恶意代码扫描平台技术报告

恶意代码扫描平台技术报告

1. 项目简介
2. 系统架构
    2.1 技术栈
    2.2 目录结构说明
3. 功能模块详解
    3.1 Yara 扫描模块
    3.2 Sigma 日志分析模块
    3.3 漏洞数据管理模块
4. 数据库设计
5. 接口设计 (API)
    5.1 通用接口
    5.2 Yara 相关
    5.3 Sigma 相关
    5.4 漏洞数据相关 (部分列举)
6. 部署说明（具体请看用户手册）

1. 项目简介

本项目是一个综合性的恶意代码分析与防治平台，旨在为安全研究人员和运维人员提供高效的威胁检测工具。平台集成了多种主流的安全检测引擎，能够对二进制文件进行静态特征扫描，以及对系统日志进行行为分析。

核心功能包括：

- **Yara 静态扫描**：利用 Yara 规则引擎，对上传的样本文件进行特征匹配，快速识别已知的恶意代码家族。
- **Sigma 日志分析**：利用 Sigma 规则和 Zircolite 工具，对 Windows 事件日志 (EVTX) 进行深度分析，检测潜在的攻击行为和异常操作。
- **漏洞数据管理**：集成 CVE、CNVD 等漏洞数据库，提供漏洞信息的查询与管理功能，辅助关联分析。

2. 系统架构

2.1 技术栈

本项目采用前后端分离的架构设计，技术选型如下：

- **后端开发语言**：Python 3.9
- **Web 框架**：Flask (轻量级、易扩展)
- **数据库**：MySQL 8.0 (数据持久化)
- **ORM 框架**：SQLAlchemy (数据库交互)
- **核心检测引擎**：
    - **Yara**: `yara-python` 库，用于文件特征匹配。
    - **Sigma/Zircolite**: `zircolite` (Python 编写的独立工具)，用于将 Sigma 规则应用于 EVTX 日志。

- **前端**：Vue.js (推测，基于 `dist` 目录结构)

2.2 目录结构说明

后端项目主要位于 `vuln_backend` 目录下，关键结构如下：

```
vuln_backend/
├── src/
|   ├── apps/
|   |   ├── models/          # 数据模型定义 (SQLAlchemy Table)
|   |   ├── routes/          # API 路由定义 (Blueprint)
|   |   ├── services/        # 核心业务逻辑实现 (扫描、上传、数据处理)
|   |   └── utils/           # 工具函数 (数据库连接、API 响应封装)
|   ├── config.py            # 项目配置文件
|   ├── extension.py         # 扩展初始化 (DB 等)
|   └── run.py               # 项目启动入口
├── third_party/             # 第三方工具依赖
|   ├── Yara/                # Yara 相关可执行文件
|   └── Zircolite/           # Zircolite 工具及脚本
└── requirements.txt         # Python 依赖列表
```

3. 功能模块详解

3.1 Yara 扫描模块

功能描述:
该模块允许用户上传二进制文件（如 `.exe`, `.dll`, `.bin`），后端加载数据库中存储的 Yara 规则对文件进行扫描，返回命中的规则名称、标签及元数据。

实现原理:

1. **规则管理**：支持单文件 (`.yar`) 和压缩包 (`.zip`) 上传。上传的规则被解析并存储在 `yara_rule` 表中，同时为了提高性能，规则在入库时或扫描前会被预编译。
2. **扫描流程**:
    - 接收前端上传的文件流。
    - 计算文件 SHA256 哈希。
    - 从数据库获取状态为 `enabled` 的规则。
    - 将预编译规则 (`compiled_rule`) 写入临时目录。
    - 调用 `yara.load()` 加载规则并执行 `match()` 方法。
    - 格式化匹配结果并返回。

关键代码位置:

- 扫描逻辑：`src/apps/services/MalYaraScan.py`
- 上传逻辑：`src/apps/services/MalYaraUpload.py`

3.2 Sigma 日志分析模块

功能描述:
该模块专注于 Windows 事件日志分析。用户上传 `.evtx` 文件，系统利用 Sigma 规则集检测日志中的攻击痕迹（如特权提升、横向移动、持久化等）。

实现原理:

1. **规则管理**：支持上传 Sigma 规则（YAML 格式）。规则以 JSON/YAML 形式存储在 `sigma_rule` 表中。
2. **扫描流程**:
    - 接收 `.evtx` 文件并保存到临时目录。
    - 从数据库导出启用的 Sigma 规则，生成对应的 YAML 文件群。
    - 调用第三方工具 **Zircolite** (`src/apps/services/MalSigmaScan.py` 中通过 `subprocess` 或直接调用库)。
    - Zircolite 解析 EVTX 文件并根据规则进行匹配，生成 JSON 结果。
    - 后端解析 JSON 结果，提取告警标题、等级、命中次数及证据字段（Evidence），返回给前端。

关键代码位置:

- 扫描逻辑：`src/apps/services/MalSigmaScan.py`
- 上传逻辑：`src/apps/services/MalSigmaUpload.py`

3.3 漏洞数据管理模块

功能描述:
提供对漏洞知识库的维护能力，支持对 CNVD、CVE、相关产品及厂商信息的增删改查。

数据实体:

- **VulnData**: 漏洞核心信息（ID, 描述, 评分等）。
- **Product**: 受影响的产品信息。
- **Company**: 厂商信息。
- **Affect**: 漏洞与产品的关联关系。

关键代码位置:

- 数据服务：`src/apps/services/CnvdDataInfoImpl.py`, `CveDataInfoImpl.py` 等。

4. 数据库设计

系统主要使用 MySQL 数据库，通过 SQLAlchemy 进行操作。主要数据表如下：

表名	用途	关键字段
yara_rule	存储 Yara 规则	`id`, `rule_name`, `source_name`, `compiled_rule` (BLOB), `enabled`, `compiled_sha256`
sigma_rule	存储 Sigma 规则	`id`, `title`, `rule_json` (JSON), `enabled`, `sha256`

表名	用途	关键字段
vuln_details	漏洞详细信息	`cve_id`, `cnvd_id`, `description`, `published_date`
product	产品信息	`product_id`, `product_name`, `vendor`
company	厂商信息	`company_id`, `company_name`

5. 接口设计 (API)

所有接口均位于 `/` 根路径下，通过 HTTP POST/GET 请求交互。

5.1 通用接口

- `GET /`: 欢迎信息，用于健康检查。

5.2 Yara 相关

- `POST /uploadYaraRule`: 上传单个 Yara 规则文件。
  - 参数: `file`, `source_name`
- `POST /uploadYaraRuleZip`: 上传 Yara 规则压缩包。
  - 参数: `file`, `source_name`
- `POST /scanSampleWithYara`: 扫描样本文件。
  - 参数: `file` (二进制文件), `label`, `rule_set` (默认为 "enabled")

5.3 Sigma 相关

- `POST /uploadSigmaRuleYaml`: 上传单个 Sigma 规则 (YAML)。
  - 参数: `file`, `source_name`
- `POST /uploadSigmaRuleZip`: 上传 Sigma 规则压缩包。
  - 参数: `file`, `source_name`
- `POST /detectEvtxWithSigma`: 扫描 EVTX 日志。
  - 参数: `file` (.evtx), `label`, `rule_set`, `return_level`

5.4 漏洞数据相关 (部分列举)

- 漏洞数据的增删改查接口分布在各 Service 实现中，通过 `src/apps/routes/main.py` 暴露（具体路由需进一步结合前端调用确认，目前后端代码中主要体现了 Service 实现）。

6. 部署说明（具体请看用户手册）

1. **环境准备**:
   - 安装 Python 3.9+。
   - 安装 MySQL 8.0，并导入 `nvd_database.sql` 初始化数据库。
2. **依赖安装**:

```
pip install -r vuln_backend/src/requirments.txt
```

3. **第三方工具配置**:
   - 确保 `vuln_backend/third_party/Zircolite` 目录下包含 `zircolite.py` 及相关依赖。
4. **配置修改**:
   - 修改 `src/config.py` 中的数据库连接字符串（`SQLALCHEMY_DATABASE_URI`）。
5. **启动服务**:

   ```
   python vuln_backend/src/run.py
   ```

   服务将默认运行在 `http://127.0.0.1:3000`。

一些尚未完善的小问题：进入非主界面后刷新界面会报错需要返回主界面重新进入