

# Описание программного макета.

## Содержание.

В zip архиве представлены файлы:

- 1) Jupyter main.ipynb.
- 2) Python файл main.py
- 3) requirements.txt, содержащий список требуемых библиотек для работы программы
- 4) model.pkl, содержащий обученную сохраненную модель нейронной сети
- 5) data.csv с собранной обучающей выборкой
- 6) requests.jmx с настройками для Apache Jemeter, имитирующий обычную сетевую нагрузку на сервер.

## Сбор сетевой статистики, используемые устройства.

Во время создания макеты были использованы:

- 1) Виртуальная машина Ubuntu-Server, с установленным сервером apache. К нему осуществлялись запросы имитирующие сетевой трафик с помощью Apache Jemeter, а также атаки с помощью встроенных в Kali Linux утилит.
- 2) Виртуальная машина с установленным дистрибутивом Kali Linux, с ее помощью осуществлялись атаки на Ubuntu-Server.

## Пример запуска программы и имитация пользовательской статистики

На данном снимке пример запуска приложения Apache-Jemeter с запросами к серверу Ubuntu.

(При имитации пользовательской активности я примерно раз в несколько часов немного изменял настройки для Jemeter-а, чтобы имитировать

## периоды повышенной и пониженной нагрузки)

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(ms)
18488	12-43:24.079	Thread Group 4-5	HTTP Request	1	✓	10981	900	1	0
18489	12-43:24.119	Thread Group 3-11	HTTP Request	11	✓	10982	576	11	1
18490	12-43:24.129	Thread Group 2-1	HTTP Request	2	✓	10982	221	2	1
18491	12-43:24.182	Thread Group 3-5	HTTP Request	1	✓	10982	576	1	0
18492	12-43:24.227	Thread Group 3-3	HTTP Request	1	✓	10982	221	1	0
18493	12-43:24.240	Thread Group 3-4	HTTP Request	1	✓	10982	576	1	0
18494	12-43:24.338	Thread Group 1-15	HTTP Request	1	✓	10982	117	1	0
18495	12-43:24.353	Thread Group 3-12	HTTP Request	1	✓	10982	576	1	1
18496	12-43:24.379	Thread Group 1-8	HTTP Request	1	✓	10982	117	1	0
18497	12-43:24.433	Thread Group 1-6	HTTP Request	1	✓	10982	117	1	1
18498	12-43:24.453	Thread Group 4-2	HTTP Request	0	✓	10981	900	0	0
18499	12-43:24.545	Thread Group 3-1	HTTP Request	2	✓	10982	576	1	1
18500	12-43:24.550	Thread Group 1-11	HTTP Request	1	✓	10982	117	1	0
18501	12-43:24.591	Thread Group 1-12	HTTP Request	1	✓	10982	117	1	0
18502	12-43:24.611	Thread Group 4-1	HTTP Request	1	✓	10981	900	1	0
18503	12-43:24.675	Thread Group 3-7	HTTP Request	6	✓	10982	576	3	0
18504	12-43:24.677	Thread Group 1-14	HTTP Request	5	✓	10982	117	5	1
18505	12-43:24.687	Thread Group 3-6	HTTP Request	3	✓	10982	576	3	0
18506	12-43:24.686	Thread Group 4-7	HTTP Request	1	✓	10981	900	1	0
18507	12-43:24.772	Thread Group 1-15	HTTP Request	1	✓	10982	117	1	0
18508	12-43:24.773	Thread Group 1-7	HTTP Request	2	✓	10982	117	2	1
18509	12-43:24.774	Thread Group 1-9	HTTP Request	1	✓	10982	117	1	0
18510	12-43:24.784	Thread Group 3-10	HTTP Request	1	✓	10982	576	1	1
18511	12-43:24.846	Thread Group 1-1	HTTP Request	1	✓	10982	117	1	0
18512	12-43:24.862	Thread Group 4-1	HTTP Request	1	✓	10981	900	1	0
18513	12-43:24.931	Thread Group 3-1	HTTP Request	1	✓	10982	576	1	0
18514	12-43:24.961	Thread Group 2-2	HTTP Request	1	✓	10982	221	1	1
18515	12-43:24.995	Thread Group 1-14	HTTP Request	1	✓	10982	117	1	0
18516	12-43:25.013	Thread Group 2-4	HTTP Request	1	✓	10982	221	1	0
18517	12-43:25.035	Thread Group 3-9	HTTP Request	1	✓	10982	576	1	0
18518	12-43:25.084	Thread Group 4-6	HTTP Request	3	✓	10981	900	3	0
18519	12-43:25.083	Thread Group 1-5	HTTP Request	1	✓	10982	117	1	0
18520	12-43:25.103	Thread Group 1-2	HTTP Request	1	✓	10982	117	1	1
18521	12-43:25.130	Thread Group 1-4	HTTP Request	1	✓	10982	117	1	0
18522	12-43:25.132	Thread Group 1-11	HTTP Request	2	✓	10982	117	2	1

Ниже представлен скриншот виртуальной машины сервера. На нем запуск программы в режиме анализа трафика, по истечении минут я прерываю программу, скрипт выводит запись о том, что атак не обнаружено.

```
root@server:/media/sf_obsh# python3 main.py
/media/sf_obsh/main.py:7: DeprecationWarning:
Pyyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0)
,
(to allow more performant data types, such as the Arrow string type, and better interoperability with
other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

import pandas as pd
[?] Выберите пункт меню:
Сбор онлайн трафика
> Анализ онлайн трафика
Сбор трафика из .rsar файла
Выход

[?] Сохранять пакетную статистику в .rsar файл?:
Да
> Нет

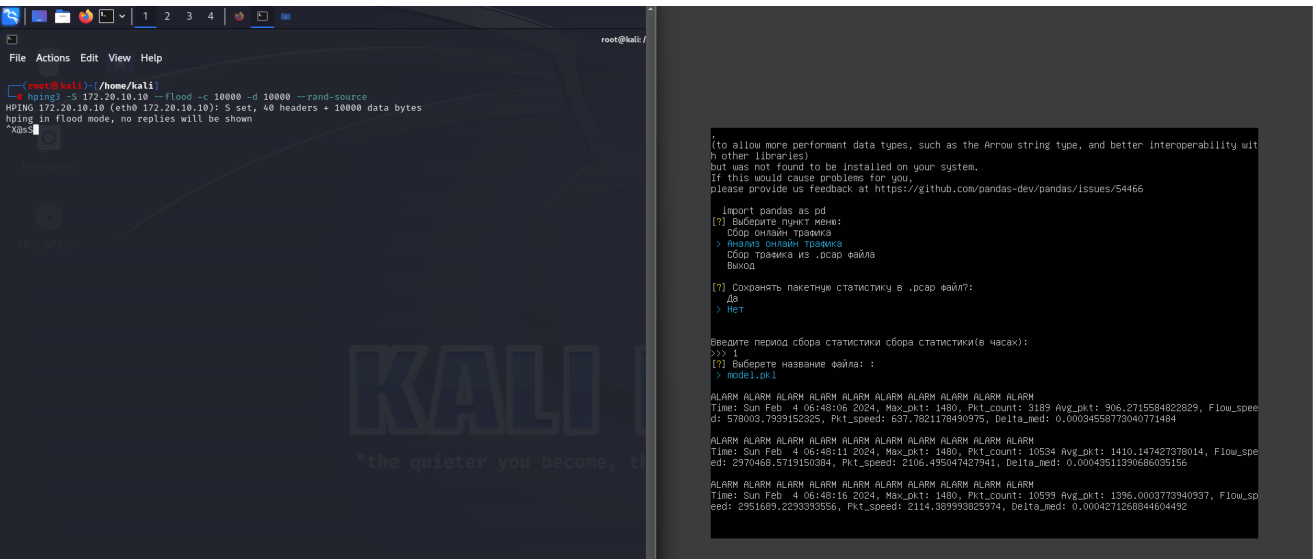
Введите период сбора статистики(в часах):
>>> 1
[?] Выберите название файла: :
> model.pkl

^C^C^C^CСниффер остановлен
Атак не зафиксировано.

[?] Выберите пункт меню:
> Сбор онлайн трафика
Анализ онлайн трафика
Сбор трафика из .rsar файла
Выход
```

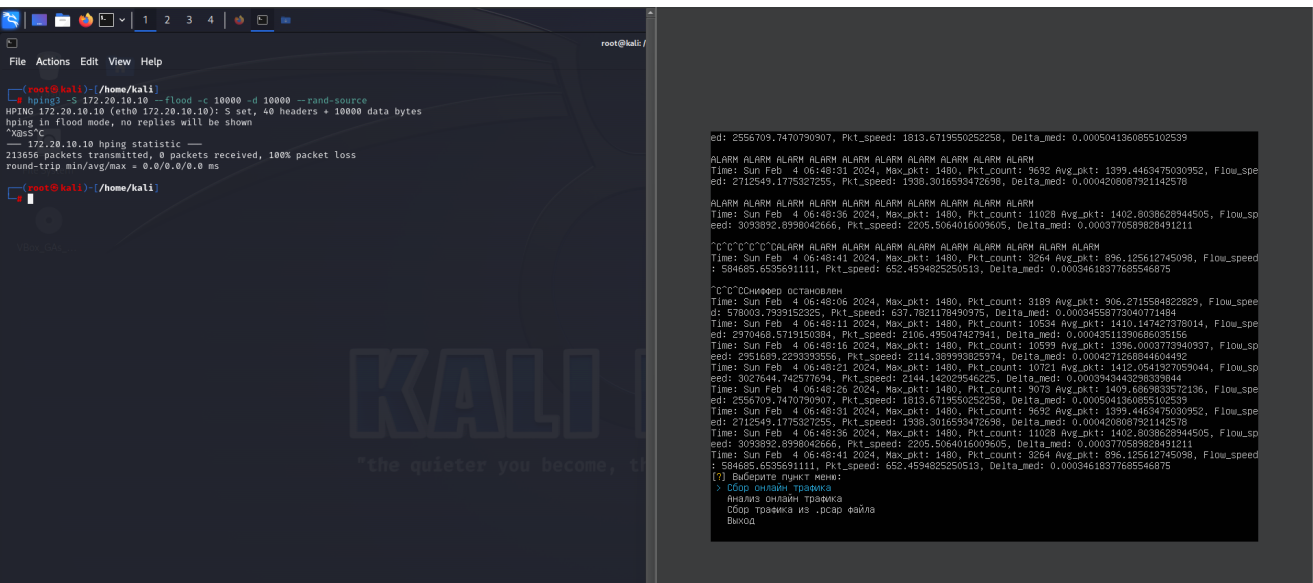
## Пример запуска программы и имитация атаки

Атака осуществляется с помощью встроенной утилиты в дистрибутив kali linux hping3, его конфигурация, а также вывод сообщений об атаке в консоль на скриншоте.



```
root@kali: /  
File Actions Edit View Help  
root@kali) ~/home/kali  
hping3 -S 172.20.10.10 -flood -c 10000 -d 10000 --rand-source  
HPING 172.20.10.10 (eth0 172.20.10.10): S set, 40 headers + 10000 data bytes  
hping in flood mode, no replies will be shown  
*Xbs5C  
[?] Выберите пункт меню:  
  Опер онлайн трафика  
  Опер трафика из .pcap файла  
  Выход  
[?] Сохранять пакетные статистику в .pcap файл?:  
  Да  
  Нет  
Введите период сбора статистики (в часах):  
>>> 1  
[?] Выберите название файла: :  
  model.pkt  
ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:06 2024, Max_pkt: 1480, Pkt_count: 3189 Avg_pkt: 906.2715584822829, Flow_speed  
ed: 578003.7939152325, Pkt_speed: 637.7821178490975, Delta_med: 0.00034558773040771484  
ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:11 2024, Max_pkt: 1480, Pkt_count: 10534 Avg_pkt: 1410.147427378014, Flow_spe  
ed: 2970468.5719150384, Pkt_speed: 2106.495047427941, Delta_med: 0.00043511390686035156  
ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:16 2024, Max_pkt: 1480, Pkt_count: 10599 Avg_pkt: 1396.0003773340937, Flow_sp  
eed: 2951689.2233333556, Pkt_speed: 2114.38993825374, Delta_med: 0.0004271268844604432
```

Дальше я снова прерываю программу. Скрипт выводит время обнаружения атаки, также статистику по каждому интервалу.



```
root@kali: /  
File Actions Edit View Help  
root@kali) ~/home/kali  
hping3 -S 172.20.10.10 -flood -c 10000 -d 10000 --rand-source  
HPING 172.20.10.10 (eth0 172.20.10.10): S set, 40 headers + 10000 data bytes  
hping in flood mode, no replies will be shown  
*Xbs5C  
--- 172.20.10.10 hping statistic ---  
21665 packets transmitted, 0 packets received, 100% packet loss  
round-trip min/avg/max = 0.0/0.0/0.0 ms  
root@kali) ~/home/kali  
ed: 2556709.7470790907, Pkt_speed: 1813.6719550252558, Delta_med: 0.0005041360895102539  
ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:13 2024, Max_pkt: 1480, Pkt_count: 9592 Avg_pkt: 1399.4463475030952, Flow_spe  
ed: 2712545.1775327255, Pkt_speed: 1938.3015593972598, Delta_med: 0.0004208087921142578  
ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:16 2024, Max_pkt: 1480, Pkt_count: 11028 Avg_pkt: 1402.8038628944505, Flow_sp  
eed: 3093892.8998042666, Pkt_speed: 2205.5064016009605, Delta_med: 0.0003770589828491211  
"C"C"C"C"ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM  
Time: Sun Feb  4 06:48:41 2024, Max_pkt: 1480, Pkt_count: 3264 Avg_pkt: 896.125612745098, Flow_speed  
ed: 584685.6535691111, Pkt_speed: 652.4594825250513, Delta_med: 0.00034618377685546875  
"C"C"C"снмпер остановлен  
Time: Sun Feb  4 06:48:06 2024, Max_pkt: 1480, Pkt_count: 3189 Avg_pkt: 906.2715584822829, Flow_spe  
ed: 578003.7939152325, Pkt_speed: 637.7821178490975, Delta_med: 0.00034558773040771484  
Time: Sun Feb  4 06:48:11 2024, Max_pkt: 1480, Pkt_count: 10534 Avg_pkt: 1410.147427378014, Flow_spe  
ed: 2970468.5719150384, Pkt_speed: 2106.495047427941, Delta_med: 0.00043511390686035156  
Time: Sun Feb  4 06:48:16 2024, Max_pkt: 1480, Pkt_count: 10599 Avg_pkt: 1396.0003773340937, Flow_sp  
eed: 2951689.2233333556, Pkt_speed: 2114.38993825374, Delta_med: 0.0004271268844604432  
Time: Sun Feb  4 06:48:21 2024, Max_pkt: 1480, Pkt_count: 10721 Avg_pkt: 1412.0541327059044, Flow_sp  
eed: 3027644.742577894, Pkt_speed: 2144.162029546225, Delta_med: 0.000394344828333844  
Time: Sun Feb  4 06:48:26 2024, Max_pkt: 1480, Pkt_count: 9073 Avg_pkt: 1409.6889833572136, Flow_spe  
ed: 2556709.7470790907, Pkt_speed: 1813.6719550252558, Delta_med: 0.0005041360895102539  
Time: Sun Feb  4 06:48:31 2024, Max_pkt: 1480, Pkt_count: 9692 Avg_pkt: 1399.4463475030952, Flow_spe  
ed: 2712545.1775327255, Pkt_speed: 1938.3015593972598, Delta_med: 0.0004208087921142578  
Time: Sun Feb  4 06:48:36 2024, Max_pkt: 1480, Pkt_count: 11028 Avg_pkt: 1402.8038628944505, Flow_sp  
eed: 3093892.8998042666, Pkt_speed: 2205.5064016009605, Delta_med: 0.0003770589828491211  
Time: Sun Feb  4 06:48:41 2024, Max_pkt: 1480, Pkt_count: 3264 Avg_pkt: 896.125612745098, Flow_speed  
ed: 584685.6535691111, Pkt_speed: 652.4594825250513, Delta_med: 0.00034618377685546875  
[?] Выберите пункт меню:  
  Опер онлайн трафика  
  Опер трафика из .pcap файла  
  Выход
```

В файле attack\_stat.txt появляются записи:

```
Time: Sun Feb  4 06:48:06 2024, Max_pkt: 1480, Pkt_count: 3189 Avg_pkt:  
906.2715584822829, Flow_speed: 578003.7939152325, Pkt_speed:  
637.7821178490975, Delta_med: 0.00034558773040771484  
.....(для экономии места сократил записи)  
Time: Sun Feb  4 06:48:41 2024, Max_pkt: 1480, Pkt_count: 3264 Avg_pkt:  
896.125612745098, Flow_speed: 584685.6535691111, Pkt_speed:  
652.4594825250513, Delta_med: 0.00034618377685546875
```

# Обработка статистики, обучение нейронной сети.

Статистика собирается в csv файл, обучение нейронной сети, а также комментарии к коду представлены в main.ipynb Jupyter Notebook.

Единственное, что хочу добавить, было выбрано обучение с учителем. Для этого данные предварительно собирались двумя "пачками" одна без атаки, другая с атакой, к ним сразу при сборе статистики добавлялась колонка status. 0 - без атаки, 1 - с атакой. После они соединялись в один файл, на котором и происходило обучение.

Также я попробовал на исходной выборке запустить НС, обучающуюся без учителя. Была использована модель KMeans, в параметрах был указано количество кластеров - 2. Однако высокой точности данное решение не показало. На момент написания этого текста, с моего последнего подобного эксперимента, прошло порядка недели, так что я не могу вспомнить показатели по f1 мере, что-то порядка 60%. К данным была применена стандартизация по минимаксу, возможно отбросив некоторые параметры, можно было получить точность выше, однако с этим я решил не возиться и представил, что нам известно какие данные были записаны при атаке, а какие нет.

## Комментарии к коду программы(main.py)

Все комментарии из кода я продублировал сюда.

```
def offline_sniffing(mac=scapy.Ether().src):
    sniffer_data = scapy.sniff(offline='output.pcap')
    try:
        pkt_time = sniffer_data[0].time
    except IndexError:
        print('Файл пуст!')
        sys.exit(-1)

    pack_times.append(pkt_time)
    dumps_time.append(pkt_time)

    for pkt in sniffer_data:
        data_selection(pkt, mac)
```

Функция `offline_sniffer` была создана для сбора статистики из уже имеющегося файла формата `.pcap`. Я решил просто добавить данную функцию например для создания оцифрованных данных для НС, собранных посредством какой-либо утилиты, например тот же `WireShark`. В параметрах у него `mac` адрес устройства, для того, чтобы можно было запускать скрипт с другой машины.

```
def online_sniffing(start, timeout, store=True, endpoint=24 * 60 * 60):
    while time.time() - start <= endpoint:
        try:
            if store:
                sniffer_data = scapy.sniff(store=store,
prn=data_selection, timeout=timeout)
                scapy.wrpcap('output.pcap', sniffer_data, append=True)
            else:
                scapy.sniff(store=store, prn=data_selection,
timeout=timeout)
        except KeyboardInterrupt:
            print('Сниффер остановлен')
            break
```

Функция `online_sniffing` была создана для сбора статистики онлайн, а также для анализа трафика, она принимает на вход параметры.

`start` - время запуска сниффера.

`timeout` - время после которого сниффер будет останавливаться и перезапускаться. Нужен для сбора статистики онлайн, чтобы не перегружать `store`

`store` - параметр, который отвечает за хранение данных сниффера, принимает `True` или `False`.

`endpoint` - время при достижении которого программа прекратит работу, по умолчанию 24 часа.

```
def data_selection(pkt, mac=scapy.Ether().src):
    if pkt.dst == mac:
        if pkt.haslayer(scapy.Raw):
            write_to_dict(pkt.time, pkt[scapy.Raw].load)
        else:
            write_to_dict(pkt.time)

    if pkt.time - dumps_time[-1] >= dump_const:
```

```
data_collect(pkt.time - dumps_time[-1])
dumps_time.append(pkt.time)
```

Функция `data_selection` вызывается на каждый пакет собранный сниффером, параметр по умолчанию сам пакет, из функции `offline_sniffing` она вызывается с дополнительным параметром `mac` адреса.

В функции происходит проверка, которая фильтрует только пакеты приходящие на сервер. Данные из пакета записываются в словарь посредством вызова функции `write_to_dict()`.

Также по истечении времени, которое установлено `dump_const` вызывается функция `data_collect`, по умолчанию это происходит каждые 5 секунд.

К сожалению я не смог придумать реализацию программы без использования глобальных переменных. Основная причина для меня кроется в самом сниффере, который не позволяет передавать функции `data_selection` дополнительные параметры. если бы такая опция была, я бы переписал код с меньшим использованием или вообще без глобальных переменных, а также с большей смысловой нагрузкой для функций.

```
def write_to_dict(curr_time, curr_data=b''):
    data_size = len(curr_data)
    delta = curr_time - pack_times[-1]

    if delta != 0:
        delta_time.append(curr_time - pack_times[-1])

    pack_times.append(curr_time)

    packet_stats['sum_data'] += data_size
    packet_stats['max_packet'] = max(data_size,
    packet_stats['max_packet'])
    packet_stats['packet_counter'] += 1
```

Функция `write_to_dict` принимает на вход время пакета, а также его содержимое(если оно есть), далее собираются параметры, которые записываются в глобальный словарь `packet_stats`. Параметры являются оцифрованной пакетной статистикой.

```

def data_collect(sec):
    if sec == 0:
        return -1

    sum_data = packet_stats['sum_data']
    max_pkt = packet_stats['max_packet']
    pkt_count = packet_stats['packet_counter']
    avg_pkt = sum_data / pkt_count if pkt_count != 0 else 1
    flow_speed = float(sum_data / sec)
    pkt_speed = float(pkt_count / sec)
    delta_med = float(median(delta_time))
    delta_min = float(min(delta_time))

    data_frame.append({
        'sum_data': sum_data,
        'max_pkt': max_pkt,
        'pkt_count': pkt_count,
        'avg_pkt': avg_pkt,
        'flow_speed': flow_speed,
        'pkt_speed': pkt_speed,
        'delta_min': delta_min,
        'delta_med': delta_med
    })

    if flag:
        model_predict(sum_data, max_pkt, pkt_count, avg_pkt, flow_speed,
            pkt_speed, delta_med)

    packet_stats.clear()
    delta_time.clear()

```

Функция `data_collect` собирает оцифрованные данные, сохраняет их в переменную `data_frame`, после чего очищает лист, который хранит значения межпакетных интервалов, и словарь. Также из нее вызывается функция `model_predict`, если пользователь выбрал модель НС.

```

def model_predict(sum_data, max_pkt, pkt_count, avg_pkt, flow_speed,
    pkt_speed, delta_med):

    X = np.array([[sum_data, max_pkt, pkt_count, avg_pkt, flow_speed,
        pkt_speed, delta_med]])
    y = int(model.predict(X)[0])

```

```

if y:
    stat = f'Time: {time.ctime(int(time.time()))}, Sum_data:
{sum_data} Max_pkt: {max_pkt}, Pkt_count: {pkt_count}' + \
        f' Avg_pkt: {avg_pkt}, Flow_speed: {flow_speed},
Pkt_speed: {pkt_speed}, Delta_med: {delta_med}\n'

    print('ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM ALARM
ALARM \n' + stat)

    with open('attack_stat.txt', 'a') as f:
        f.write(stat)

```

Функция `model_predict` вызывает подгруженную модель НС, а также выводит информацию пакетной статистики, если обнаруживает аномальную активность. После чего записывает ее в файл.

Оставшиеся функции я комментировать не буду)) Они были написаны для создания видимости пользовательского интерфейса. По факту во время использования программы для выполнения задачи задействованы они не были.

## Заключение

Собственно, не знаю что еще можно добавить в качестве комментария к самой программе. Рефлексия по задаче разве что. Было интересно попробовать сделать что-то подобное. До этого я в целом никогда не занимался сбором сетевой статистики и тд, а применение НС было только в рамках учебных задач, поэтому фактически подобный опыт у меня впервые.

По поводу моего проекта, будем считать, что это просто MVP, который требует доработок) Можно много чего улучшить, например ту же самую сборку статистики реализовать не по времени, а по завершении сессии/ потока, дать большую смысловую нагрузку функциям, написать свой обработчик пакетов, который будет адаптирован под задачу, в отличие от `scapy` и тд.

На этом все, добавлю ссылку на гитхаб, хоть там и одно открытое репо, может будет нужно: <https://github.com/nedeadininside>