Описание программного макета.

Содержание.

В zip архиве представлены файлы:

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

Сбор сетевой статистики, используемые устройства.

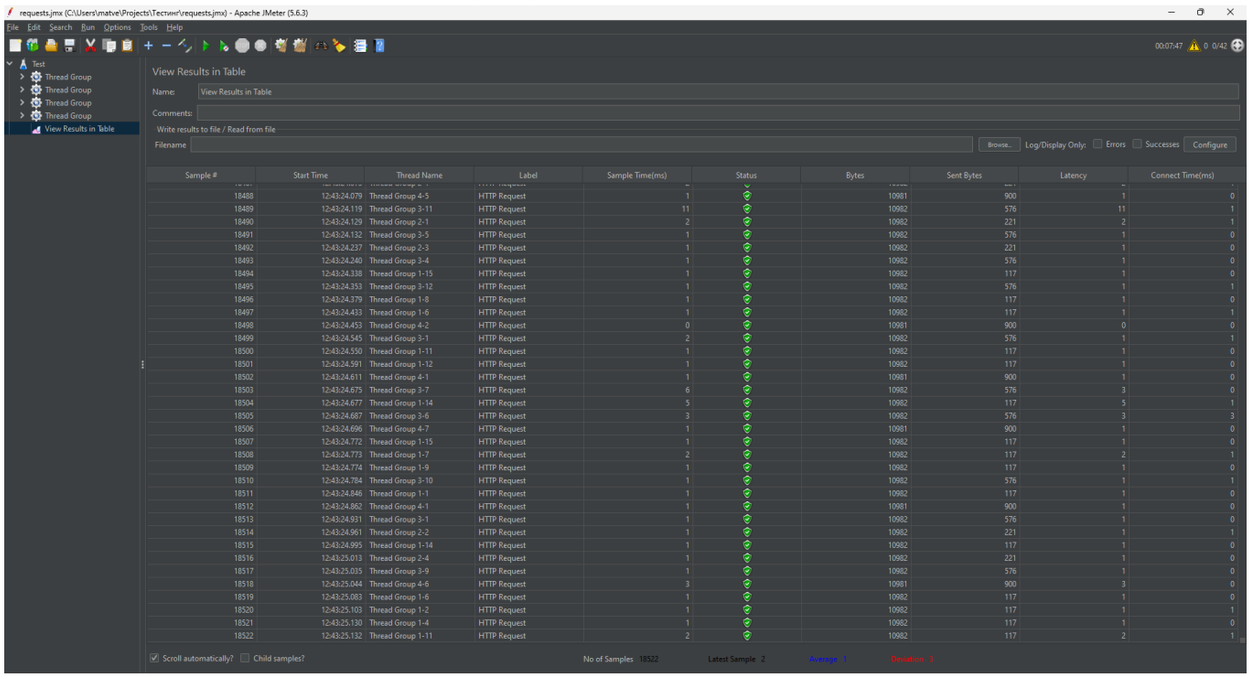
Во время создания макеты были использованы:

- 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.
- 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)



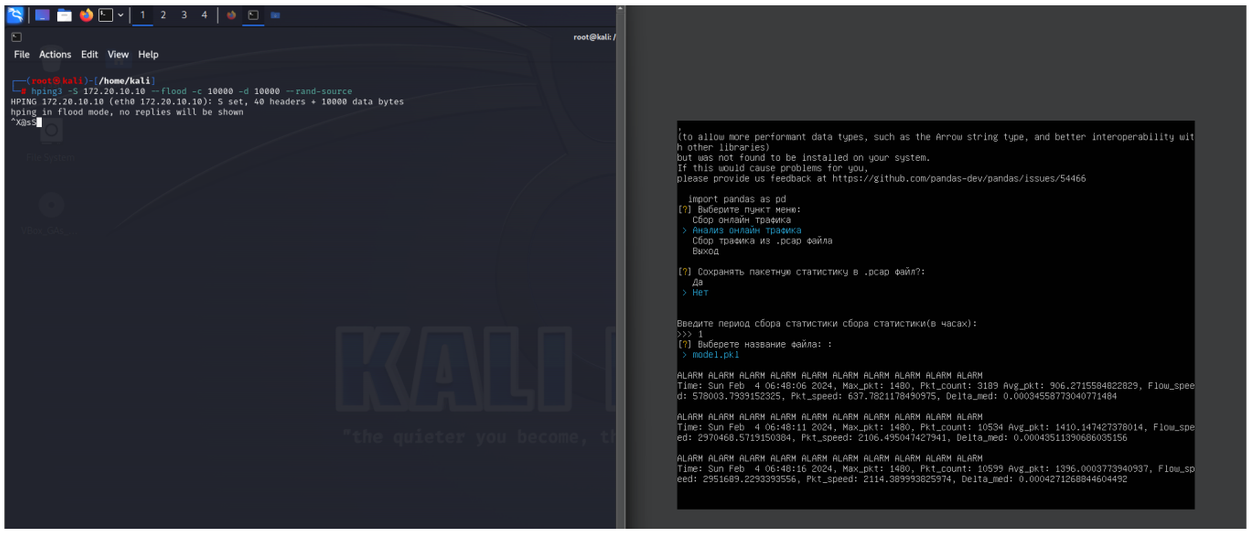
Ниже представлен скриншот виртуальной машины сервера. На нем запуск программы в режиме анализа трафика, по истечении минут я прерываю программу, скрипт выводит запись о том, что атак не обнаружено.

```
root@server:/media/sf_obsh# python3 main.py
/media/sf_obsh/main.py:7: DeprecationWarning:
Pyarrow 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 wit
h 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 .pcap файла
   Выход

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


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

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

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

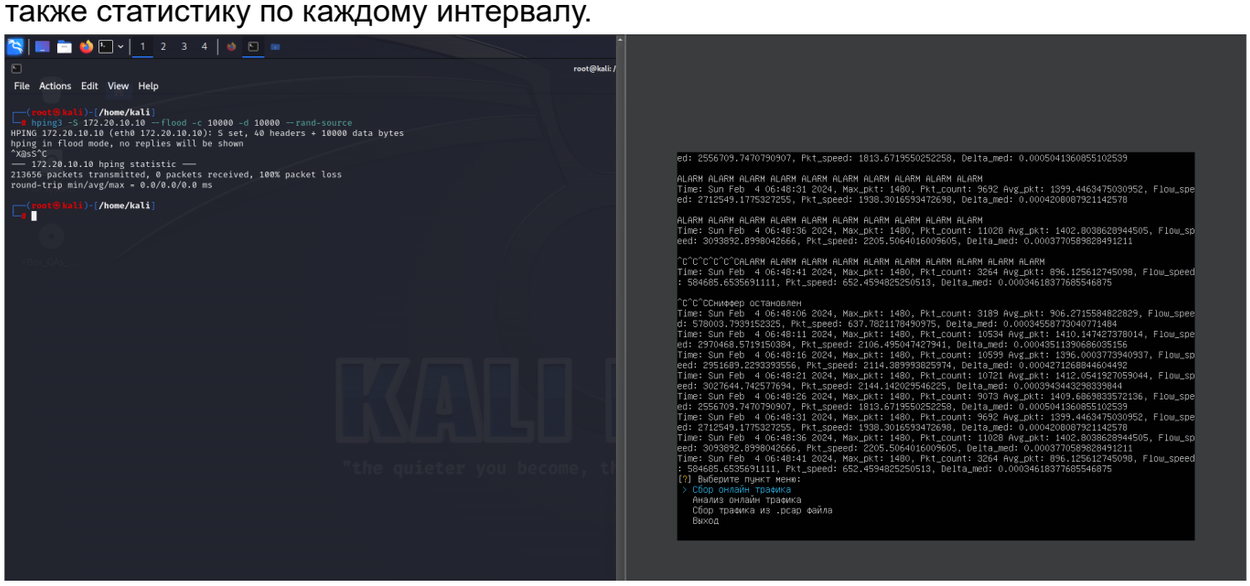
Пример запуска программы и имитация атаки

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

скриншоте.



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



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
```

Описание используемого датасета

Так как датасет собирался под контролем, данные были размечены изначально вручную. Условно я знал, что сейчас буду имитировать норму, поэтому добавлял к каждой записи столбец status=0, при записи атак соответственно status=1.

Количество записей, время формирования датасета

Время записи датасета составляло 19 часов, из них примерно 4.5 часа составляют данные атак, оставшееся время моделировался нормальный сетевой трафик

Итого датасет имеет 13750 записей: из которых 3184 записей приходятся на атаки, а оставшиеся 10566 на нормальную сетевую активность.

Записи атак

В качестве данного источника данных был выбран самый простой вариант - DOS атаки, их было достаточно легко воспроизвести для примера.

Во время записи атак, как я уже упоминал выше использовался встроенный инструмент Kali Linux hping3, который генерировал запросы со следующими параметрами:

- S флаг отвечал за подключение по протоколу TCP.
- d - объем передаваемых пакетом данных. Сами данные не содержат какой-либо смысловой нагрузки, их объем(в байтах) варьировался от 0 до 1480, хоть в примерах запросов и указано большее значение, это ограничивалось MTU = 1500, по крайней мере в нашем примере. Сами же значения не выбирались рандомно и менялись дискретно: 0, 500, 1000, 1480.
- Также использовался флаг --flood, который отвечал за минимальный межпакетный интервал, при такой установке каждый пакет отправляется с минимально возможным временным промежутком, примерная скорость 950-1000 пакетов в секунду.
- Параметр --rand-source фактически не играл для нас роли. Он подменял исходный адрес машины, которая отправляла запросы, на рандомный. Однако ip(или mac адрес) не были задействованы в ходе формирования датасета.

```
Итоговый запрос для имитации атак выглядел примерно так:
hping3 -S 111.11.11.11 --flood -d 1000 --rand-source
```

Также хочу добавить, что во время исполнения данной команды не отключался Apache-Jemeter, который отвечал за формирования нормального трафика.

Записи нормы

В качестве записи норм использовались повторяющиеся post и get запросы, формируемые с помощью Apache Jemeter. Его параметры описывать подробно я не буду, напишу лишь общее представление.
Примерно имитировалось от 15 до 50 пользователей(потоков) параллельно взаимодействующих с сервером, пользователи были разбиты на 4 группы: 2 разных post запроса и 2 разных get запроса, каждый из которых содержал примерно от 0 до 14000 байт данных. Тут, по сравнению с данными атак, они менялись чаще и хаотичнее. Также между запросами случайно формировался интервал от 1 до 2000 миллисекунд.

Не могу что-либо еще добавить к информации о норме. Разве что частоту изменения параметров. Примерно каждые 1-1.5 часа я изменял параметры в большую или меньшую сторону для вариативности данных.

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.

При формировании Train сета и Test сета использовалась StratifiedShuffleSplit, которая сохраняла соотношение исходных данных атак и нормы(изначальное соотношение составляет ~77/23) При этом 30% датасета уходила на тест, а оставшиеся 70% на обучение. В прикрепленном jupyter notebook оставлены комментарии к

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.

Для контроля качества предсказаний модели можно добавить следующий код:

```
incorrect_predictions = [i for i in range(len(y_pred)) if y_pred[i] !=
y_test.values[i]]
```

Он позволит нам увидеть индексы предсказаний, которые расходятся с размеченными данными, что позволит обратиться к тестовому набору и рассмотреть их. 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.

Результаты тестирования

Обработка предсказаний модели

В качестве метрики был выбран f1_score:

```
from sklearn.metrics import f1_score
y_pred = rfc.predict(X_test.values)

f1_score(y_pred, y_test)
```

```
0.9989528795811519
```

По итогам тестирования модель показала следующие результаты:

Total	Norma detect	Attack detect	Issues norma predict	Issues attack predict
4125	3169	954	1	1

Где :

- **Total** - общий объем train сета
- **Norma detect** - количество правильно определенных записей нормы
- **Attack detect** - количество правильно определенных записей атак
- **Issues norma predict** - количество записей, в которых модель допустила ошибку при определении записи нормы
- **Issues attack predict** - количество записей, в которых модель допустила ошибку при определении записи атаки

Данные этой таблицы сохраняются в файл **report.csv**. Исходный код с формированием таблицы, обработкой предсказаний модели и тд. находится в файле блокнота Jupyter: main.ipynb.

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/nedeadinside