TONGJI UNIVERSITY

TONGJI UNIVERSITY

安全软件

设计与开发报告

目录

一、项目概述 3
1.1 背景 3
1.2 核心功能 3
1.3 技术栈 3
1.4 安装部署 3
1.5 分工 ..... 错误!未定义书签。
二、需求分析 5
2.1 功能需求 5
2.2 非功能需求 5
三、系统设计 6
3.1 总体设计 6
3.2 数据包抓取与分析模块 7
3.3 流量特征分析和威胁检测模块 7
3.3.1 从 pcap 文件提取流量特征 7
3.3.2 威胁检测模型训练 9
3.3.3 威胁检测 10
四、 详细设计 12
4.1 主页设计 12
4.2 网络扫描模块 12
4.2.3 内容表示 16
4.2.4 筛选器 16
4.3 威胁检测模块 16
4.3.1 页面设计 16
4.3.2 从 pcap 文件提取流量特征 17
4.3.3 威胁检测模型训练 20
4.2.3 威胁检测 22
五、 测验 24
5.1 数据包抓取模块 24
5.2 威胁检测模块 24
5.2.1 输入文件名方式 24
5.2.2 实时检测方式 25
六、 心得和总结 27
6.1 总结 27
6.2 心得 27
参考文献 28

一、项目概述

1.1 背景

网络入侵检测是网络安全领域的重要技术之一，其目的是检测和识别网络中的恶意行为，如未经授权的访问、恶意软件攻击、拒绝服务攻击等。入侵检测系统（IDS）被称作网络安全的“第二道防线”，在防火墙之后发挥着关键作用。随着网络技术的发展，网络入侵检测技术也在不断演进，主要包括基于特征检测、基于异常检测、基于机器学习和深度学习等多种方法。

1.2 核心功能

实现数据包抓取，协议分析，并保存为 pcap 文件供入侵检测部分提取流量特征，使用训练的随机森林模型进行入侵检测。

1.3 技术栈

前端技术:

1. Bootstrap 5 - 界面框架
2. jQuery - DOM 操作
3. Plotly (v5.3.1) - 数据可视化

后端框架：

Flask (v2.2.5) - 作为核心 Web 框架

打包成 exe：

Pywebview 将前后端架构转化成桌面程序

Pyinstaller 打包

数据处理与分析

1. pandas (v2.2.3) - 数据操作
2. numpy (v2.2.6) - 数值计算
3. Scikit-learn (1.6.1) - 特征预处理（StandardScaler）和降维（PCA）

其他

1. scapy (v2.4.5) - 数据包操作
2. Joblib - 模型持久化
3. sklearn 中的 RandomForestClassifier – 机器学习所用

1.4 安装部署

运行：

下载 npcap(scapy 抓包需要）：https://npcap.com/#download

点击 ids.exe 文件

源码运行/打包过程：

打开源代码文件夹

激活虚拟环境：venv\Scripts\activate（requirement.txt 注明所需库，所用虚拟环境也打包在 venv 文件夹中）

打包成 exe：pyinstaller main.spec
也可直接运行 main.py，将显示同 exe 运行
也可以直接运行 app.py，将使用浏览器运行项目

二、需求分析

2.1 功能需求

完成入侵检测的前提是需要完成网络扫描，进行抓包和解析，在此基础上判断是否有入侵行为，因此对此任务做出以下拆分：

- 数据包抓取：

  监听主机所有端口捕获网络数据包。

- 数据包解析，包括协议分析、ASCII字符表示：

  选择某一数据包进行解包，对数据包中包含的IP，TCP，UDP，ICMP，Ether 协议进行分层解析，且打印元数据包的十六进制和ASCII字符表示。

捕获数据包后，使用机器学习等方法对数据文件的进行入侵检测，对此任务做出以下拆分：

- 提取流量特征：

  对得到的pcap文件进行解析，从中提取流量特征。

- 威胁检测：

  训练模型，并利用模型根据流量特征进行检测。

- 可视化：

  将分析结果在前端用图形界面显示。

2.2 非功能需求

- 易用性

提供直观的图形化界面，支持可视化配置、实时监控和信息实时分析，降低运维复杂度。具备详细的日志记录和数据文件生成功能，帮助快速定位问题并进行系统维护。

- 准确性

系统结合先进的机器学习算法，优化检测模型以减少误报和漏报，确保报警信息的准确性和可信度。

- 可扩展性

系统应采用模块化设计，以适应不断变化的网络规模和安全策略。能够自定义 API 接口，便于集成第三方工具或扩展功能，满足未来优化需求。

三、系统设计

3.1 总体设计

本项目按照 MVC 架构，各层如下：

控制器层：app.py 处理路由，调用服务层处理数据实现各个功能

视图层：templates 文件夹中

服务层：modules 文件夹中存放了两个模块数据包抓取和流量特征提取并调用 model 文件夹中的模型进行威胁检测

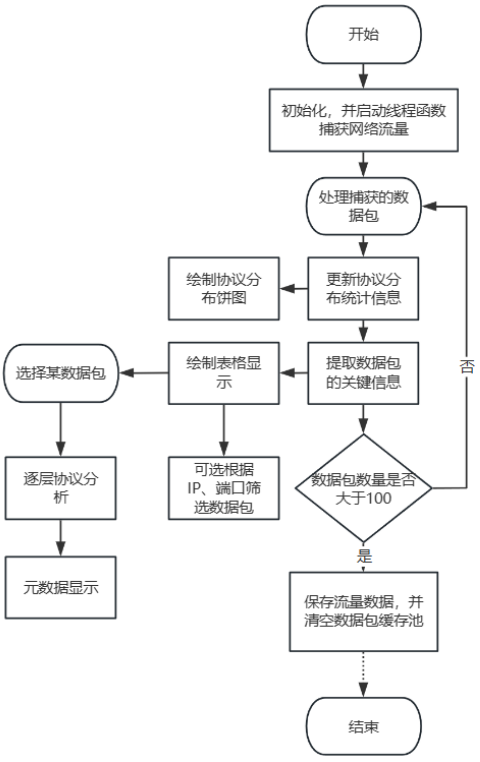
文件结构如下：

```
IDS/
├── app/                        # 主应用模块
│   ├── app.py                  # Flask 应用入口
│   ├── main.py                 # 将前后端架构转为桌面程序
│   ├── model/                  # 机器学习模型
│   │   ├── preprocessing_pipeline.pkl
│   │   ├── randomforest_model.pkl
│   │   └── train/              # 模型训练相关
│   │       ├── main.py         # 训练代码
│   │       ├── merge_data.py
│   │       └── requirements.txt
│   ├── modules/                # 核心功能模块
│   │   ├── pocket_detection/   # 数据包捕获模块
│   │   │   └── detector.py
│   │   └── threat_find/        # 威胁发现模块
│   │       ├── flow_feature/   # 数据包整合为流量，提取流量特征
│   │       │   ├── BasicFlow.py BasicPacketInfo.py
│   │       │   └── FlowFeature.py FlowGenerator.py PacketReader.py utils.py
│   │       └── ThreatFind.py   # 整合流量特征提取和威胁检测
│   ├── templates/              # 前端界面
│   │   ├── index.html          # 主页
│   │   ├── intrusion_detection.html # 威胁检测页面
│   │   ├── logs.html           # 日志页面
│   │   └── network_monitor.html # 数据包捕获页面
│   └── 测试用例/               # 存放一些测试数据
│       └──test.pcap .....
├── data/                       # 运行时存储的数据文件
├── requirements.txt            # Python 依赖库
└── venv/                       # 虚拟环境目录
```

3.2 数据包抓取与分析模块

在此模块，使用 flask+socketIO 进行前后端数据传送，使用 spcap 库进行网络数据包抓取，然后对于抓取的每个数据包一一进行协议分析，根据最高层协议绘制协议分布饼图，同时用表格可视化数据包的相关信息，支持选择某一数据包进行详细协议分析和数据包元数据浏览。

模块工作流程如下图，主要使用核心模块——TrafficDetector 类。在后文详细设计具体介绍。



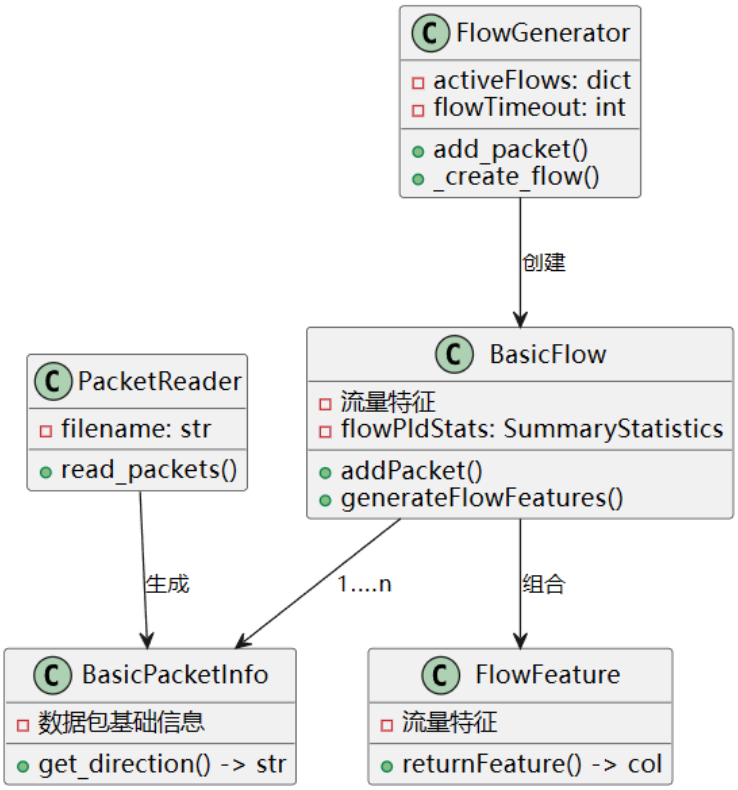
捕获的数据包表格每 10 秒进行一次刷新，每 100 个数据包写入以 hh：mm：ss 格式的时间戳命名的流量数据文件，traffic 文件夹保存.json 格式，内容为数据包关键数据即表格中信息，包括 MAC 地址、IP 地址、端口地址、时间戳、载荷大小以及十六进制元数据；pcap 文件夹保存.pcap 格式，内容为数据包二进制数据。

3.3 流量特征分析和威胁检测模块

在这个模块，首先将一段时间内的数据包组成的 pcap 文件进行解析重组，得到流量特征，同时使用数据集训练分类模型，利用训练得到的模型和提取到的特征完成流量分类，识别可能的威胁。

3.3.1 从 pcap 文件提取流量特征

这个部分涉及以下类，UML 图如下：



主要包含以下核心模块：

1. 数据包解析层 （类 PacketReader）

使用 Scapy 库解析 PCAP 文件，生成标准化的 BasicPacketInfo 对象，记录源/目的 IP、端口、协议类型、时间戳、负载等核心信息。

2. 会话流重组层 （类 FlowGenerator）

根据流超时设置，重组网络会话流，主要流程包括：

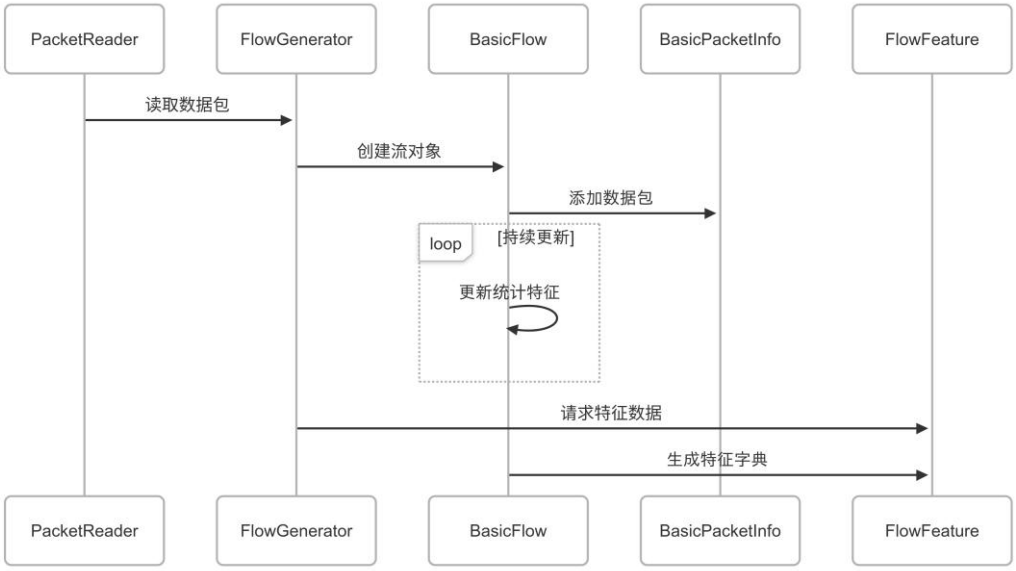
1、通过流 ID 哈希表管理活跃连接

2、处理 TCP 连接状态（RST/FIN 等标志）

3、支持子流分割和 Bulk 传输检测

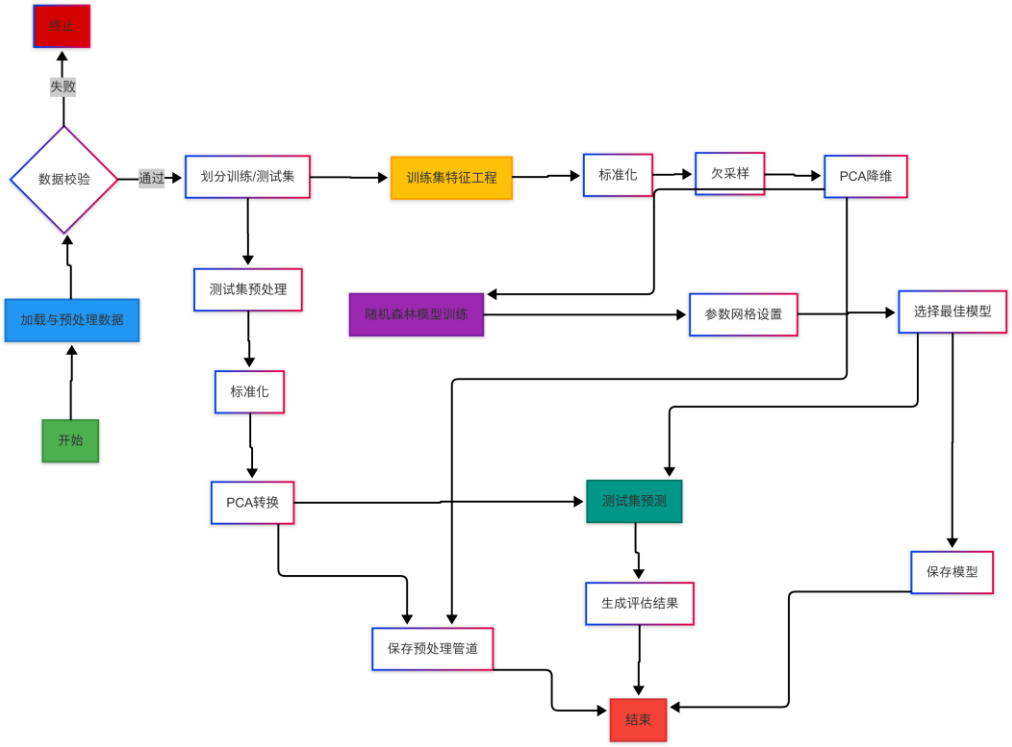
3. 流量特征工程层 （类 BasicFlow.py 和类 FlowFeature）：提取 86 维网络流量特征

整理成时序图，如下：



3.3.2 威胁检测模型训练

模型的目标是通过流量特征进行分类，因此采用随机森林模型，数据集为 CIC-IDS201 (下载链接：https://www.unb.ca/cic/datasets/ids-2017.html)，数据集按天数分成了不同文件，首先将所有数据合并，然后进行训练，训练流程如下：



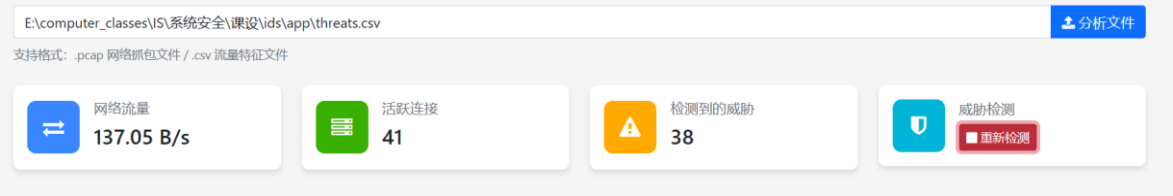
模型训练部分，首先通过元数据剔除、缺失值清洗及标签编码完成数据预处理，采用分层抽样划分训练/测试集（8:2）；针对特征维度高、类别分布失衡问题，设计标准化并定向欠采样，然后进行 PCA 降维（保留 95%方差）的级联特征工程；构建网格搜索随机森林模型，在交叉验

证框架下探索树结构参数与类别权重策略；最终输出包含分类报告（精确率/召回率/F1 值）、序列化模型及预处理管道。通过分离训练/测试集处理流程避免数据泄露，采用 joblib/pickle 双格式保障以便部署使用。

3.3.3 威胁检测

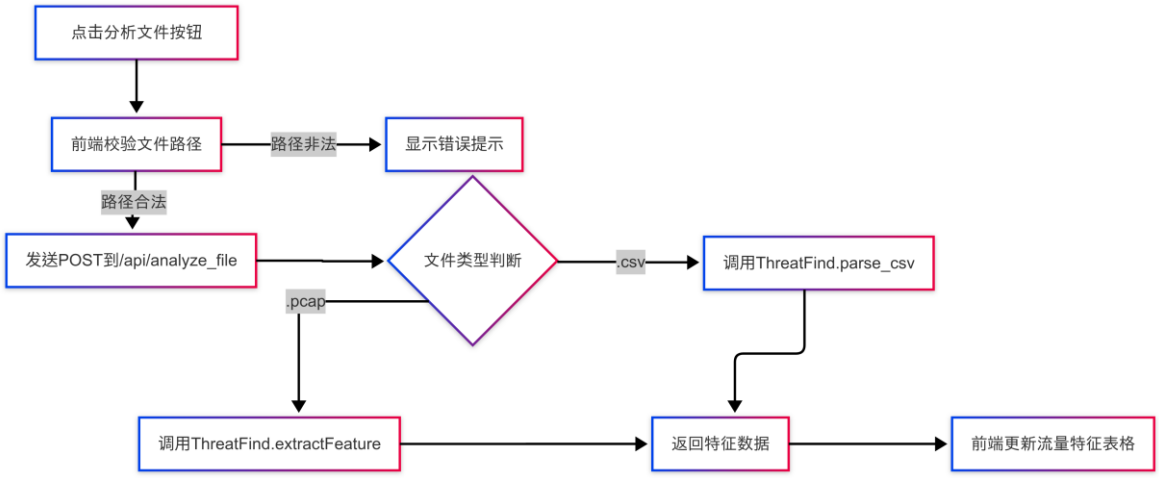
设计类 ThreatFind，整合特征提取和调用模型进行检测的功能。在初始化时，读取保存的模型，设计 extractFeature 方法输入 pcap 文件名和 parse_csv 方法输入 csv 文件，返回流量特征，设计 predictThreat 方法来进行威胁预测，在该方法中，会先对数据进行预处理，然后利用模型进行预测，最终返回威胁分类结果。

前端设计 2 个按钮，分别为为文件分析按钮和威胁检测按钮。



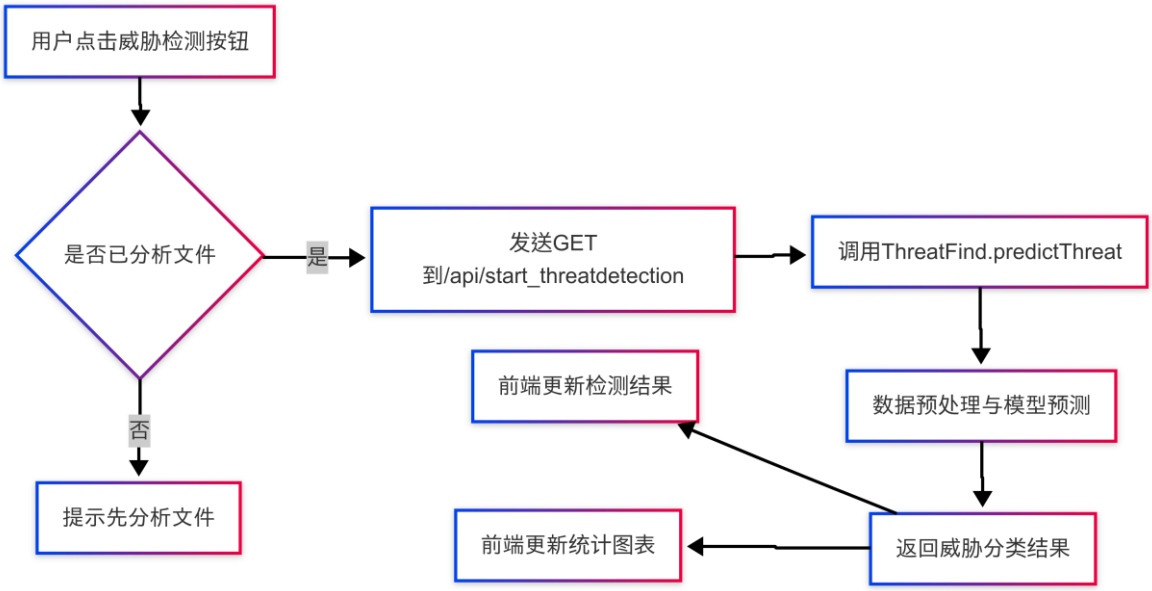
点击分析文件按钮后，前端会先对文件路径进行校验。如果路径合法，就会发送一个 POST 请求到“/api/analyze_file”接口；若路径非法，则会显示错误提示。在接口收到请求后，会进行文件类型判断，如果是.pcap 文件，就调用 ThreatFind 的 extractFeature 方法来提取特征；如果是.csv 文件，就调用 ThreatFind 的 parse_csv 方法来解析数据。无论是哪种文件类型，经过相应处理后都会返回特征数据，最后前端会根据返回的特征数据更新流量特征表格。

流程图如下：



点击威胁检测按钮后，系统会先判断是否已分析文件。如果已分析文件，就发送 GET 请求到“/api/start_threatdetection”接口；若未分析文件，则会提示用户先进行文件分析。接口收到请求后，会调用 ThreatFind 的 predictThreat 方法来进行威胁预测。前端接收到结果后，会更新威胁列表，并且还会根据结果更新攻击类型图表。

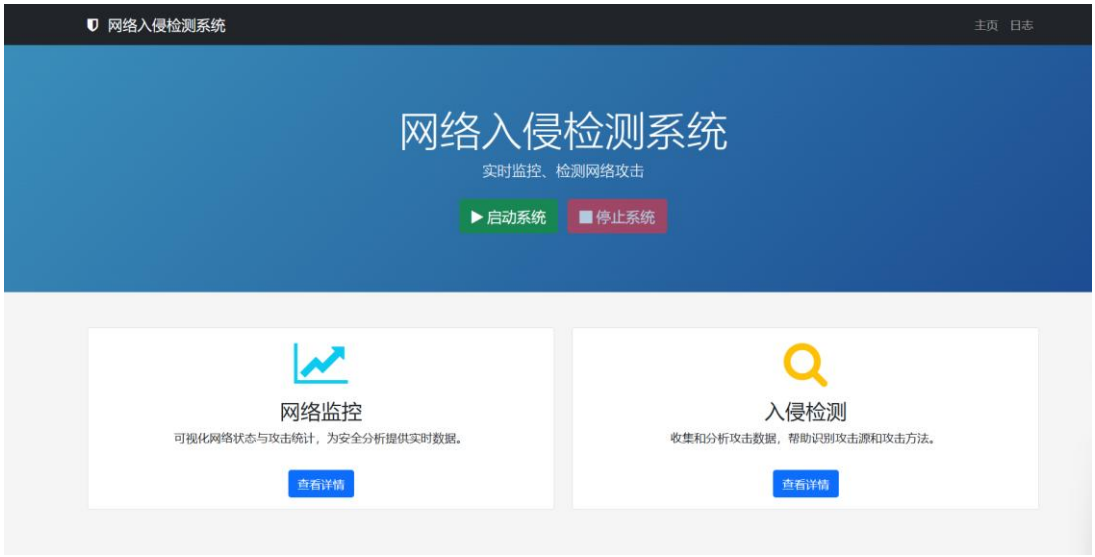
流程图如下：


用户点击威胁检测按钮
是否已分析文件
是
否
发送GET
到/api/start_threatdetection
调用ThreatFind.predictThreat
数据预处理与模型预测
返回威胁分类结果
前端更新检测结果
前端更新统计图表
提示先分析文件

四、详细设计

4.1 主页设计

在首页点击启动系统，下方显示系统运行中即可。

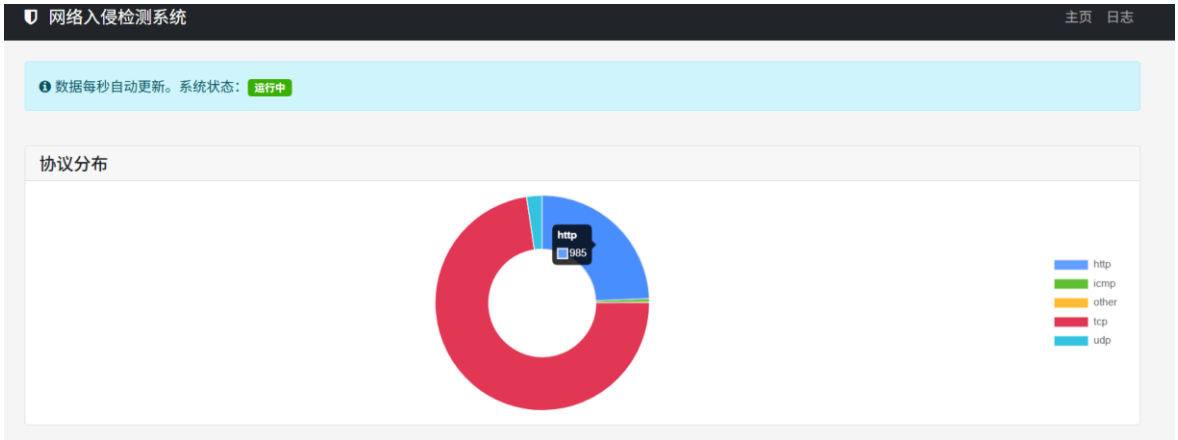


4.2 网络扫描模块

点击网络监控的查看详情按钮进入网络监控页面。

4.2.1 协议分布

协议分布饼图如下，鼠标移动可以看见每种协议的数据包具体数量。



在 network_monitor.html 文件中实现相关代码，相关函数有 initCharts()和 updateProtocolChart()，设置 Socket 事件实现实时数据更新。

```
// 初始化图表
function initCharts() {
    // 协议分布图表
    const protocolCtx = document.getElementById('protocolChart').getContext('2d');
    protocolChart = new Chart(protocolCtx, {
```

```
// 更新协议分布图表
function updateProtocolChart(data) {
    if (!protocolChart) return;
    // 如果有数据则使用数据，否则模拟数据
    let protocolLabels = [];
    let protocolCounts = [];
    if (data) {
       protocolLabels = Object.keys(data);
       protocolCounts = Object.values(data);
    }
    // 更新图表数据
    protocolChart.data.labels = protocolLabels;
    protocolChart.data.datasets[0].data = protocolCounts;
    protocolChart.update();
}
```

4.2.2 数据包捕获

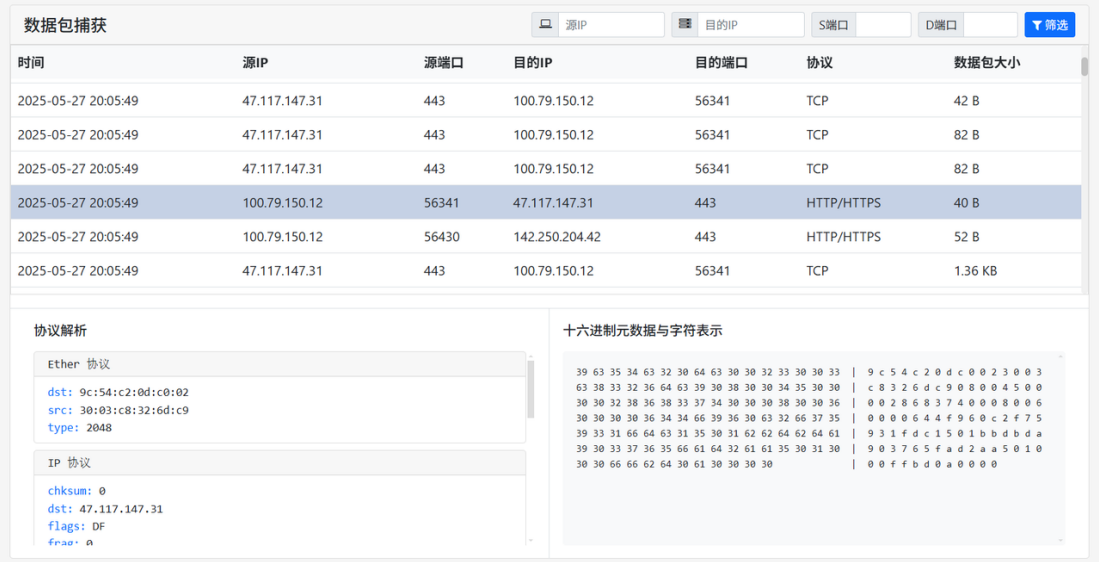
数据包捕获表格如下图，选择表格任意行可进行数据包分析，被选数据包高亮表示。通过点击事件将被选的数据包信息传输给后端 。

```
    // 添加点击事件
    row.click(function () {
        const packetData = $(this).data('packet');
        showPacketDetails(packetData);

        // 高亮当前选中行
        $('#packetTable tr').removeClass('table-active');
        $(this).addClass('table-active');
    });
```

左下角协议分析部分对于数据包中使用的 ether、IP、HTTP、TCP、UCP、ICMP 协议进行分析，调用库函数抓取协议各部分内容。

右下角元数据用十六进制和 ASCII 字符分别表示，左右一一对应。



下面通过介绍 TrafficDetector 类来讲述该部分的详细设计。

• 数据包捕获和关键信息提取核心函数是_extract_packet_info(self, packet)，使用 spacp 库函数进行协议判断和统计，返回即为数据包关键信息，此信息构成数据包表格。

```
def _extract_packet_info(self, packet):
        """提取数据包的关键信息"""
        packet_type = 'other'
        src_ip = dst_ip = 'unknown'
        src_port = dst_port = 0
        protocol = 'unknown'
        payload_size = 0
        src_mac = dst_mac = 'unknown'
        try:
            # 提取 MAC 地址
            if Ether in packet:
                src_mac= packet[Ether].src
                dst_mac = packet[Ether].dst
            # 检查是否包含 IP 层
            if IP in packet:
                src_ip = packet[IP].src
                dst_ip = packet[IP].dst
                # TCP 数据包
                if TCP in packet:
                    packet_type = 'tcp'
                    self.packet_stats['tcp'] += 1
                    src_port = packet[TCP].sport
                    dst_port = packet[TCP].dport
                    protocol = 'TCP'
                    # 检查是否是 HTTP
                    if packet.haslayer(HTTP) or dst_port == 80 or dst_port == 443:
                        packet_type = 'http'
                        self.packet_stats['http'] += 1
                        protocol = 'HTTP/HTTPS'
                # UDP 数据包
                elif UDP in packet:
                    packet_type = 'udp'
                    self.packet_stats['udp'] += 1
                    src_port = packet[UDP].sport
                    dst_port = packet[UDP].dport
                    protocol = 'UDP'
                # ICMP 数据包
                elif ICMP in packet:
                    packet_type = 'icmp'
                    self.packet_stats['icmp'] += 1
                    protocol = 'ICMP'
                # 计算载荷大小
                if hasattr(packet, 'payload'):
                    payload_size = len(packet.payload)
                # 返回数据包信息
```

```
                return {
                    'timestamp': datetime.now().isoformat(),
                    'src_ip': src_ip,'dst_ip': dst_ip,
                    'src_mac': src_mac,'dst_mac': dst_mac,
                    'src_port': src_port,'dst_port': dst_port,
                    'protocol': protocol,'size': payload_size,
                    'type': packet_type,'raw': packet.copy().build().hex()
                }
        except Exception as e:
            logger.error(f"数据包处理错误: {str(e)}")
        return None
```

• 协议分析使用函数 analyze_packet(self, raw_hex)，使用十六进制数据复现元数据包，遍历协议层一一分析

```
def analyze_packet(self, raw_hex):
    """分析数据包的所有协议层（类似Wireshark的协议分层结构）"""
    try:
        raw_bytes = bytes.fromhex(raw_hex)
        packet = Ether(raw_bytes)
        protocols = []

        # 遍历所有协议层
        for layer in packet.layers():
```

• 在 process_packet 函数中设置定期保存，_save_traffic_data()和_save_recent_pcap()分别负责保存.json 文件和.pcap 文件，后者支持分块储存，固定.pcap 文件包含 100 个数据包。

```
def process_packet(self, packet):
        # 提取和分析数据包
        packet_info = self._extract_packet_info(packet)
        # 保存流量数据
        if packet_info:
            self.traffic_data.append(packet_info)
            # 定期保存流量数据，防止内存占用过多
            if len(self.traffic_data) >= 100:
                self._save_traffic_data()
                self._save_recent_pcap()
                self.traffic_data = []
```

```
def _save_traffic_data(self):
    """保存流量数据到文件"""
    try:
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        os.makedirs('data/traffic', exist_ok=True)
        filename = f'data/traffic/traffic_{timestamp}.json'
        with open(filename, 'w') as f:
            json.dump(self.traffic_data, f)
        logger.info(f"已保存流量数据到 {filename}，共 {len(self.traffic_data)} 条记录")
```

```python
def _save_recent_pcap(self, total=100, chunk_size=100):
    """将前total个数据包按chunk_size分块保存为多个PCAP文件
    Args:
        total (int): 总数据包数量
        chunk_size (int): 每个PCAP文件包含的数据包数
    """
    try:
        # 获取数据包记录（线程安全）
        with threading.Lock():
            target_data = self.traffic_data[:total].copy()

        os.makedirs('data/pcap', exist_ok=True)
```

4.2.3 内容表示

由 network_monitor.html 文件中 hexDump 函数实现显示。

```javascript
// 十六进制转储函数
function hexDump(data) {
    if (!data) return "无数据";
    const bytes = new TextEncoder().encode(data);
    // 每行显示16字节（32个十六进制字符 + 15个空格）
    const bytesPerLine = 16;
    let hexOutput = '';
    let asciiOutput = '';
    bytes.forEach((byte, index) => {
        // 每个字节转为两位十六进制+空格
        hexOutput += byte.toString(16).padStart(2, '0') + ' ';
        // ASCII字符显示，不可打印字符用'.'代替
        asciiOutput += (byte >= 32 && byte <= 126) ? String.fromCharCode(byte) : '.';
        asciiOutput +=  ' ';
```

4.2.4 筛选器

可以根据源 IP、目的 IP、源端口和目的端口筛选数据包。

时间	源IP	源IP归属地	源端口	目的IP	目的IP归属地	目的端口	协议	数据包大小
2025-05-26 22:07:14	100.80.129.213	未知	60773	121.51.49.150	未知	80	HTTP/HTTPS	40 B
2025-05-26 22:07:14	100.80.129.213	未知	60773	121.51.49.150	未知	80	HTTP/HTTPS	40 B
2025-05-26 22:07:14	100.80.129.213	未知	60773	121.51.49.150	未知	80	HTTP/HTTPS	40 B
2025-05-26 22:07:14	100.80.129.213	未知	60773	121.51.49.150	未知	80	HTTP/HTTPS	40 B
2025-05-26 22:07:14	100.80.129.213	未知	60773	121.51.49.150	未知	80	HTTP/HTTPS	40 B

4.3 威胁检测模块

4.3.1 页面设计

点击主页的入侵检测，进入入侵检测页面,入侵检测页面设计如下：

4.3.2 从 pcap 文件提取流量特征

在上一部分，给出了流量特征提取需要的各个类的关系，最后可以得到以下流量特征，在 ThreatFind 类中方法 extractFeature 利用流量特征提取相关类完成流量特征提取，代码如下：

```
        def extractFeature(self,pcap_file=None):
            if pcap_file != None :
                self.file_path=pcap_file
            packetReader = PacketReader(self.file_path)
            flowGenerator = FlowGenerator(self.flowTimeout, self.activityTimeout,
self.subFlowTimeout, self.bulkTimeout)
            basicPacket = packetReader.nextPacket()
            if basicPacket == None:
                print("None")
            while basicPacket != None:
                flowGenerator.addPacket(basicPacket)
                basicPacket = packetReader.nextPacket()
            flowGenerator.clearFlow()
            flowGenerator.dumpFeatureToCSV("data/result.csv") # 保存到csv中
            self.features_dict=flowGenerator.get_features_dict()
```

最后可以得到流量特征如下：

类别	属性名称	描述
流基本信息	flowId	流 ID
	srcIP	源 IP 地址
	dstIP	目的 IP 地址
	srcPort	源端口

	dstPort	目的端口
	protocol	传输层协议
数据包个数与负载字节数相关特征	flowPktNum	流数据包数量
	flowPldByteSum	流负载字节数总和
	flowPldByteMax	流负载字节数最大值
	flowPldByteMin	流负载字节数最小值
	flowPldByteMean	流负载字节数平均值
	flowPldByteStd	流负载字节数标准差
	fwdPktNum	正向数据包数量
	fwdPldByteSum	正向数据包负载字节数总和
	fwdPldByteMax	正向数据包负载字节数最大值
	fwdPldByteMin	正向数据包负载字节数最小值
	fwdPldByteMean	正向数据包负载字节数平均值
	fwdPldByteStd	正向数据包负载字节数标准差
	....	反向同上
数据包头字节数相关特征	fwdHeadByteMax	正向数据包头字节数最大值
	fwdHeadByteMin	正向数据包头字节数最小值
	fwdHeadByteMean	正向数据包头字节数平均值
	fwdHeadByteStd	正向数据包头字节数标准差
	...	反向同上
流速相关特征	flowDurationMS	流持续时间（ms）
	flowPktsS	每秒传输的数据包数
	flowPldBytesMS	每毫秒传输的数据包负载字节数
	fwdPktsS	每秒正向传输的数据包数
	fwdPldBytesMS	每毫秒正向传输的数据包负载字节数
	bwdPktsS	每秒反向传输的数据包数
	bwdPldBytesMS	每毫秒反向传输的数据包负载字节数
	pktsRatio	反向/正向传输的数据包数比例

	bytesRatio	反向/正向传输的数据包负载字节数比例
间隔时间相关特征	flowIatMax	数据包间隔时间最大值
	flowIatMin	数据包间隔时间最小值
	flowIatMean	数据包间隔时间平均值
	flowIatStd	数据包间隔时间标准差
	....	正向/反向数据包同上
TCP 标志相关特征	FINcnt	带有 FIN 的数据包数量
	SYNcnt	带有 SYN 的数据包数量
	RSTcnt	带有 RST 的数据包数量
	PSHcnt	带有 PSH 的数据包数量
	ACKcnt	带有 ACK 的数据包数量
	URGcnt	带有 URG 的数据包数量
	ECEcnt	带有 ECE 的数据包数量
	CWRcnt	带有 CWR 的数据包数量
	fwdPSHcnt	正向数据包中设置 PSH 标志的数量（UDP 为 0）
	bwdPSHcnt	反向数据包中设置 PSH 标志的数量（UDP 为 0）
	fwdURGcnt	正向数据包中设置 URG 标志的数量（UDP 为 0）
	bwdURGcnt	反向数据包中设置 URG 标志的数量（UDP 为 0）
初始窗口大小	fwdInitWinBytes	正向的初始 TCP 窗口大小（UDP 为 0）
	bwdInitWinBytes	反向的初始 TCP 窗口大小（UDP 为 0）
有效负载数据包个数	fwdPktsWithPayload	具有有效负载的正向数据包个数
	bwdPktsWithPayload	具有有效负载的反向数据包个数
子流相关特征	subFlowFwdPkts	正向子流中数据包的平均数量
	subFlowFwdPldBytes	正向子流中字节的平均数量
	subFlowBwdPkts	反向子流中数据包的平均数量
	subFlowBwdPldBytes	反向子流中字节的平均数量
流活动-空闲相关特征	flowActSum	流在空闲之前处于活动状态的时间总和
	flowActMax	流在空闲之前处于活动状态的时间最大值

flowActMin	流在空闲之前处于活动状态的时间最小值
flowActMean	流在空闲之前处于活动状态的时间平均值
flowActStd	流在空闲之前处于活动状态的时间标准差
flowIdleSum	流在激活之前处于空闲状态的时间总和
flowIdleMax	流在激活之前处于空闲状态的时间最大值
flowIdleMin	流在激活之前处于空闲状态的时间最小值

4.3.3 威胁检测模型训练

上一部分给出训练流程，这里将详细介绍各个模块：

1、数据加载与预处理模块：

该模块主要负责加载数据并进行一系列预处理操作。首先通过 pandas 库的 read_csv 函数加载指定路径的 csv 文件，并对列名进行清理（去除首尾空白字符）。然后删除一些不必要的列。

为了保证数据的随机性，使用 sample 函数对数据进行打乱，并重置索引。对于标签列“Label”，使用 LabelEncoder 进行编码，将类别标签转换为数值形式，并将编码器保存在字典 le_dict 中，方便后续对测试数据进行相同的编码操作。

代码如下：

```
    def load_and_preprocess(filepath):
        data = pd.read_csv(filepath)
        data.columns = data.columns.str.strip()
        print("列名：", data.columns.tolist())
        data = data.drop(['Flow ID', 'Source IP', 'Destination IP',
'Timestamp','Fwd Header Length.1'], axis=1, errors='ignore')
        data = data.dropna()
    # 打乱顺序
        data = data.sample(frac=1, random_state=42).reset_index(drop=True)
le_dict = {}
        for col in ['Label']:
            le = LabelEncoder()
            data[col] = le.fit_transform(data[col])
            le_dict[col] = le
        X = data.drop('Label', axis=1)
        y = data['Label']
        print("各类别样本数量：", Counter(y))
        return X, y, le_dict
```

2、特征工程模块

特征工程模块主要包括特征标准化、欠采样和 PCA 降维操作。首先，利用

StandardScaler 对特征进行标准化处理，使每个特征的均值为 0，标准差为 1，从而消除不同特征之间的量纲差异。

然后，针对类别分布不平衡的问题，采用 RandomUnderSampler 进行欠采样，设置类别 0 保留 50000 样本，其他类别保持不变，以平衡不同类别之间的样本数量，提高模型对 minority 类别的学习能力。

最后，使用 PCA（主成分分析）进行降维，保留 95% 的方差信息，将高维特征投影到低维空间，减少特征维度，降低计算复杂度，同时去除冗余信息和噪声干扰。

代码如下：

```
def feature_engineering(X, y):
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    rus = RandomUnderSampler(sampling_strategy={0: 50000}, random_state=42)
    X_under, y_under = rus.fit_resample(X_scaled, y)
    print("欠采样后类别分布：", Counter(y_under))
    pca = PCA(n_components=0.95)
    X_pca = pca.fit_transform(X_under)
    return X_pca.astype(np.float32), y_under, scaler, pca
```

3、模型构建模块

构建了一个基于随机森林的分类模型，并使用 GridSearchCV 进行超参数优化。定义了参数网格 param_grid，包括随机森林的决策树数量 n_estimators、最大树深 max_depth、最小样本分裂 min_samples_split 以及类别权重 class_weight 等参数的取值范围。

GridSearchCV 通过交叉验证的方式对参数网格进行搜索，寻找使 f1_macro 评价指标最大的最佳参数组合，从而构建出性能较优的随机森林分类模型，提高模型的泛化能力和分类精度。

代码如下：

```
def build_model():
    param_grid = {
        'n_estimators': [100, 200],
        'max_depth': [None, 10],
        'min_samples_split': [2, 5],
        'class_weight': ['balanced', None]
    }
    model = GridSearchCV(
        estimator=RandomForestClassifier(random_state=42),
        param_grid=param_grid,
        cv=3,
        n_jobs=-1,
        scoring='f1_macro'
    )
```

```
        return model
```

4、训练结果

如下图所示，其中 0-11 标签为：'正常流量', 'Bot', 'DDoS', 'DoS GoldenEye', 'DoS Hulk', 'DoS Slowhttptest', 'DoS slowloris', 'FTP-Patator', 'Heartbleed', 'Infiltration', 'PortScan', 'SSH-Patator'

```
最佳参数: {'class_weight': 'balanced', 'max_depth': None, 'min_samples_split': 5, 'n_estimators': 200}
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    336542
           1       0.68      0.97      0.80       313
           2       0.99      1.00      1.00     20448
           3       0.94      1.00      0.97      1648
           4       0.94      1.00      0.97     36825
           5       0.93      0.99      0.96       880
           6       0.99      0.99      0.99       927
           7       0.99      1.00      0.99      1264
           8       1.00      1.00      1.00         2
           9       0.50      0.17      0.25         6
          10       0.84      1.00      0.91     25430
          11       0.95      0.99      0.97       939

    accuracy                           0.98    425224
   macro avg       0.90      0.92      0.90    425224
weighted avg       0.98      0.98      0.98    425224
```

4.2.3 威胁检测

首先需要将提取到的流量特征与数据集的特征对齐，使用 dataframe 处理，相关代码如下：

```
    # 检查是否有重复列名
            duplicated_cols = df.columns[df.columns.duplicated()]
            if not duplicated_cols.empty:
                print(f"警告：发现重复列名 {duplicated_cols.tolist()}，已自动去重")

            # 确保列名唯一
            df = df.loc[:, ~df.columns.duplicated(keep='first')]  # 保留第一个出现的列
            selected_columns = get_train_header()
            selected_columns.remove("Label")
            # 检查是否存在缺失列
            missing_columns = [col for col in selected_columns if col not in
df.columns]
            if missing_columns:
                print(f"警告：以下列仍然缺失：{missing_columns}")
            # 筛选 DataFrame，只保留指定列
            df_selected = df[selected_columns]
```

保证 threatFind 类中的 feature_dict 无误后，加载模型进行预测，相关代码如下：

```
    def predictThreat(self):
            if not self.features_dict:
                raise ValueError("features_dict 为空，请先 extractFeature()")
```

```
            # Step 1: 构建 DataFrame
            df = pd.DataFrame(self.features_dict)
            # Step 2: 去除无意义列和 ID 列
            id_cols = ["Flow ID", "Source IP", "Destination IP","Timestamp"]
            id_info = df[id_cols].copy()
            df = df.drop(columns=id_cols, errors='ignore')
            # Step 3: 处理无穷值/空值
            df.replace([np.inf, -np.inf], np.nan, inplace=True)
            df.dropna(inplace=True)
            if df.empty:
                print("无有效数据")
                return []
            # Step 4: 标准化 + PCA
            X_scaled = self.scaler.transform(df)
            X_pca = self.pca.transform(X_scaled)
            # Step 5: 预测
            y_pred = self.model.predict(X_pca)
            # Step 6: 转换为原始标签
            labels = self.label_encoder.inverse_transform(y_pred)
            # Step 7: 构建结果
            results = []
            for i in range(len(labels)):
                threat_type = labels[i]
                results.append({
                    "threat_type": self.threat_mapping.get(threat_type, threat_type),
                    "src_ip": id_info.iloc[i]["Source IP"] if "Source IP" in
id_info.columns else "unknown",
                    "dst_ip": id_info.iloc[i]["Destination IP"] if "Destination IP" in
id_info.columns else "unknown",
                    "severity": self.severity_mapping.get(threat_type, '未知'),
                    "original_type": threat_type  # 保留原始类型名称
                })
            return results
```

五、测验

5.1 数据包抓取模块

观察运行时的前端界面，能够实时显示协议分布图和数据包表格，正确进行协议分析和元数据显示。

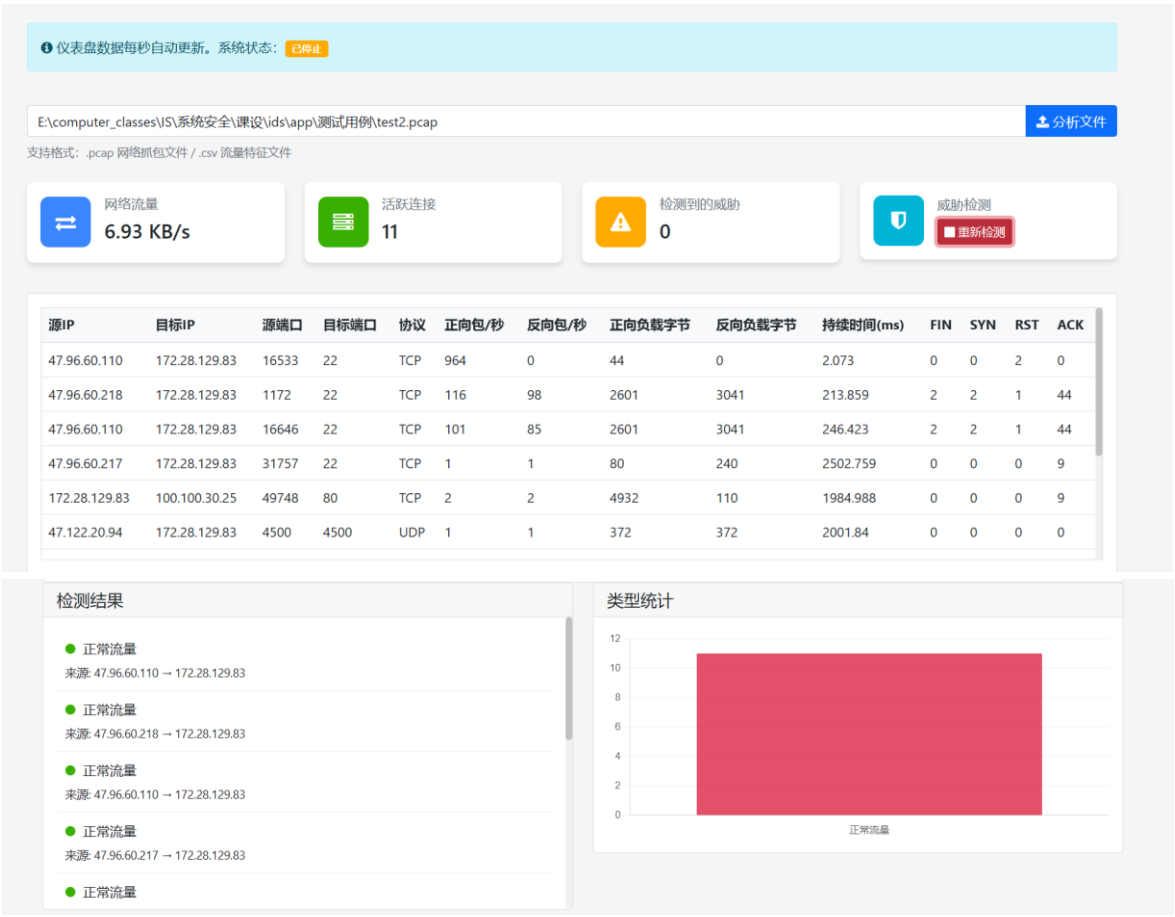
观察日志信息，可见数据文件被成功保存

```
2025-05-27 20:20:02,470 [INFO] 客户端已连接: H1Jj2ajjL4d6CUQFAAAF
127.0.0.1 - - [27/May/2025 20:20:02] "POST /socket.io/?EIO=4&transport=polling&t=PSHhtiS&sid=TOM88Pb6y8C6y8JsAAAE HTTP/1.1" 200 219 0.00137
127.0.0.1 - - [27/May/2025 20:20:02] "GET /socket.io/?EIO=4&transport=polling&t=PSHhtiU&sid=TOM88Pb6y8C6y8JsAAAE HTTP/1.1" 200 181 0.000000
127.0.0.1 - - [27/May/2025 20:20:02] "GET /socket.io/?EIO=4&transport=polling&t=PSHhtjO&sid=TOM88Pb6y8C6y8JsAAAE HTTP/1.1" 200 181 0.000000
127.0.0.1 - - [27/May/2025 20:20:02] "GET /api/status HTTP/1.1" 200 269 0.000000
127.0.0.1 - - [27/May/2025 20:20:05] "POST /api/detect_protocols HTTP/1.1" 200 1110 0.000994
2025-05-27 20:20:06,590 [INFO] 已保存流量数据到 data/traffic/traffic_20250527_202006.json，共 100 条记录
2025-05-27 20:20:06,635 [INFO] 成功保存 data/pcap/20250527_202006.pcap (100 个数据包)
2025-05-27 20:20:11,544 [INFO] 已保存流量数据到 data/traffic/traffic_20250527_202011.json，共 100 条记录
2025-05-27 20:20:11,611 [INFO] 成功保存 data/pcap/20250527_202011.pcap (100 个数据包)
```

5.2 威胁检测模块

5.2.1 输入文件名方式

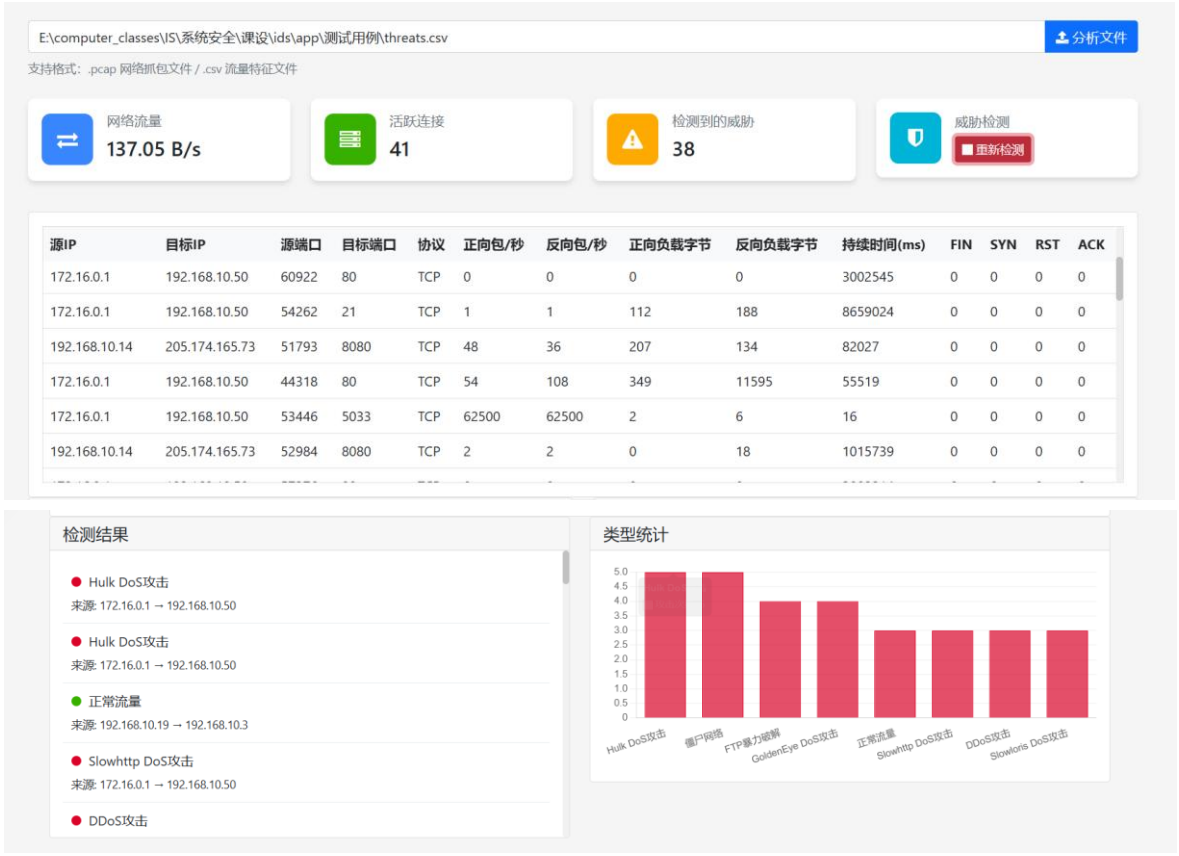
输入 pcap 文件名检测结果：采用的测试文件为用 wireshark 抓取的 pcap



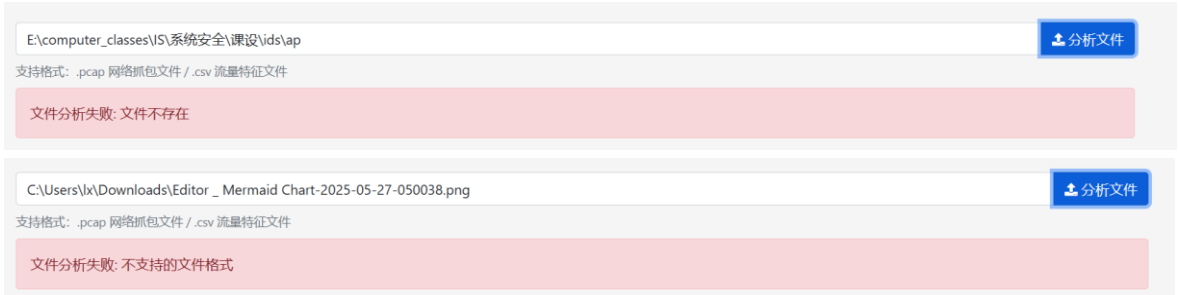
源IP	目标IP	源端口	目标端口	协议	正向包/秒	反向包/秒	正向负载字节	反向负载字节	持续时间(ms)	FIN	SYN	RST	ACK
47.96.60.110	172.28.129.83	16533	22	TCP	964	0	44	0	2.073	0	0	2	0
47.96.60.218	172.28.129.83	1172	22	TCP	116	98	2601	3041	213.859	2	2	1	44
47.96.60.110	172.28.129.83	16646	22	TCP	101	85	2601	3041	246.423	2	2	1	44
47.96.60.217	172.28.129.83	31757	22	TCP	1	1	80	240	2502.759	0	0	0	9
172.28.129.83	100.100.30.25	49748	80	TCP	2	2	4932	110	1984.988	0	0	0	9
47.122.20.94	172.28.129.83	4500	4500	UDP	1	1	372	372	2001.84	0	0	0	0

同时会存储流量提取结果，与 pcap 文件同名：

输入 csv 文件，采用的测试文件为划分数据集的测试集的一部分,结果如下：



文件上传部分会对文件路径是否合法进行判断，输出错误提示如下：



5.2.2 实时检测方式

实时检测为进行一段时间的扫描后，点击停止扫描后，得到 pcap 包存储在固定路径，将会自动读取这些文件进行检测，同时存储得到的流量特征 csv 文件：


仪表盘数据每秒自动更新。系统状态：已停止
输入PCAP文件或CSV文件的绝对路径
分析文件
支持格式：.pcap 网络抓包文件 / .csv 流量特征文件
网络流量
7.07 MB/s
活跃连接
76
检测到的威胁
0
威胁检测
重新检测
源IP 目标IP 源端口 目标端口 协议 正向包/秒 反向包/秒 正向负载字节 反向负载字节 持续时间(ms) FIN SYN RST ACK
18.183.25.2 100.80.79.79 28785 61340 TCP 216 288 153 24 13.857 3 0 1 6


新加卷 (E:) › computer_classes › IS › 系统安全 › 课设 › ids › app › data › flow_features
排序
查看
名称 修改日期 类型 大小
20250527_233224.csv 2025/5/27 23:32 XLS 工作表 11 KB
20250527_233227.csv 2025/5/27 23:32 XLS 工作表 12 KB
20250527_233231.csv 2025/5/27 23:32 XLS 工作表 6 KB

六、心得和总结

6.1 总结

本次所设计的软件完成了网络扫描、抓包和解析，提取流量特征，判断入侵行为。任务拆分为：数据包抓取（监听主机端口捕获数据包）、数据包解析（对协议分层解析并打印十六进制及 ASCII 字符表示）、提取流量特征（解析 pcap 文件）、威胁检测（训练模型并检测流量特征）和可视化（将结果以图形界面展示）。

6.2 心得

在确立入侵检测选题后进行了广泛的调研，在查阅到一篇关于机器学习和深度学习在入侵检测的综述文献（参考文献 5）后，确定了采用机器学习模型进行入侵检测的思路，并采用文献中提到的数据集进行训练，数据集中提供的流量的 83 维特征，且数量庞大（总计 1.13GB），直接训练不仅会导致电脑内存崩溃，还因为其中非入侵正常类别过多，导致采样不均使训练效果较差，最终采用随机扔掉部分正常流量，一次模型训练仍需耗费较长时间。

得到训练好的模型后，要想进行检测，必须对扫描到的数据包进行流量特征提取，且提取到的特征需要对齐用于训练的特征。在广泛调研后，了解到 python 库 CICFlowMeter 可以完成流量特征提取，但写测试代码对这个库进行实验发现，他需要的 tcpdump 版本无法在 windows 上运行，以致这个库无法正常使用，但好在代码开源，通过阅读源码，进行部分修改，重写了一份可用的流量特征提取相关类。

本次软件仍有可用改进的地方，前端数据显示没有与抓包十分同步，威胁检测需在停止抓包得到 pcap 文件后才能进行，没有做到实时威胁检测（做到这一步，还需考虑内存管理，线程同步等更复杂的问题）

参考文献

[1] 蹇诗婕,卢志刚,杜丹,姜波,刘宝旭.网络入侵检测技术综述[J].信息安全学报,2020,5(4):96-122

[2] https://www.snort.org

[3] https://github.com/hustakin/jpcap-mitm

[4] Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP). Portugal.

[5] G. Kocher and G. Kumar, "Machine Learning and Deep Learning Methods for Intrusion Detection Systems: Recent Developments and Challenges," in Soft Computing, 2021.

[6] https://github.com/datthinh1801/cicflowmeter