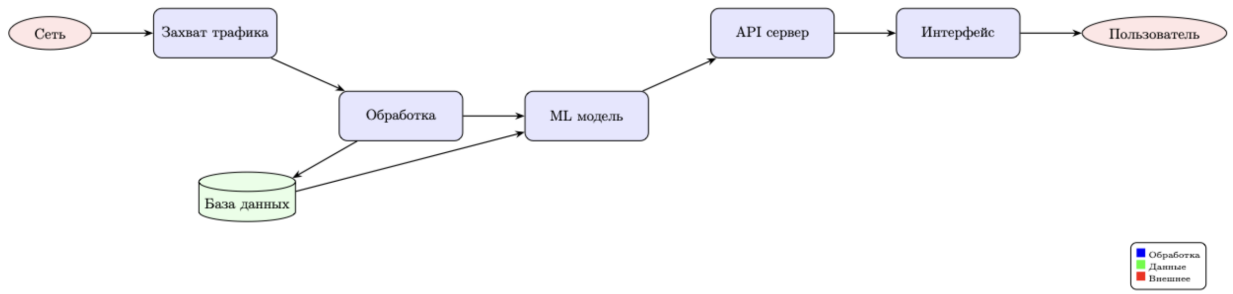
Руководство программиста

1 Архитектура системы

Система состоит из нескольких логически связанных компонентов, взаимодействующих между собой в реальном времени. Общая архитектура представлена на рисунке ниже:



- **Сеть** — источник трафика, поступающего в систему.
- **Сниффер (TrafficSniffer)** — захватывает пакеты через `tcpdump/pyshark`.
- **Обработка и нормализация** — извлечение признаков и приведение к виду, пригодному для ML-модели и хранения в базе данных.
- **База данных (SQLite)** — временное хранилище трафика.
- **Модель машинного обучения** — классифицирует трафик как нормальный или вредоносный.
- **API-сервер (Flask)** — предоставляет интерфейсы взаимодействия с фронтендом и моделью.
- **Веб-интерфейс** — отображает трафик, статистику, результаты анализа.

2 Работа с базой данных

Хранение сетевого трафика реализовано с помощью SQLite. Структура таблицы `traffic`:

```
CREATE TABLE traffic (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  timestamp TEXT,
  src_ip TEXT,
  dst_ip TEXT,
  protocol TEXT,
  bytes INTEGER,
  service TEXT,
  flag TEXT,
  count INTEGER,
  srv_count INTEGER,
  dst_host_count INTEGER,
  dst_host_srv_count INTEGER
)
```

Каждая запись представляет собой обработанный сетевой пакет с извлечёнными признаками. Эти данные позже используются для анализа модели.

3 Подготовка данных для модели

Перед подачей в ML-модель трафик нормализуется. Используются следующие признаки:

- `protocol_type` — нормализованный протокол (tcp, udp, icmp)
- `service` — по определенному порту определяется тип сервиса (http, ssh и др.)
- `flag` — TCP-флаги (SF, S0 и др.)
- `src_bytes` — размер переданных данных от источника
- `dst_bytes` — размер переданных данных к получателю
- `count`, `srv_count`, `dst_host_count`, `dst_host_srv_count` — агрегированные признаки по IP и сервисам

4 Модуль сниффинга трафика (traffic_sniffer.py)

Класс `TrafficSniffer` реализует захват трафика с использованием библиотеки `pyshark` и обёртки над `tcpdump`.

Основные методы:

- `__init__(interface, db_path)` — инициализация с указанием сетевого интерфейса и пути к базе данных
- `start_sniffing()` — запуск перехвата трафика

- `stop_sniffing()` — остановка снифинга
- `packet_handler(packet)` — обработка каждого сетевого пакета (извлечение признаков, нормализация, запись в БД)
- `normalize_protocol()`, `normalize_service()`, `normalize_flag()` — приведение полей пакета к стандартизованному виду

5 Серверное приложение (app.py)

`Flask`-приложение управляет логикой взаимодействия с пользователем:

5.1 Маршруты:

- `/` — главная страница
- `/sniffing` — мониторинг трафика (управление захватом)
- `/cyberattack` — страница анализа с использованием ML
- `/analytics` — статистика по протоколам и сервисам
- `/instruction` — руководство пользователя

5.2 API-эндпоинты:

- `/data` — GET-запрос, возвращает последние 10 пакетов из БД
- `/predict` — POST-запрос, анализирует конкретный трафик через ML-модель
- `/start`, `/stop` — POST-запросы для управления захватом трафика
- `/traffic/analytics` — статистика по протоколам и сервисам

6 Модель машинного обучения

Модель обучается заранее на размеченных данных и сохраняется в виде `attack_model.pkl`. Для корректной работы также сохраняются энкодеры `label_encoders.pkl`, с помощью которых категориальные признаки преобразуются в числовые.

6.1 Используемые признаки:

см. раздел 3.

6.2 Формат вывода:

бинарная метка (атака / норма), вероятность класса.

7 HTML-шаблоны

- `home.html` — структура навигации по приложению
- `index.html` — панель управления мониторингом трафика (старт/стоп)
- `cyberattack.html` — ручной запуск анализа трафика на наличие атак
- `analytics.html` — визуализация статистики по трафику
- `instruction.html` — страница руководства пользователя

8 Развёртывание и использование

1. Установка зависимостей:

   ```
   pip install -r requirements.txt
   ```

2. Запуск приложения:

   ```
   sudo python3 app.py
   ```

3. Использование `TrafficSniffer`:

   ```
   sniffer = TrafficSniffer(interface="eth0")
   sniffer.start_sniffing()
   # ...
   sniffer.stop_sniffing()
   ```

9 Расширение функциональности

- Добавить новые графики: изменить маршруты `/analytics` и шаблон `analytics.html`
- Добавить новые признаки: расширить обработку в `packet_handler`
- Подключить другую модель: заменить файл `attack_model.pkl`, обновить энкодеры

10 Рекомендации и ограничения

10.1 Безопасность:

- отсутствует аутентификация и авторизация
- уязвимость к SQL-инъекциям — использовать параметризованные запросы

10.2 Производительность:

- `pyshark` может быть ресурсоёмким
- SQLite не подходит для масштабируемых решений

10.3 Масштабируемость:

- модульность архитектуры позволяет подключать внешние инструменты, такие как Suricata
- можно реализовать очередь обработки или потоковый анализ