Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б07-мм

Анализатор сетевых пакетов с веб-интерфейсом.

Диденко Андрей Антонович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
старший преподаватель кафедры системного программирования, И. В. Зеленчук

Санкт-Петербург
2023

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. CloudShark 5

2.2. PacketSafari 5

2.3. PacketTotal 5

3. Обзор используемых инструментов **6**

3.1. Flask 6

3.2. Bootstrap 6

3.3. dpkt 6

4. Реализация **7**

4.1. Проектирование сервиса 7

4.2. Реализация серверной части 8

4.3. Реализация графической составляющей 9

4.4. Парсинг и передача пакетов на сайт 10

4.5. Внедрение готового проекта в Miminet 13

Заключение **14**

Список литературы **15**

Введение

На матмехе есть предмет ”компьютерные сети”, для того, чтобы лучше обьяснить студентам дисциплину, было приянто решение написать веб приложение работы сети и передачи пакетов между устройствами сети.

Miminet [9] — это веб-симулятор компьютерной сети для образовательных целей. Работы над этим приложением ведутся при поддержке компании Yadro.

Wireshark [8] — программа-анализатор трафика для компьютерных сетей Ethernet и некоторых других. Имеет графический пользовательский интерфейс. Программа позволяет пользователю просматривать весь проходящий по сети трафик в режиме реального времени, переводя сетевую карту в неразборчивый режим. [10].

Для более наглядной работы компьютерных сетей в Miminet необходимо добавить такой инструмент, который сможет делать:

- Захват данных в реальном времени из сетевого интерфейса.
- Отображать пакеты с очень подробной информацией о протоколе.
- Открывать файлы, содержащие пакетные данные

В рамках данной работы предлагается реализовать веб-интерфес с отображением пакетов, как в Wireshark.

Дата сборки: 14 мая 2023 г.

1. Постановка задачи

Целью работы является интеграция веб-интерфейса, отображающего сетевые пакеты и информацию о протоколе в Miminet. Для её выполнения были поставлены следующие задачи:

1. провести обзор существующих веб аналогов, предоставляющих такие же функции, как Wireshark;
2. на основе требований спроектировать сервис;
3. сверстать веб-страницу;
4. передать информацию о пакетах на страницу;
5. внедрить реализованный сервис в Miminet.

2. Обзор

Так как изначально было понятно, в каком виде должен был выглядеть сайт (такой же интерфейс, как в Wireshark), задачей стало именно нахождение аналога Wireshark, но в виде веб-приложения, для того, чтобы его можно было внедрить в Miminet

2.1. CloudShark

Одним из наилучших аналогов оказался CloudShark [3]. Очень удобный и понятный проект, в котором реализованы все функции Wireshark. Единственный и существенный минус этого сервиса – он платный. В дальнейшем именно этот сайт стал референсом для реализации продукта.

2.2. PacketSafari

PacketSafari [4] - также является хорошим веб-анализатором трафика, но также является платным.

2.3. PacketTotal

PacketTotal [5] - бесплатный сервис с открытым api, но с неудобным интерфейсом, который не подходит для проекта.

У всех представленных сервисов очень много лишнего функционала, который не является необходимым в конечном продукте.

Исходя из собранной информации, было принято решение реализовать собственный сервис отображения пакетов.

3. Обзор используемых инструментов

3.1. Flask

Для реализации сервиса был выбран Flask [2]. Flask – фреймворк для создания веб-приложений на языке программирования Python, использующий набор инструментов Werkzeug, а также шаблонизатор Jinja2. Это гибкий и легкий инструмент, который позволяет облегчить процесс создания веб-приложений на Python. Flask – легкий, в отличии от его аналога Django. Он предоставляет только самые базовые инструменты, что отлично подходит для реализации небольшого веб-сервиса. Miminet также написан с использованием Flask.

3.2. Bootstrap

Для написания веб-приложения был выбран фреймворк Bootstrap5 [1]. Bootstrap - свободный набор инструментов для создания сайтов и веб-приложений. Включает в себя HTML- и CSS-шаблоны оформления для типографики, веб-форм, кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса, включая JavaScript-расширения. Он позволяет создать красивый дизайн приложения за короткое время. Еще одной причиной выбора именно этого фреймворка стало то, что Miminet разработан именно с использованием Bootstrap.

3.3. dpkt

Для анализа PCAP [6] файлов была выбрана очень простая в освоении библиотека для Python – dpkt [7]. dpkt — это модуль Python для быстрого и простого создания/анализа пакетов с определениями для основных протоколов TCP/IP. dpkt также, как и Flask и Bootstrap, использовался при написании Miminet.

4. Реализация

Для практической реализации использовался текстовый редактор Visual Studio Code и открытый репозиторий на GitHub.

4.1. Проектирование сервиса

После анализа аналогов, был сделан вывод, что за основу стоит взять CloudShark, так как его интерфейс максимально приближен к Wireshark. Для реализация серверной части был использован фреймворк Flask. Для верстки веб-страницы был выбран Bootstrap. Для парсинга пакетов была использована библиотека для Python – dpkt. Основной причиной выбора этих библиотек и фреймворков послужило то, что они были использованы при создании Miminet. Все pcap файлы после обработки представляли JSON (JavaScript Object Notation) файл, с помощью которого удобно передавать информацию в html часть сервиса.

Схема работы сервиса:

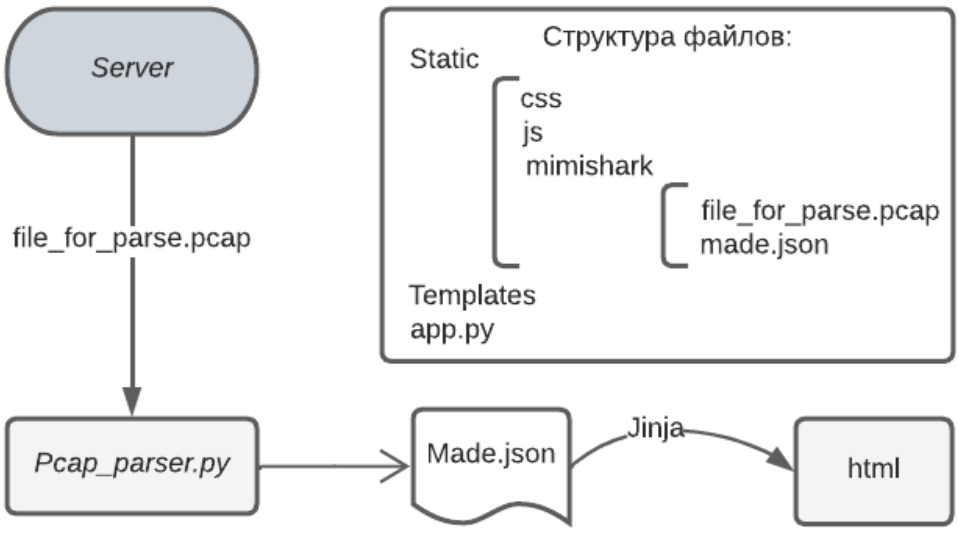

Рис. 1: Схема обработки файлов на сервере

4.2. Реализация серверной части

Испольщование Flask помогло без проблем созать серверную часть. Помог также встроенный в него шаблонизатор Jinja. Благодаря ему, можно с легкостью передавать в html обработанные в json pcap файлы. Учитывая, что Miminet написан на Flask, готовый продукт без труда можно было подключить в качестве новой вкладки.

Ниже представлен Flask декоратор – функция, которая запускает страницу проекта:

Листинг 1: **Основной декоратор**

```
@app.route('/MimiShark')
def main_page():
  data = ReadJson(path)
  return render_template('main.html',pcap_data = data)
```

В этой функции происходит считывание заготовленного json файла, а далее рендер нужной html страницы, с переданными данными.

Далее представлена работа шаблонизатора Jinja:

Листинг 2: **Передача pcap файлов в html**

```
{% block pcap %}
{% for el in pcap_data %}
<tr id="{{loop.index}}">
  <th scope="row">{{loop.index}}</th>
  <td>{{el.time}}</td>
  <td>{{el.source}}</td>
  <td>{{el.destination}}</td>
  <td>{{el.protocol}}</td>
  <td>{{el.length}}</td>
</tr>
{%endfor%}
{% endblock %}
```

Здесь представлен блок кода (block pcap), который будет отображен на веб-странице, в нем с помощью цикла, выводятся все файлы,

полученные в качестве переменной pcap data из предыдущего листинга.

4.3. Реализация графической составляющей

Как уже говорилось в обзоре используемых инструментов, с помощью Bootstrap, можно легко созать шаблон сайта за короткое время. Так как Miminet также был написан на Bootstrap, было принято решение сделать страницу по стилю, максимально приближенному к стилю Miminet. Поначалу задумывалось использовать приложение Figma, для составления наброска сайта. Но интерфейс сайта был настолько легким в реализациии, а шаблоны Bootstrap в освоении., что использование Figma было излишним.

Макет сайта по собранной информации:

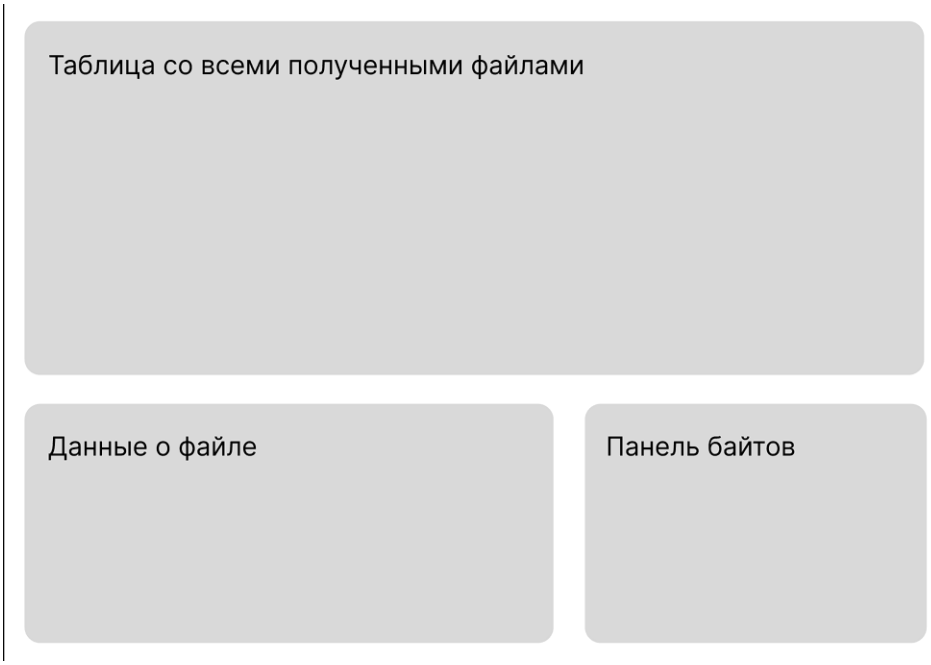

Рис. 2: Макет сайта

Конечная версия сайта выглядит так:

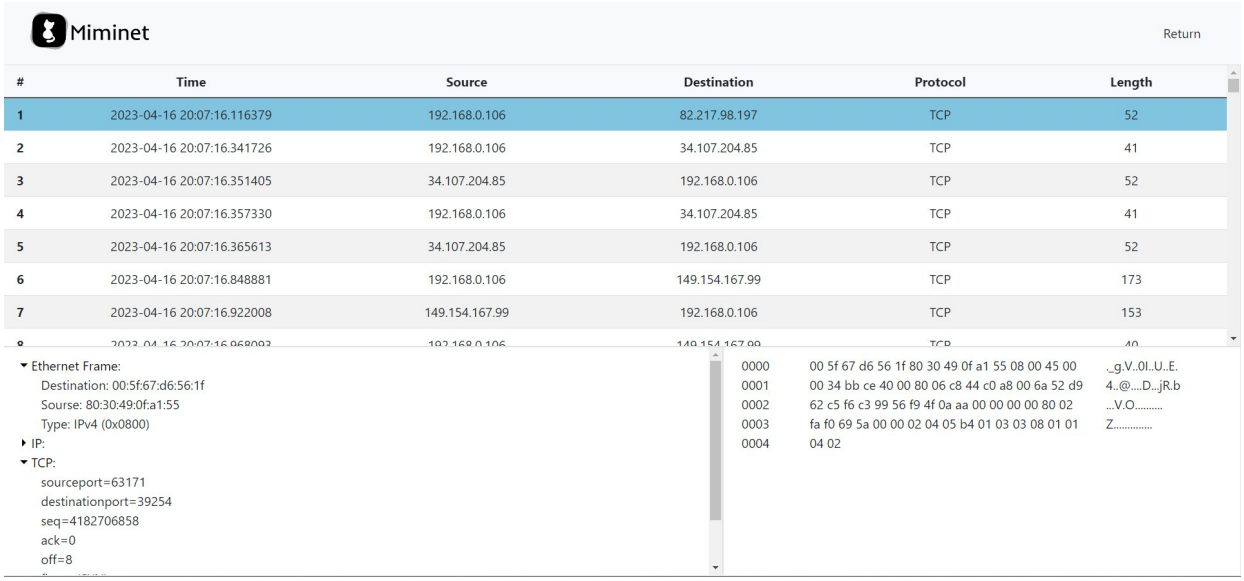

Рис. 3: Внешний вид сайта

Также для написания сайта использовался JavaScript. С его помощью были написаны функции вывода данных при нажатии по файлу из таблицы.

4.4. Парсинг и передача пакетов на сайт

Чтобы проверить правильность написания парсера, из Wireshark были экспортированные некоторые pcap файлы. Благодаря библиотеке dpkt из них легко можно было достать нужную информацию: Ethernet данные файла(место назначения переданных данных, место, откуда эти данные были переданы, его тип), версию и другие полезные данные интернет протокола, а также все данные протокола управления передачей.

На листинге представлена функция получения всех данных о протоколах:

Листинг 3: **Получение данных из протоколов**

```
import dpkt
def protocol_prop(self):
```

```
    l_ = []

    def add_field(fn, fv):
      l_.append('%s=%s, ' % (fn, fv))

    for field_name in self.__public_fields__:
        if isinstance (self, dpkt.tcp.TCP):
            tcp = self
            d = {dpkt.tcp.TH_FIN: 'FIN', dpkt.tcp.TH_SYN: 'SYN',
            dpkt.tcp.TH_RST: 'RST', dpkt.tcp.TH_PUSH: 'PUSH',
            dpkt.tcp.TH_ACK: 'ACK', dpkt.tcp.TH_URG: 'URG'}
            active_flags = filter(lambda t:
                                  t[0] & tcp.flags, d.items())
            flags_str = '+'.join(t[1] for t in active_flags)
            flag = f'({str(flags_str)})'

        add_field(field_name, getattr(self, field_name))

    ip_prot = '%s:' % self.__class__.__name__
    for ii in l_:
        ip_prot += '%s' % ii
    return ip_prot
```

В функции protocol properties есть дополнительная функция добовления поля с его значением в лист, из которого они в дальнейшем будут считываться в ответ. Далее функция проходит циклом по всем полям, при этом происходит проверка: если переданный файл содержит tcp протокол, то выводим нужное имя флага. Также в цикле определяются значения полей и добавляются в список для отображения. В конце функции собранные данные из списка добавляются в ответ, который по итогу функция возвращает.

На этом листинге представлен парсер, основанный на dpkt:

Листинг 4: **Работа парсера pcap файлов**

```
def add_packets(pcap):
  with open("pcap.json","w") as file:
      for  timestamp, buf in pcap:
          pcap_file = {}
          eth = dpkt.ethernet.Ethernet(buf)

          if not isinstance(eth.data, dpkt.ip.IP):
              continue

          pcap_file["time"] = str(datetime.datetime
                              .utcfromtimestamp(timestamp))

          ip = eth.data
          pcap_file["source"] = inet_to_str(ip.src)
          pcap_file["destination"] = inet_to_str(ip.dst)
          pcap_file["protocol"] = ip.get_proto(ip.p).__name__
          pcap_file["length"] = ip.len

          bytes_repr = ' '.join(mac_to_str(buf).split(':'))

          ascii = ''
          for i in bytes_repr.split(' '):
              a = bytes.fromhex(i)
              b = str(a)[2:len((str(a)))-1]
              if(len(b)<2):
                  ascii += b
              else:
                  ascii+= '.'
          pcap_file["ascii"] = ascii
          pcap_file["bytes"] = bytes_repr

          pcap_file["decode_eth"] = '%s:%s:%s' %
          (mac_to_str(eth.dst),
```

```
            mac_to_str(eth.src), bytes_repr[36:41])
            pcap_file["decode_ip"] = ip_protocol_prop(ip)
            pcap_file[f"decode_{ip.data}"] =
                                ip_protocol_prop(ip.data)

            json_file.append(pcap_file)
        print(json.dumps(json_file), file=file)
```

Перед вызовом этой функции идет открытие и считывание pcap файлов. В функции происходит следующее:

Для каждого пакета из pcap обрабатывается содержимое. Распаковывается ethernet frame. Достается ip пакетов и байты. Далее байты сопоставляются с ascii таблицей. А также подробно описываются данные файлов. Обработанные файлы отправляются в созданный json.

4.5. Внедрение готового проекта в Miminet

Завершающей частью работы была интеграция готового продукта в приложение Miminet. Для выполнения этой задачи была изменена не только сктруктура файлов проекта, но и исходные файлы Miminet. С помощью шаблонизатора Jinja веб-страница анализатора была перебазированна под стиль, который используется разработчиками Miminet. Переход на страницу анализатора происходит по добавленной для каждого компонента ссылке, которая содержит свой маршрут – /MimiShark?guid=..., где guid – статистически уникальный 128-битный идентификатор, присваеваемый каждому конструктору сети. На странице выводятся файлы, содержащиеся в компоненте.

Заключение

В результате работы был реализован анализатор сетевых пакетов для Miminet. Были выполнены поставленные задачи:

- проведен обзор веб-аналогов Wireshark;
- спроектирован сервис;
- написана веб-страница;
- информация о пакетах выведена на страницу;
- конечный продукт интегрирован в Miminet.

Дальнейшие планы:

- реализовать некоторые функции (фильтры поиска и т.д.);
- добавить возможность импорта/экспорта данных;

Всю проделанную работу можно увидеть на GitHub репозитории [1].

[1] https://github.com/K0lba/MiminetShark (Дата доступа: 2023-05-09)

Список литературы

[1] Bootstrap. — URL: https://getbootstrap.com/ (дата обращения: 2023-05-09).

[2] Flask. — URL: https://flask.palletsprojects.com/en/2.3.x/ (дата обращения: 2023-05-09).

[3] MCEACHERN JOE. CloudShark Enterprise – QA Cafe. — URL: https://www.qacafe.com/analysis-tools/cloudshark/ (дата обращения: 2023-05-09).

[4] Packet Safari. — URL: https://www.packetsafari.com/ (дата обращения: 2023-05-09).

[5] Packet Total. — URL: https://packettotal.com/ (дата обращения: 2023-05-09).

[6] Pcap are data files created by the program and they contain packet data on the network. — URL: https://ru.wikipedia.org/wiki/Pcap (дата обращения: 2023-05-14).

[7] dpkt documentation. — URL: https://kbandla.github.io/dpkt/ (дата обращения: 2023-05-09).

[8] Анализатор сетевого трафика WireShark. — URL: https://www.wireshark.org/ (дата обращения: 2023-05-14).

[9] Зеленчук И. В. Симулятор компьютерных сетей Miminet. — URL: https://miminet.ru/ (дата обращения: 2023-05-14).

[10] Описание приложения Wireshark. — URL: https://ru.wikipedia.org/wiki/Wireshark (дата обращения: 2023-05-08).