煜鉴技术白皮书

AI驱动的多引擎代码安全检测平台

版本：v1.0

发布日期：2026年10月

文档密级：公开

长沙煜旺智能科技有限公司

目录

1. 执行摘要

2. 关于煜鉴

3. 代码安全检测的挑战

4. 煜鉴解决方案

5. 技术架构

6. 七引擎体系（E1-E7）

7. AI语义分析引擎

8. 检测能力与性能

9. 与业界方案对比

10. 典型应用场景

11. 技术路线图

12. 结论

附录

煜鉴技术白皮书

AI驱动的多引擎代码安全检测平台

版本：v1.0

发布日期：2026年10月

文档密级：公开

1. 执行摘要

煜鉴（Xinpect）是一款面向AI时代的多引擎代码安全检测平台，采用多引擎并行扫描架构，结合AI语义分析技术，为软件研发团队提供从代码提交到发布的全生命周期安全检测服务。

平台核心引擎版本 v6.1.0，内置 11,380+ 条检测规则，覆盖语法规范、安全漏洞、性能瓶颈、架构合规、AI语义分析等七大维度。通过 ProcessPoolExecutor 文件级并行 + ThreadPoolExecutor 引擎级并行的双层并发架构，实现高效扫描与精准检测的平衡。

核心价值：

- 多引擎并行：七引擎并行扫描 + 三层分级防护（规范层/质量层/防护层），覆盖代码安全全维度
- AI语义分析：基于AST抽象语法树 + DeepSeek大模型双重驱动，三级置信度误报过滤，精准率 94.5%
- DevOps原生：REST API + CI/CD流水线 + Git Webhook，无缝嵌入研发流程
- PoC实弹验证：对高危漏洞自动生成攻击载荷，Docker沙箱隔离验证，输出确认/可疑/排除三级结论

技术指标：

- 支持语言：Python、JavaScript、TypeScript、Java、Go、PHP、Ruby、Kotlin、Swift、Rust、Shell、SQL、Lua、Scala 等 15+ 主流语言
- 检测规则：11,380+ 条（含安全规则 189 条、性能与可靠性 4,127 条、业务安全 1,983 条等）
- 扫描模式：基础版（规则引擎）/ 完整版（AI全增强）
- Benchmark 精确率：94.55%（P0 关键漏洞精确率 100%）

2. 关于煜鉴

2.1 公司背景

煜鉴由长沙煜旺智能科技有限公司研发，是一家专注于AI代码安全检测的创新型企业。团队核心成员拥有8年餐饮连锁运营、互联网产品开发、安全检测等跨领域实战经验，深刻理解中小型研发团队在代码安全检测领域的痛点。

2.2 产品定位

煜鉴对标国际领先的代码安全检测工具（如SonarQube、Checkmarx），专注于为中小型研发团队提供高性价比、易集成、低误报的代码安全检测服务。

产品采用 SaaS 模式运营，支持 Web 界面上传扫描、Git 仓库集成、CI/CD 流水线接入、REST API 自定义调用等多种使用方式。同时支持私有化部署，满足企业级数据安全需求。

2.3 核心理念

- 安全第一：代码安全是软件质量的基石，宁可报错也不漏报
- AI驱动：用AI技术提升检测精度，将误报率降至行业平均水平的 1/2
- 开发者友好：降低安全检测门槛，让每个开发者都能写出安全代码
- 产品诚信底线：扫描引擎不可达时直接报错，绝不用 fallback 假结果

---

3. 代码安全检测的挑战

3.1 传统方案的局限

规则匹配的局限性：

- 基于正则表达式的规则匹配，误报率高（行业平均 30-40%）
- 无法理解代码上下文，容易产生假阳性
- 难以检测复杂的业务逻辑漏洞

性能与精度的矛盾：

- 深度检测耗时长，影响开发效率
- 快速检测精度低，漏报率高

集成复杂度：

- 传统工具需要复杂的配置和部署
- 与现有 DevOps 流程集成困难

3.2 AI时代的代码安全挑战

AI生成代码的安全风险：

- AI辅助编程（如 Copilot）生成的代码可能包含安全隐患
- 开发者对AI生成代码的安全审查不足
- 缺乏针对AI生成代码的专项检测能力

开源组件的安全隐患：

- 开源依赖组件的漏洞频发
- 供应链攻击风险上升
- 缺乏有效的依赖组件安全评估

4. 煜鉴解决方案

4.1 产品概述

煜鉴提供 SaaS 化的代码安全检测服务，支持 Web 界面上传、Git 仓库集成、CI/CD 流水线接入、REST API 调用等多种使用方式。

扫描服务运行于独立进程（端口 8620），采用零外部依赖的 Python 标准库 http.server 构建，支持批量扫描接口（/scan/batch）和扫描结果查询（/result/{scan_id}）。

4.2 核心能力

多引擎并行扫描：

引擎	名称	职责
E1	语法规范引擎	代码语法错误、格式规范、编码风格检测，基于 Tree-sitter 多语言解析
E2	安全漏洞引擎	SQL注入、XSS、命令注入、硬编码密钥等安全漏洞检测，覆盖 16+ 语言专用规则
E3	性能检测引擎	性能瓶颈、资源泄漏、死循环、AST深度语义分析
E4	代码质量引擎	圈复杂度、函数长度、重复代码、测试质量检测
E5	架构规范引擎	模块耦合度、循环依赖、分层架构、数据合规审计
E6	依赖扫描引擎	开源组件漏洞检测（SCA）、许可证合规、版本风险评估
E7	AI语义引擎	AI幻觉检测、语义去噪、部署就绪性检查、LLM安全扫描

三层分级防护体系：

- 规范层（代码骨架）：E1语法 + E4质量 + E5架构 + 专项引擎，权重均等
- 质量层（代码血肉）：E3语义分析 40% + 性能可靠性 30% + 可观测性 30%
- 防护层（代码皮肤）：E2安全审计 40% + 业务安全 35% + 依赖SCA 25%

AI语义分析：

- 基于 AST 抽象语法树的代码结构分析
- DeepSeek 大模型驱动的代码语义理解（fast/standard/deep 三档模型）
- 三级置信度误报过滤：confirmed（确认）/ review（待确认）/ dismissed（误报）

PoC 自动验证：

- 对高危安全漏洞自动生成 PoC 攻击载荷
- Docker 沙箱隔离环境实弹验证
- 输出确认/可疑/排除三级结论，进一步降低误报

DevOps 无缝集成：

- REST API 支持自定义集成
- CI/CD 配置自动生成
- Git Webhook 触发自动扫描
- 扫描结果回写 Pull Request

---

5. 技术架构

5.1 系统架构图



5.2 核心技术栈

层级	技术选型	说明
Web 服务	Python Flask	主站 API + 用户管理
扫描引擎	Python http.server	零外部依赖，独立进程，端口 8620
语法解析	Tree-sitter	多语言增量解析，支持 15+ 语言

层级	技术选型	说明
AI 引擎	DeepSeek（火山引擎）	fast/standard/deep 三档模型
数据库	SQLite（主站）+ SQLite（引擎）	轻量部署，可升级 PostgreSQL
并行计算	ProcessPoolExecutor + ThreadPoolExecutor	文件级 + 引擎级双层并行
增量缓存	SHA-256 文件哈希	文件未变直接返回缓存，重复扫描减少 90%+ 计算
前端	原生 HTML/CSS/JavaScript	零框架依赖，极速加载
部署	腾讯云轻量服务器	Docker 容器化，支持私有化部署

5.3 安全机制

- 数据隔离：多租户数据严格隔离，所有数据库操作必带 user_id
- 传输加密：全站 HTTPS 加密传输
- 访问控制：基于 JWT 的身份认证 + API Key 绑定 user_id 不可跨用户
- 代码保护：扫描完成后自动删除用户上传的代码
- API 鉴权：公网 API 必须携带鉴权密钥，内网可配置免鉴权
- 资源保护：内存监控 + 自适应批次 + 限流，单文件 > 5MB 自动跳过

---

6. 七引擎体系（E1-E7）

6.1 E1 语法规范引擎

检测范围：

- 代码语法错误检测
- 编码风格一致性检查
- 命名规范验证
- 格式规范检测

技术实现：

- 基于 Tree-sitter 多语言解析器，支持增量语法分析
- 每种语言独立 Parser，精确到 AST 节点级别
- 内置语言专用解析器：Python、JavaScript、TypeScript、Java、Go、C/C++、Rust、Kotlin、Swift、PHP、Ruby、Lua、Scala 等
- 规则ID格式：E1-SYN-xxx

6.2 E2 安全漏洞引擎

检测范围：

- SQL注入、XSS跨站脚本、命令注入
- 路径遍历、硬编码密钥、不安全反序列化
- CSRF跨站请求伪造、SSRF服务端请求伪造
- 敏感信息泄露、配置文件安全
- 业务逻辑安全（支付安全、权限绕过、数据越权）

技术实现：

- 16 个语言专用安全子引擎：
  - e2_security（核心）+ e2_security_javascript + e2_security_typescript
  - e2_security_go + e2_security_php + e2_security_ruby
  - e2_security_kotlin + e2_security_swift + e2_security_cpp
  - e2_security_shell + e2_security_sql + e2_security_html
  - e2_security_lua + e2_security_scala + e2_security_vbnet
  - e2_security_plsql + e2_security_tsql + e2_security_ansible
  - e2_security_github_actions + e2_security_cloudformation
- 业务逻辑安全专项：支付边界检测、IDOR越权、写幂等性、事务一致性、缓存穿透/不一致
- 规则ID格式：E2-SEC-xxx

6.3 E3 性能检测引擎

检测范围：

- 性能瓶颈识别（低效循环、冗余计算）
- 资源泄漏检测（文件/连接未关闭）
- 内存泄漏风险
- 并发安全检测（竞态条件、死锁）
- AST 深度语义分析（未定义变量、异常静默、死代码）

技术实现：

- 基于 Python AST 模块的深度语义分析（e3_python_ast）
- 作用域分析：变量定义/引用追踪、未定义变量检测
- except 静默失败检测：识别空 except 块、缺少日志记录
- 资源泄漏检测：文件句柄、数据库连接、网络连接的 close 检查
- 控制流分析：死代码路径识别

6.4 E4 代码质量引擎

检测范围：

- 圈复杂度分析
- 函数长度检测
- 重复代码识别
- 测试质量检测（断言质量、用例完整性）
- Prompt 质量评估

技术实现：

- 基于 AST 的复杂度计算
- 可维护性指标量化
- 测试覆盖率评估
- 支持自定义阈值配置（如最大函数行数 50、最大文件行数 500、最大参数数 5、最大嵌套深度 4）
- 规则ID格式：E4-QUAL-xxx

6.5 E5 架构规范引擎

检测范围：

- 代码架构合规性审计
- 循环依赖检测（基于依赖图分析）
- 模块耦合度分析
- 分层架构违规检测
- API 文档覆盖度检查
- 数据一致性校验（字段约定、枚举一致性、外键约束）
- 配置完整性审计（未定义配置项、未使用配置项、开关一致性）

技术实现：

- 基于依赖图的架构分析（_build_dependency_graph + _detect_cycles）
- 分层违规检测（_check_layer_violations）
- 路由文档一致性校验（代码路由 vs 文档路由比对）
- Flask 路由冲突检测
- WebSocket 超时清理检查
- 数据层安全：事务保护、幂等性、软删除、外键约束、级联风险
- 规则ID格式：E5-ARCH-xxx

6.6 E6 依赖扫描引擎

检测范围：

- 第三方依赖已知漏洞检测（CVE）
- 过期版本检测
- 许可证合规校验
- 供应链攻击防范

技术实现：

- 依赖关系解析（package.json / requirements.txt / go.mod / pom.xml 等）
- 漏洞库匹配（CVE、CNVD）
- 组件评分体系：S/A/B/C 四档（90/75/60 分阈值）
- SCA（软件成分分析）完整流程
- 规则ID格式：E6-CMP-xxx

6.7 E7 AI语义引擎

检测范围：

- AI 幻觉检测（e7_ai_hallucination）
- 语义去噪（e7_ai_semantic_denoising）
- LLM 安全扫描（e7_llm_semantic_scan）
- AI 健康度检查（e7_ai_health）
- 部署就绪性检查（e7_deployment_readiness）

技术实现：

- AI 编排器框架（ai_engine/ 目录）
- 仅在 scan_mode='full' 且 is_vip=True 时启用
- AI 错误隔离：AI 增强失败不影响规则引擎结果返回
- 多模型调度：DeepSeek-v4-flash（快速）+ DeepSeek-v4-pro（深度分析）

---

7. AI语义分析引擎

7.1 技术原理

AST 抽象语法树分析：

煜鉴的 AST 分析引擎（e3_python_ast）是一个完全自主研发的轻量级分析器，不依赖外部 LLM，通过 Python 标准库 ast 模块实现深度代码分析：

```
# 示例：AST 分析检测 SQL 注入
def execute_query(user_input):
query = "SELECT * FROM users WHERE id = " + user_input
db.execute(query)
```

AST 解析结果：

```
FunctionDefinition: execute_query
├── Parameter: user_input ← 污点源
├── Assignment: query
│ └── BinOp: +
│ ├── String: "SELECT * FROM users WHERE id = "
│ └── Name: user_input ← 污点传播
└── Call: db.execute ← 敏感函数（Sink）
└── Name: query ← 污点到达
```

分析维度：

- 作用域分析：变量定义/引用追踪（_Scope 类）
- 未定义变量检测（_check_undefined_var）
- except 静默失败检测（识别空 except + 缺少日志）
- 资源泄漏检测（with 语句缺失、连接未关闭）
- 死代码路径识别

7.2 大模型语义增强

AI 客户端架构（3S 架构）：

煜鉴的 AI 客户端采用模块化 3S 架构，零外部依赖（纯 Python 标准库 http.client + json），支持三种后端模式：

模式	后端	用途
节能阀网关模式	AI_GATEWAY_URL	统一网关调度，成本优化
直连模式	AI_UPSTREAM_URL	直连火山引擎豆包
OpenAI 兼容模式	OPENAI_API_KEY	兼容 OpenAI 接口

三档模型调度：

档位	模型	用途	最大Token
fast	DeepSeek-v4-flash	简单判断、快速筛选	1,024
standard	DeepSeek-v4-flash	常规分析、修复建议	2,048

档位	模型	用途	最大Token
deep	DeepSeek-v4-pro	复杂漏洞分析、深度语义	4,096

7.3 误报过滤算法

三级置信度过滤机制：

规则引擎初筛后，AI 对高/中危结果做二次语义判读：

判定结果	含义	处理方式
confirmed	确认是真实问题	保留告警
review	边界情况，需人工确认	标记待确认
dismissed	误报，可忽略	过滤移除

过滤策略：

- 安全类（SEC 前缀）和正确性类（COR 前缀）的中高危问题优先过 AI
- 风格/命名类跳过 AI 过滤（本身误报率低）
- 核心原则：宁漏勿误 —— 如果不确定是否误报，标记为 review 而非 dismissed

过滤效果：

- 传统规则匹配：误报率 30-40%（行业平均）
- 煜鉴 AI 语义过滤：精确率 94.55%（Benchmark 实测）

7.4 智能修复建议

生成逻辑：

- 1. 识别漏洞类型和位置
- 2. 提取代码上下文（extract_code_context）
- 3. 构建提示词（build_user_prompt）
- 4. 调用 AI 大模型生成修复建议
- 5. 批量处理优化（_fix_suggestion_batch）

示例：

```
漏洞：SQL注入
位置：app.py 第42行

修复建议：
使用参数化查询替代字符串拼接

修复前：
query = "SELECT * FROM users WHERE id = " + user_input
db.execute(query)

修复后：
query = "SELECT * FROM users WHERE id = ?"
db.execute(query, (user_input,))
```

8. 检测能力与性能

8.1 检测规则统计

规则分布（基于生产环境实际数据）：

引擎	规则数量	覆盖范围
E1 语法规范	—	语法错误、格式规范（基于 Tree-sitter 解析）
E2 安全与依赖审计	189	SQL注入、XSS、命令注入、硬编码密钥等
E3 性能与可靠性	4,127	性能瓶颈、资源泄漏、并发安全、AST语义
E4 代码质量	1,089	复杂度、函数长度、重复代码、测试质量
E5 架构与数据合规	489	循环依赖、模块耦合、分层违规、数据一致性
E6 组件与SCA	1,534	开源组件漏洞、许可证合规、版本风险
E7 AI语义	823	AI幻觉、语义去噪、部署就绪性
业务安全	1,983	支付安全、权限绕过、数据越权、事务一致性
专项引擎	1,146	前端组件合规、小程序连接完整性
合计	11,380	全维度覆盖

> 注：以上数据来源于生产环境 /opt/xinpect-scanner-green/rules/brain_rules_bundle.json（2026-10-20 统计）

8.2 性能指标

Benchmark 测试数据（离线基准测试 2026-10-13）：

指标	数值	说明
测试样本	132 个文件	覆盖 Python/JS/Java/Go/PHP
已知漏洞样本	52 个	人工标注的正样本
整体精确率	94.55%	—
整体召回率	64.2%	—
F1 Score	76.47%	—
误报率	5.88%	仅 3 个误报
P0 关键漏洞精确率	100%	45 TP / 0 FP
P0 关键漏洞召回率	65.22%	—
P0 F1 Score	78.95%	—

分语言 Benchmark 表现：

语言	精确率	召回率	F1 Score
Python	90.48%	76.00%	82.61%
JavaScript	100%	60.00%	75.00%
Java	100%	71.43%	83.33%
Go	100%	54.55%	70.59%
PHP	88.89%	50.00%	64.00%

> 注：以上数据来源于 benchmark_results 表（bench_offline_20261013）

生产环境扫描统计（截至 2026-10-20）：

指标	数值
累计扫描次数	773 次

指标	数值
服务用户数	12 个
平均代码行数	1,350 行/次
最大单次扫描	29,318 行
平均文件数	14 个/次
平均发现问题	64 个/次
平均得分	86.6 分

8.3 并行架构性能

双层并行调度：

- 文件级并行：ProcessPoolExecutor，最大 4 Worker 进程，绕过 GIL 限制
- 引擎级并行：ThreadPoolExecutor，各引擎独立线程执行
- SHA-256 增量缓存：文件未变直接返回缓存结果，重复扫描减少 90%+ 计算
- 自适应进程池：根据文件数量自动调整 Worker 数

资源保护机制：

- ResourceMonitor 内存监控 + 自适应批次
- 大文件过滤：单文件 > 5MB 自动跳过
- 文件数上限：超出默认 50,000 分批处理
- 总超时保护：超时后返回已完成结果
- MemoryError 专项捕获 + 优雅降级

8.4 语言支持

已支持语言（15+）：

类别	语言
后端语言	Python、Java、Go、PHP、Ruby、Kotlin、Swift、Lua、Scala
前端语言	JavaScript、TypeScript
系统语言	C、C++、Rust
数据库	SQL、PL/SQL、T-SQL
脚本/配置	Shell/Bash、Terraform、Ansible、YAML/JSON

类别	语言
小程序	WXML、WXSS
前端框架	Vue、HTML、SCSS/Less/CSS

> 完整的代码文件扩展名支持列表见 CODE_EXTS 配置，覆盖 50+ 种文件类型

9. 与业界方案对比

9.1 功能对比

功能	煜鉴	SonarQube	Checkmarx
检测引擎	7+ 引擎并行	单引擎	多引擎
AI语义分析	DeepSeek 三档模型	无	基础
误报过滤	三级置信度 AI 过滤	手动标记	部分自动
PoC 验证	Docker 沙箱		
中文支持	原生	部分	无
SaaS 服务			
私有化部署			
CI/CD 集成	REST API + Webhook		
检测规则	11,380+	5,000+	10,000+
精确率	94.55%	~80%	~85%
价格	低	中	高

9.2 核心优势

vs SonarQube：

- 多引擎并行，检测维度更全面（11,380 vs 5,000+ 规则）
- AI 语义分析，精确率更高（94.55% vs ~80%）
- PoC 自动验证，真正确认漏洞可用性
- 中文原生支持，更适合国内团队
- 价格更低，性价比更高

vs Checkmarx：

- AI 驱动的三级误报过滤
- 智能修复建议自动生成
- 价格仅为 Checkmarx 的 1/10
- 更适合中小型团队

9.3 适用场景

煜鉴最适合：

- 中小型研发团队（10-100人）
- 需要快速集成的 DevOps 团队
- 预算有限但重视代码安全的企业
- 国内团队（需要中文支持和合规要求）
- AI/软件公司（精准客群）

不适合：

- 超大型企业（1000+ 开发者）
- 需要 CMA/CNAS 认证的场景（暂不支持）

---

10. 典型应用场景

10.1 场景一：AI/软件公司代码安全审计

背景： 某 AI 软件公司在产品发布前需要对核心代码进行安全审计

流程：

- 1. 通过 REST API 上传代码仓库
- 2. 煜鉴自动执行 E1-E7 全引擎扫描
- 3. AI 语义引擎对高危结果进行二次验证
- 4. 生成包含评分、问题清单、修复建议的完整报告

收益：

- 首次扫描发现 63 个问题，得分 86 分
- 通过修复建议快速定位并修复关键漏洞
- 建立代码安全基线，持续跟踪改进

10.2 场景二：CI/CD 流水线集成

流程：

- 1. 代码推送到 Git 仓库
- 2. Webhook 自动触发煜鉴扫描
- 3. 扫描结果回写到 Pull Request
- 4. 质量门禁检查，不通过则阻止合并

收益：

- 自动化安全检测，无需人工介入
- 保证每次合并的代码都符合安全标准
- 安全评分趋势可视化

10.3 场景三：小程序安全专项检测

流程：

- 1. 上传微信小程序项目代码（WXML/WXSS/JS）
- 2. 专项引擎检测组件契组合规性
- 3. 检查跨文件连接完整性
- 4. 验证属性约束和嵌套规则

收益：

- 覆盖小程序特有的安全检测维度
- 前端组件合规性自动检查
- 减少小程序审核被拒风险

---

11. 技术路线图

11.1 短期（2026 Q4）

- [ ] 支持更多编程语言（C/C++ 深度支持、Rust）
- [ ] 优化 AI 语义分析精度（召回率提升至 75%+）
- [ ] 完善 PoC 自动验证引擎（扩展更多漏洞类型）
- [ ] 批量扫描能力增强

11.2 中期（2027）

- [ ] 申请 ISO 27001 认证
- [ ] 参加信通院代码安全检测能力评估
- [ ] 支持 IAST 交互式应用安全测试
- [ ] 推出企业版（多团队管理、私有化部署增强）

11.3 长期（2028+）

- [ ] 申请 CMA/CNAS 认证
- [ ] 推出 AI 安全编码助手（实时检测+修复）
- [ ] 建立代码安全知识库（培训+认证）
- [ ] 拓展海外市场

---

12. 结论

煜鉴作为 AI 时代的代码安全检测平台，通过多引擎并行扫描和 AI 语义分析技术，为中小型研发团队提供了 高性价比、低误报、易集成的代码安全检测解决方案。

核心技术优势：

- 11,380+ 检测规则：覆盖语法、安全、性能、架构等全维度
- 94.55% 精确率：AI 三级置信度过滤，误报率远低于行业平均
- 双层并行架构：文件级 + 引擎级并行，大项目也能快速完成
- PoC 实弹验证：不只是规则匹配，真正验证漏洞可用性
- 15+ 语言支持：覆盖主流开发语言和技术栈

核心价值：

- 让每个开发者都能写出安全代码
- 在最早阶段发现并修复安全问题
- 降低代码安全检测的门槛和成本

附录

A. 术语表

术语	全称	说明
AST	Abstract Syntax Tree	抽象语法树
SAST	Static Application Security Testing	静态应用安全测试
IAST	Interactive Application Security Testing	交互式应用安全测试
SCA	Software Composition Analysis	软件成分分析

术语	全称	说明
CVE	Common Vulnerabilities and Exposures	通用漏洞披露
CWE	Common Weakness Enumeration	通用缺陷枚举
PoC	Proof of Concept	概念验证
SaaS	Software as a Service	软件即服务

B. 参考资料

- OWASP Top 10：https://owasp.org/www-project-top-ten/
- CWE Top 25：https://cwe.mitre.org/top25/
- 等保2.0：GB/T 22239-2019《信息安全技术 网络安全等级保护基本要求》

C. 联系方式

- 官网：https://xinpect.xingwangzhineng.com
- 邮箱：hdyabcd@163.com
- 电话：13789007691
- 公司：长沙煜旺智能科技有限公司

文档版本：v1.0 发布日期：2026年10月20日 文档密级：公开

数据来源：生产环境实际数据（2026-10-20 统计） 版权声明：© 2026 长沙煜旺智能科技有限公司 版权所有