

# 恶意代码分析与防治技术实验报告

## 一、实验名称

恶意代码扫描与防治平台的设计与实现

## 二、小组成员及分工

张肇秋\_2312796 总负责人，项目首席运营官，搭建完善整体的前后端框架

杨中秀\_2312323 Yara规则书写与搜集，检测

李胜林\_2313781 Sigma规则书写与搜集

杜泽琦\_2313508 Sigma规则检测数据，项目文档书写

## 三、实验目的

- 构建综合检测平台：**设计并实现一个集成了多种检测引擎的恶意代码分析平台，为用户提供便捷的威胁检测服务。
- 掌握静态分析技术：**通过集成 Yara 规则引擎，实现对二进制文件（如 PE 文件、脚本等）的静态特征扫描，识别已知的恶意代码家族。
- 探索动态日志分析：**引入 Sigma 规则和 Zircolite 工具，对 Windows 事件日志（EVTX）进行深度分析，识别潜在的攻击行为和异常操作（如横向移动、权限提升）。
- 漏洞关联与管理：**建立漏洞知识库，集成 CVE、CNVD 等数据，提供漏洞信息的查询与管理功能，辅助安全人员进行关联分析。

## 四、实验环境

- 操作系统：**Windows 10 / Windows 11
- 开发语言：**
  - 后端：Python 3.9
  - 前端：Vue.js
- Web 框架：**Flask (后端), Vue (前端)
- 数据库：**MySQL 8.0
- 核心引擎与工具：**
  - Yara**: 用于静态特征匹配 (`yara-python`)
  - Sigma / Zircolite**: 用于 EVTDX 日志分析
- 开发工具：**PyCharm / VS Code

## 五、实验内容与步骤

### 1. 系统架构设计

本项目采用典型的前后端分离架构，确保系统的可扩展性和维护性。

- 后端：**基于 Flask 框架构建 RESTful API，负责处理业务逻辑、调用检测引擎以及与数据库交互。

- **前端**: 基于 Vue.js 构建单页应用 (SPA)，提供文件上传、规则管理、结果展示等可视化交互界面。
- **数据存储**: 使用 MySQL 存储 Yara/Sigma 规则、漏洞信息 (CVE/CNVD)、扫描日志等持久化数据。

## 2. 功能模块实现

### (1) Yara 静态扫描模块

- **规则管理**: 实现了规则的上传接口，支持 .yar 单文件及 .zip 压缩包批量上传。为了提升匹配效率，系统在接收规则时会进行预编译处理。
- **扫描流程**:
  1. 用户上传待检测样本。
  2. 后端计算文件 SHA256 并保存临时文件。
  3. 加载数据库中状态为 enabled 的 Yara 规则。
  4. 调用 yara.match() 对样本进行全量扫描。
  5. 解析匹配结果，返回命中的规则名称、标签及元数据。

### (2) Sigma 日志分析模块

- **核心机制**: 利用 Zircolite 工具作为中间件，将 Sigma 规则转换为查询语句，对 Windows 事件日志 (.evtx) 进行匹配。
- **扫描流程**:
  1. 用户上传 .evtx 日志文件。
  2. 后端提取数据库中的 Sigma 规则并生成临时规则集。
  3. 调用 Zircolite 引擎分析日志文件。
  4. 解析输出结果，提取威胁等级 (Critical, High, Medium, Low) 及具体的日志证据。

### (3) 漏洞数据管理模块

- 实现了对 CVE、CNVD 漏洞数据的增删改查功能。
- 设计了 VulnData (漏洞信息), Product (受影响产品), Company (厂商) 等数据模型，并通过 SQLAlchemy ORM 进行管理，支持多维度的漏洞查询。

## 3. 关键技术点

- **规则预编译**: 为了解决大量 Yara 规则加载慢的问题，系统采用了规则预编译技术，显著减少了每次扫描时的初始化时间。
- **基于哈希的规则去重**: 在规则上传阶段，系统自动计算文件内容的 SHA256 哈希值，通过比对数据库中的哈希记录，有效防止了重复规则的录入，节省了存储空间并提升了检索效率。
- **Zip Slip 防护**: 在处理用户上传的规则压缩包时，加入了路径安全检查，防止恶意构造的压缩包进行路径穿越攻击。
- **统一 API 响应**: 封装了标准的 JSON 响应格式 (status, msg, data)，便于前后端数据交互和错误处理。
- **ORM 与数据完整性**: 后端采用 SQLAlchemy ORM 进行数据库操作，结合数据库层面的外键约束 (Foreign Key) 和级联删除 (Cascade)，确保了漏洞数据、产品信息与厂商信息之间的高度一致性。

# 六、实验结果与分析

## 1. 功能测试结果

- 样本上传与扫描：**平台能够正常接收用户上传的恶意样本和规则文件。在前端界面上上传测试样本后，系统成功返回了扫描结果，能够准确展示命中的 Yara 规则。
- 日志分析：**上传包含攻击痕迹的 `log_lab1.evtx` 文件，Sigma 引擎成功识别出 "Suspicious PowerShell Command" 等高危行为，并给出了详细的日志条目证据。

## 2. 准确率与性能分析

为了评估系统的检测能力，我们使用 VirusShare 数据集进行了多轮大规模测试（受限于虚拟机性能，部分测试在后端命令行模式下进行）。

测试数据统计：

| 测试批次 | 样本数量 | 命中数量 | 准确率   | 漏报率   |
|------|------|------|-------|-------|
| 第一批  | 490  | 200  | 40.8% | 59.2% |
| 第二批  | 2000 | 792  | 39.6% | 60.4% |
| 第三批  | 1880 | 787  | 41.8% | 58.2% |

结果分析：

- 准确率瓶颈：**目前的平均检测准确率约为 40% 左右。主要原因是本地集成的 Yara 规则库规模较小（初始约 800 条，后扩充至 1700 条），覆盖面有限，无法识别所有种类的恶意样本。
- 性能表现：**在处理单个样本时，平均耗时约 1 秒。但在大规模批量扫描时（如 2000 个样本），受限于虚拟机内存和 CPU 性能，耗时较长。
- 改进措施：**通过引入 GitHub 开源社区的高质量规则库（如 `100DaysofYARA`, `Yara-Rules`），规则数量翻倍后，检测能力有了一定提升。未来可进一步接入在线威胁情报接口，提升未知威胁的识别率。

# 七、实验总结

本次实验成功设计并实现了一个基于 Web 的恶意代码扫描平台。通过整合 Yara 和 Sigma 引擎，平台具备了“文件静态特征”和“日志行为特征”双重检测能力。

主要收获：

- 深入理解了 Yara 和 Sigma 规则的编写与匹配原理。
- 掌握了 Flask + Vue 全栈开发流程及前后端交互机制。
- 通过实际样本测试，认识到了特征匹配技术的局限性（依赖规则库）以及持续更新规则的重要性。

遇到的问题与解决：

- 问题：**大规模样本测试时虚拟机内存不足。
  - 解决：**采用分批测试策略，并编写后端自动化脚本（`yara_folder_scan_client.py`）绕过浏览器进行高效扫描。
- 问题：**部分 Yara 规则误报率较高。
  - 解决：**对规则进行筛选和优化，移除过于宽泛的特征匹配规则。

未来，可以进一步引入沙箱动态执行分析（Sandbox）和机器学习检测模型，构建更加立体和智能的防御体系。