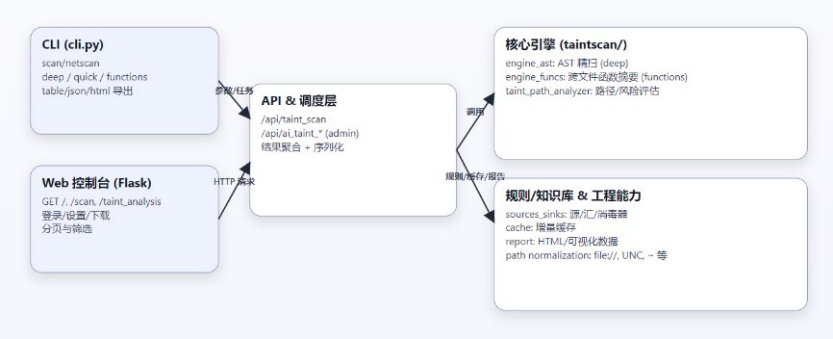

Figure 1. Architecture of the TaintScan platform
图 1. TaintScan 平台总体架构图

从模块划分出发实现工程边界：CLI 入口实现参数解析、路径归一化以及结果导出，适用于 CI 流水线的自动化扫描，Web 入口提供扫描页面及 API 接口，直接支持交互式审计及团队协作。引擎层用调度模块来协调聚合过程，依次调用 AST 深度扫描引擎、跨文件函数摘要引擎，再把二者结果统一纳入同一数据模型中。污点路径分析模块把传播关系整理为路径对象，据此计算风险等级、消毒状态及各节点/边的统计信息。报告生成模块负责 HTML 的序列化及转义处理，防止渲染时造成安全风险。此外缓存模块及分页机制为大规模代码仓库提供性能的保障。

3.2. 三模式扫描与污点路径

平台提供快速扫描、深度扫描及函数摘要三种模式如图 2 所示适用于不同的场景。

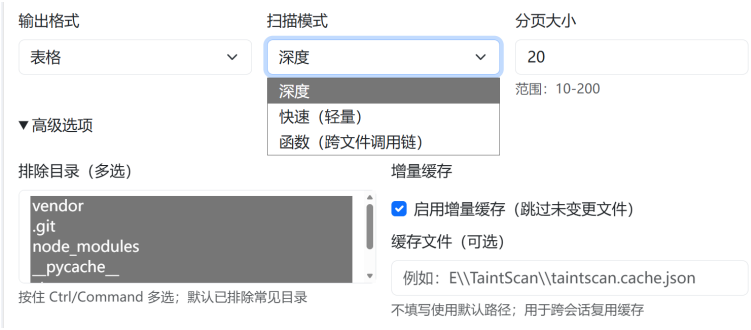

Figure 2. Three-mode scanning working display diagram
图 2. 三模式扫描工作显示图

1) 快速扫描

基于 token 匹配、正则表达式及简单语义约束，快速识别输入直接进入危险函数的显式漏洞模式。该模式适用于 CI 流水线或大规模代码仓库的初步预筛，主要输出高风险点及建议深度扫描的文件列表。对于置信度不足的检测项，直接加入深度扫描建议列表，避免误报。

2) 深度扫描

采用 AST/CFG 级语义分析，通过变量回溯方法从敏感函数参数向前追溯数据来源，验证是否经过安全处理[3]。该模式针对 SQL 拼接等场景提供结构化证据，包括拼接表达式、参与变量及完整回溯链。通过剪枝与缓存复用控制分析复杂度。

3) 函数摘要模式

以函数摘要为基本单位进行跨文件污点传播[4]。摘要记录形参污点对返回值、全局状态及潜在 sink 的影响关系，沿调用图迭代传播直至收敛，最终生成源到汇的路径证据链及风险等级。

在以上三种模式的基础上，需要考虑到动态特性处理策略以及风险等级评估。

针对 PHP 的动态包含、可变函数、反射调用等特性，采用“可解析则精确、不可解析则保守”的策略：优先通过常量传播及白名单摘要缩小候选集合；无法精确解析时进行污点传播而非截断路径，使用不确定边连接并标记需人工复核的节点，避免漏报。

风险等级评估综合考虑 sink 类型、消毒强度以及路径不确定性：对于命令执行与文件包含类 sink 标记为最高风险；弱消毒函数(如 addslashes)仅降低置信度而不判定为安全；动态包含或反射引入的不确定边在报告中注明相应的依据以备复核。

3.3. 污点模型与风险评估机制

3.3.1. 污点状态与传播规则

平台为变量维护污点状态 $T(x)\in\{\text{untainted}, \text{tainted}, \text{maybe}\}$，采用源/汇/消毒器三元组检测如图 3 所示。污点源包括$_GET、$_POST 等外部输入，汇点包括数据库执行、命令执行等危险操作，消毒器根据场景分类(如 htmlspecialchars 用于 XSS、PDO::prepare 用于 SQL 注入)。对 addslashes 等弱消毒仅降低置信度。规则库支持扩展配置[8]。

平台遵循保守原则(如公式 1)：只要操作数中存在潜在污染，结果即为污染，除非经过消毒器(Sanitizer)处理。为降低误报，采用上下文约束机制——消毒器仅在类型和位置匹配的适用场景生效，例如输出编码仅作用于输出上下文，SQL 场景需验证参数化绑定，文件路径场景则检查 realpath、白名单目录及后缀限制。其次是可达性约束，对于路径存疑的检测项，可借助符号执行或动态复核做二次验证以过滤不可达路径，这一思路与现有研究中通过符号执行降低误报的方法相符。

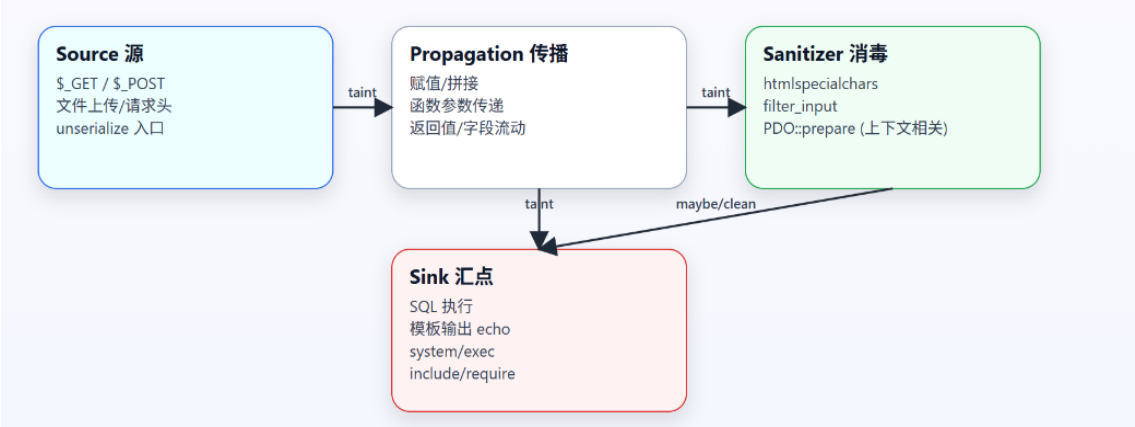

Figure 3. Taint model and evidence chain (source-propagation-sanitizer-sink)
图 3. 污点模型与证据链示意图

$$T(x)=\max\left(T(y),T(z)\right)\times\left(1-\text{IsSanitized}(x)\right) \tag{1}$$

3.3.2. 风险量化评估算法

为了解决传统工具仅输出布尔结果(有/无漏洞)的局限性，平台引入风险评分函数 Risk(p)。对于任意一条从源 s 到汇 k 的污点路径 p，其风险得分计算公式如下：

$$\text{Risk}(p)=I_{sink}\times C_{reach}\times\left(1-E_{san}\right) \tag{2}$$

危害基准因子(I)基于 CWE 危害等级定义，RCE/文件包含类型 Sink 设定为 10.0，SQL 注入为 8.0，XSS/SSRF 为 6.0，信息泄露为 4.0。

路径置信度(C)：量化静态分析的不确定性。文件内确定性路径取 1.0；涉及循环/条件分支的路径取 0.8；涉及接口多态或反射调用的跨文件路径取 0.6。

消毒有效性(E)：完全无消毒取 0；弱消毒(如宽字节下的 addslashes)取 0.3；强消毒(如 PDO::prepare)取 0.9。

基于 Risk(p)值，平台将漏洞划分为严重(8.0~10.0)、高危(5.0~7.9)、中危(3.0~4.9)及低危提示(0~2.9)四个等级，从而实现报告的优先级排序。

3.4. 工程能力与安全

3.4.1. 缓存与分页

平台提供增量缓存以复用 AST 及函数摘要，缓存键由规范路径、文件哈希和规则版本构成。Web 端提供分页及分类筛选功能。由于交互式审计中缓存及分页的价值十分明确：审计人员在使用交互式界面时要反复调整排除目录、模式、关键字过滤诸种参数，若每次都从头解析 AST，响应时间显然无法接受，故平台很自然地用缓存把“重复解析”转化为“复用摘要”，同时在 Web 侧合理限制`page_size`的取值范围，防止一次渲染太多条目导致卡顿。

3.4.2. 路径归一化

路径归一化直接影响扫描覆盖的完整性和结果准确性。为实现不同系统的正常运行，平台对绝对/相对路径、file://协议、UNC 共享路径、~家目录及环境变量进行统一处理，并保留原始输入与规范路径的映射关系以供报告追溯如图 4 所示，避免漏扫和检索不到对应的路径。

Figure 4. Cross-platform path normalization pipeline
图 4. 跨平台路径归一化流程图

归一化采用确定性流水线实现，便于复现和测试。将所述的归一化抽象为流水线函数 Norm(P)，函数 Expand 为展开环境变量(%USERPROFILE%或者~家目录)，函数 Resolve 是处理 file://协议、UNC 路径及../相对符号，函数 Abs 结合 ROOT_DIR 转换为绝对路径并校验文件存在性(如公式 3)，最后将映射关系用于生成缓存、去重及确定报告位置。若路径解析或校验失败，平台返回初始路径并提示异常错误，便于问题排查。

$$\mathrm{Norm}(P) = \mathrm{Abs}\left(\mathrm{Resolve}\left(\mathrm{Expand}\left(P\right)\right)\right) \tag{3}$$

3.4.3. 权限控制与审计留痕

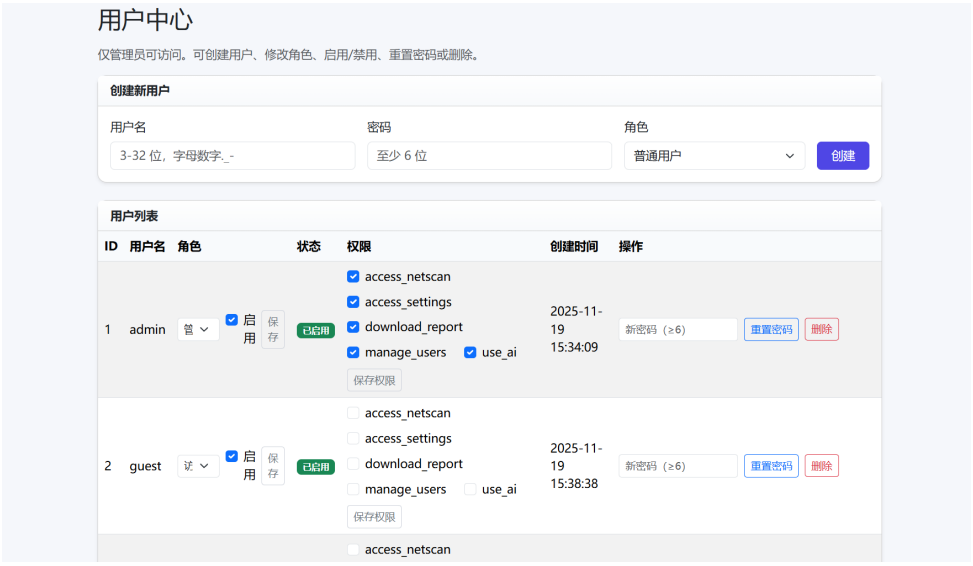

Figure 5. RBAC roles and control points
图 5. RBAC 角色与控制点示意图

平台实现了基于角色的访问控制，定义了 admin、user 和 guest 三种角色如图 5 所示。AI 辅助能力及系统设置页面仅对管理员开放，报告下载和网络扫描功能对非管理员角色进行限制。API 及页面层均实施权限校验并返回明确的错误信息，满足审计追溯需求，具体权限的分配如表 1。数据采用 mysql 进行存储，利用 hash 加密进行后端存储，保障用户的安全性，提高安全管理的可视化能力。登录采用 session 维持，实现用户的可持续登录。

Table 1. User permission table
表 1. 用户权限表

权限标识(Permission Key)	功能描述	管理员(Admin)	普通用户(User)	访客(Guest)
manage_users	用户中心管理	√	-	-
access_settings	AI 参数配置	√	-	-
use_ai	AI 功能调用	√	-	-
access_netscan	网络扫描任务	√	√	-
download_report	扫描报告下载	√	√	-

3.4.4. 报告与可视化

CLI 提供表格、JSON 与 HTML 三种输出格式；Web 端提供结果表格、筛选搜索与污点路径图可视化。报告序列化与前端渲染过程统一进行转义与清洗(如图 6)，以防止扫描报告本身成为 XSS 攻击入口。

类型	文件	行号	汇点	变量	说明
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	27	echo	$user_input	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
命令执行	E:\TaintScan\examples\advanced_sanitization_demo.php	54	system	$user_cmd	风险：命令执行；函数：system；原因：参数包含不可信输入且未充分消毒；建议：避免拼接系统命令；如必须，使用 escapeshellarg/escapeshellcmd 并限制白名单。
文件包含	E:\TaintScan\examples\advanced_sanitization_demo.php	77	include	$user_file	风险：文件包含；函数：include；原因：参数包含不可信输入且未充分消毒；建议：对路径做白名单校验，结合 basename/realpath，移除用户可控路径片段。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	105	echo	$user_content	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	108	echo	$user_content	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	111	echo	$user_content	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	114	echo	$user_content	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	203	echo	$safe_html	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\advanced_sanitization_demo.php	204	echo	$safe_js	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
命令执行	E:\TaintScan\examples\command_exec_demo.php	9	system	$cmd	风险：命令执行；函数：system；原因：参数包含不可信输入且未充分消毒；建议：避免拼接系统命令；如必须，使用 escapeshellarg/escapeshellcmd 并限制白名单。
文件包含	E:\TaintScan\examples\complex_array_object_demo.php	80	include	$merged_array	风险：文件包含；函数：include；原因：参数包含不可信输入且未充分消毒；建议：对路径做白名单校验，结合 basename/realpath，移除用户可控路径片段。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	32	echo	$deep_result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	108	echo	$closure_result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	122	echo	$processed	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	187	echo	$dynamic_result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	194	echo	$reflection_result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	222	echo	$processed_data	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	244	echo	$result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	263	echo	$recursive_result	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\multi_layer_nested_demo.php	310	echo	$decoded	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
SQL 注入	E:\TaintScan\examples\multi_layer_nested_demo.php	314	mysqli_query	$string_sql	风险：SQL 注入；函数：mysqli_query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
SQL 注入	E:\TaintScan\examples\pdo_demo.php	5	->query	$name	风险：SQL 注入；方法：->query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
命令执行	E:\TaintScan\examples\pdo_demo.php	13	exec	$id	风险：命令执行；函数：exec；原因：参数包含不可信输入且未充分消毒；建议：避免拼接系统命令；如必须，使用 escapeshellarg/escapeshellcmd 并限制白名单。
代码执行	E:\TaintScan\examples\preg_replace_e_demo.php	6	preg_replace	$replacement	风险：代码执行；函数：preg_replace；原因：参数包含不可信输入且未充分消毒；建议：禁止使用 eval/assert 或 /e 修饰符，采用安全替代实现（如 preg_replace_callback）。
代码执行	E:\TaintScan\examples\preg_replace_e_demo.php	9	preg_replace	<直接源>	风险：代码执行；原因：preg_replace 使用 /e 修饰符且替换参数含不可信输入；建议：移除 /e 修饰符，改用 preg_replace_callback
SQL 注入	E:\TaintScan\examples\sql_injection_demo.php	6	mysqli_query	$sql	风险：SQL 注入；函数：mysqli_query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
SQL 注入	E:\TaintScan\examples\sql_injection_demo.php	10	mysqli_query	$sql2	风险：SQL 注入；函数：mysqli_query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
SQL 注入	E:\TaintScan\examples\sql_injection_demo.php	15	->query	$term	风险：SQL 注入；方法：->query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
XSS	E:\TaintScan\examples\visualization_demo.php	12	echo	$user_input	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\visualization_demo.php	22	echo	$comment	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
SQL 注入	E:\TaintScan\examples\visualization_demo.php	47	mysql_query	$query	风险：SQL 注入；函数：mysql_query；原因：参数包含不可信输入且未充分消毒；建议：使用预处理语句（prepare/bindParam）或参数化查询，避免字符串拼接。
命令执行	E:\TaintScan\examples\visualization_demo.php	71	exec	$command	风险：命令执行；函数：exec；原因：参数包含不可信输入且未充分消毒；建议：避免拼接系统命令；如必须，使用 escapeshellarg/escapeshellcmd 并限制白名单。
代码执行	E:\TaintScan\examples\visualization_demo.php	101	eval	$content	风险：代码执行；函数：eval；原因：参数包含不可信输入且未充分消毒；建议：禁止使用 eval/assert 或 /e 修饰符，采用安全替代实现（如 preg_replace_callback）。
XSS	E:\TaintScan\examples\visualization_demo.php	166	echo	$value	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\visualization_demo.php	176	echo	$global_tainted	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\visualization_demo.php	199	echo	$data	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\xss_comprehensive.php	4	echo	$name	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\xss_comprehensive.php	7	print	$msg	风险：XSS；输出：print；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\xss_comprehensive.php	13	echo	$safe1	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。
XSS	E:\TaintScan\examples\xss_comprehensive.php	23	echo	$b	风险：XSS；输出：echo；原因：输出包含不可信输入且未转义；建议：对输出进行正确转义（htmlspecialchars/htmlentities），并确保正确的上下文与 Content-Type。

Figure 6. Web report diagram
图 6. Web 报告示意图

平台输出结果采用稳定化设计以便于复核。各发现项保留 file、line、category、sink、sanitized 等基本字段，各污点路径输出风险等级、节点数、消毒状态及 source_node/sink_node 摘要。可视化部分使用 nodes/edges 结构呈现，节点包含位置和类别信息，边表示参数传递、赋值、返回值传播等关系，便于前端图形化展示，也可导入其他审计工具或用于构造训练数据集。

3.4.5. AI 辅助

为降低人工分析大量检测结果的工作量，平台设计了大模型 API 调用功能，支持主流服务商的 API 接口。代码中对 prompt 进行了封装处理，防止通过大模型攻击造成敏感信息泄露如图 7 所示。

Figure 7. Large model call interface
图 7. 大模型调用界面

4. 实验与评估

4.1. 数据集

实验采用三类数据：DVWA (典型 SQL 注入与 XSS)；自建样例(反序列化链、命令执行、文件包含/路径穿越)如图 8 所示；开源 PHP 项目(验证跨文件路径、缓存与分页的工程可用性)。

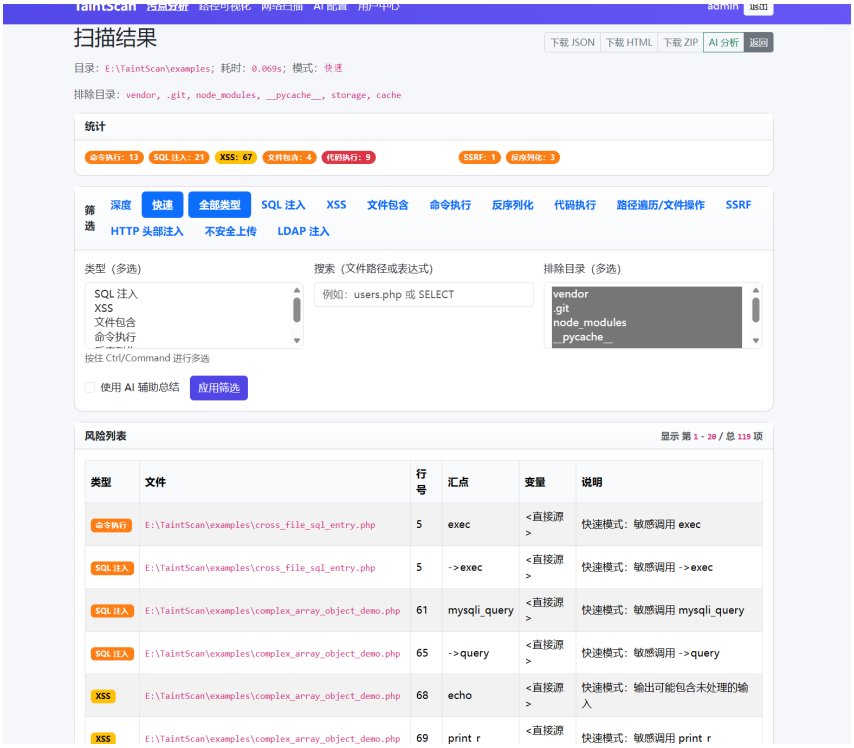

Figure 8. Self-built sample data graph
图 8. Web 自建样例数据图

4.2. 指标与对比设置

评估指标包括：人工复核准确性、已知样例覆盖趋势、单次扫描耗时、缓存命中率，以及路径证据对审计时间的影响。对比实验包括 quick/deep/functions 三模式消融，以及路径归一化开关对漏扫的影响。静态污点分析容易产生误报，实验中将可达性复核作为人工确认环节，并参考符号执行验证可达性的方法解释误报来源。

4.3. 对比与消融

Table 2. Example categories and sinks in TaintScan
表 2. TaintScan 中类别与汇的示例

类别	典型源	典型汇点(sink)	说明
SQL 注入	$_GET/$_POST	mysql_query/PDO::query	优先提示参数化与拼接位置
XSS	路由参数/表单	echo/模板输出	匹配编码消毒器与输出上下文
命令执行	HTTP 输入	system/exec/passthru	高危，弱消毒不降级为安全
文件包含/遍历	路径参数	include/require/file_put_contents	结合 realpath 等路径规约判断

对三种模式在同一数据集上进行对比：快速适合快速排查但误报率较高，深度对单文件分析最准确，函数模式在跨文件分析上有相对优势，适用于入口分散的框架项目。平台记录强弱消毒及未知消毒的情况，用于分析误报和漏报来源。表 2 列出了平台内置的典型漏洞类别及汇点示例，说明了规则覆盖范围及报告字段含义。

消融实验考察了两组配置的影响。关闭跨文件只保留单文件传播时，虽然精度较高但会丢失完整的证据链，导致能识别出风险点却无法准确定位漏洞入口。关闭路径归一化后，在包含符号链接、UNC 路径或混合分隔符的代码仓库中容易出现漏扫和重复扫描问题，影响缓存命中率和结果一致性。

跨文件摘要和路径归一化是系统可用的重要组成部分，其作用在于提高可解释和复现性，而非单纯增加单次扫描的发现项数量。

4.4. 实验案例

为验证平台在复杂面向对象场景下的检测能力，模拟现代 PHP 框架(如 Laravel/Symfony)中常见的依赖注入模式，传统基于正则或单文件 AST 的工具极易漏报此类漏洞。

漏洞链路横跨三个文件，涉及输入接收、逻辑处理与底层实现的分离(代码片段如图 9)：

- **入口文件(Controller.php)**：接收用户输入，使用相对路径加载依赖。
- **接口定义(LoggerInterface.php)**：定义抽象行为。
- **危险实现(SystemLogger.php)**：包含__destruct 魔术方法与命令执行汇点。

扫描器识别到__destruct 方法引用了受污染的$this->cache，且该变量最终进入 system()函数。由于未发现有效的消毒器(如 escapeshellarg)，平台判定为远程代码执行(RCE)漏洞，可以进行任意命令执行，为高风险。

接由 ai 辅助分析该风险后给出的内容为“代码逻辑中$logger 实例化过程由工厂模式控制，若工厂可能返回 SystemLogger 实例，则攻击者可通过构造特定的 cmd 参数在脚本执行结束时触发 system 命令执行。建议在 SystemLogger::log 中对输入进行严格过滤，或禁用 system 函数。”

遵循平台的保守原则以及命中的高风险度和 ai 的建议，人工进行二次复查确定漏洞可利用性。

```php
// [File 1] /var/www/html/controllers/Controller.php
include_once '../services/LogFactory.php'; // 触发路径归一化
$userInput = $_GET['cmd']; // 1. 污点源 (Source)
$logger = LogFactory::getLogger('system');
// 2. 跨文件传递：调用接口方法，具体实现未知
$logger->log("User input: " . $userInput);

// [File 2] /var/www/html/services/SystemLogger.php
class SystemLogger implements LoggerInterface {
    public function log($msg) {
        $this->cache = $msg; // 3. 污点暂存入对象属性
    }
    // 4. 魔术方法触发：对象销毁时执行
    public function __destruct() {
        // 5. 汇点 (Sink)：system 属于高危函数
        system("echo " . $this->cache);
    }
}
```

Figure 9. Code example diagram
图 9. 代码示例图

4.5. 结果与分析

实验观察到，functions 模式能有效缩短人工复核，将控制器到模型、模型到数据库、路由参数到模板输出等跳转形成证据链，降低代审成本。从 DVWA 及自建样例的实验结果看，与 quick 模式比，functions 模式在同等规则覆盖下召回率提高约 10 个百分点，且因路径证据更完备，人工确认时间明显缩短。

路径归一化在 Windows 及包含 UNC/软链接的场景下减少了路径不存在和重复扫描问题，使 Web 与 CLI 的结果更易复现。缓存命中后可将二次扫描耗时压缩到首次扫描的一小部分，使平台更适合在 CI/CD 中频繁运行。表 3 从中间表示、上下文敏感度及工程落地性等维度，详细对比了本文方法与现有主流工具的差异。

Table 3. Method comparison chart
表 3. 方法对比图

维度	细分指标	Pixy	RIPS	商业/现代工具 (Generic SOTA)	本文方法
核心架构	中间表示(IR)	P-Tac (线性三地址码)	Block-based CFG (控制流图)	CPG (代码属性图) 或 Custom IR	混合表示 (AST + 语义向量)
精度分析	上下文敏感度	k-limiting (固定深度)	Context-sensitive (受限)	1-CFA/2-CFA	按需敏感 (Demand-driven)
	跨文件分析	包含文件解析	需完整构建	全量索引	增量索引/跨库检索
覆盖广度	OO 支持	不支持	部分 (类内分析为主)	支持 (基于类型推断)	增强型 (多态解析 + 反射模拟)
	框架支持	无	插件/配置模式	预置规则库	自动 Hook/ 启发式识别
工程落地	验证机制	无	无	静态可达性分析	大模型辅助验证
	报告输出	行号 + 变量名	调用栈列表	数据流图/污点路径	自然语言描述 + 修复建议

4.6. 局限性

由于 PHP 动态特性及运行时依赖影响，平台在 eval、动态模板执行、长反序列化 gadget 链等场景都可能存在不确定边或漏报。故平台采用的策略是：将不确定性透明地呈现给用户，提供可复核的最小证据链，后续可引入更精细的环境建模及可达性验证来降低误报[5]。

5. 结论

本文基于污点分析的 PHP 应用威胁检测平台，用快速扫描、深度扫描、函数摘要三种模式分别解决预筛、精扫、跨文件解释诸种需求，又以函数摘要及路径可视化来增强可解释性，再以路径归一化、缓存分页、基于角色的访问控制、报告安全渲染等手段提高项目的可用性。本系统的主要优势是利用三模式对于不同情况下的代码进行快速高效的分析，并提供 AI 辅助高效了解代码的威胁点以及修复方式，同时可导出多样式的报告便于后续复核。并界定了角色的访问权限，安全地保存相关敏感信息。但面对复杂的扩展反序列化 gadget 链及动态特性场景，组合符号执行与动态验证仍有不足。

基金项目

江苏省大学生创新创业项目(xcx2025341, xcx2025350)，徐州工程学院大学生创新创业项目(xcx2025356)的阶段性成果之一。

参考文献

[1] W3Techs (2026) Usage Statistics of Server-Side Programming Languages for Websites.

[2] OWASP Foundation (2021) OWASP Top 10: 2021 the Ten Most Critical Web Application Security Risks.

[3] 曹凯, 何晶, 范文庆, 黄玮. 基于污点分析的 PHP 漏洞检测[J]. 传媒大学学报, 2019, 27(1): 33-38.

[4] 王国峰, 唐云善, 徐立飞. 基于污点分析的 SQL 注入漏洞检测[J]. 信息技术, 2024(2): 185-190.

[5] 刘行波, 李源林, 余明俊, 等. 基于污点分析与符号执行的 Web 漏洞检测[J]. 计算机应用与软件, 2022, 39(11): 297-303.

[6] Jovanovic, N., Kruegel, C. and Kirda, E. (2006) Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities.

[7] RIPS Technologies (2015) RIPS—Static Code Analysis for PHP.

[8] Zhao, J., Zhu, K., Lu, C., Zhao, J. and Lu, Y. (2025) Benchmarking Static Analysis for PHP Applications Security. *Entropy*, **27**, Article No. 926. https://doi.org/10.3390/e27090926