

Анализ сетевого трафика с помощью Python и Scapy

Саяпин Никита Анатольевич

Приамурский государственный университет им.Шолом-Алейхема

Студент

Аннотация

Статья посвящена базовому анализу сетевого трафика в контексте кибербезопасности с использованием Python и библиотеки scapy. Рассматриваются основы перехвата и анализа сетевых пакетов, включая извлечение IP-адресов источника и назначения для TCP-пакетов. В практической части представлен простой скрипт, демонстрирующий перехват пакетов, который легко воспроизвести на локальной машине. Статья подходит для начинающих специалистов по кибербезопасности и сетевым технологиям, желающих освоить базовые инструменты мониторинга сети.

Ключевые слова: кибербезопасность, сетевой трафик, анализ пакетов, Python, Scapy, TCP, IP, мониторинг сети.

Network traffic analysis using Python and Scapy

Sayapin Nikita Anatolyevich

Sholom-Aleichem Priamursky State University

Student

Abstract

The article is devoted to the basic analysis of network traffic in the context of cybersecurity using Python and the scapy library. The basics of intercepting and analyzing network packets are discussed, including extracting source and destination IP addresses for TCP packets. In the practical part, a simple script is presented that demonstrates packet interception, which is easy to reproduce on a local machine. The article is suitable for novice cybersecurity and network technology specialists who want to master basic network monitoring tools.

Keywords: cybersecurity, network traffic, packet analysis, Python, Scapy, TCP, IP, network monitoring.

1. Введение

1.1 Актуальность

В условиях стремительного роста киберугроз, таких как DDoS-атаки, фишинг и утечки данных, анализ сетевого трафика становится критически важным для обеспечения безопасности информационных систем. По данным отчётов кибербезопасности 2025 года, более 60% организаций сталкиваются с инцидентами, связанными с несанкционированным доступом к сети. Использование инструментов, таких как scapy, позволяет специалистам

оперативно выявлять аномалии и потенциальные угрозы в реальном времени. Данная статья актуальна для начинающих специалистов, так как предлагает доступный и практичный подход к изучению анализа трафика, что способствует развитию навыков мониторинга и защиты сетей.

1.2 Обзор исследований

В работе В.В. Рубцова стоит цель - разобрать программу Scapy и с ее помощью понять, как генерируются интернет-пакеты, проанализировать сетевые пакеты и протоколы передачи данных. В результате анализа сетевых пакетов выяснится преимущество протокола TCP над UDP. В данной статье мы познакомим вас с интересной программой, написанной на Python для анализа сетевых пакетов. [1]. В.Ю. Деарт анализирует основные перспективные подходы к классификации трафика в режиме реального времени для поддержания QoS в SDN-сетях методами ML, а также представлен сравнительный обзор наиболее выдающихся работ. [2]. В публикации В.А. Овчарова и О.М. Степанюк проанализированы актуальные угрозы сетевым ресурсам организаций при реализации удаленного несанкционированного доступа (НСД) в обход межсетевого экрана (МЭ), используя незащищенный DNS-протокол [3]. В трудах Е.В. Абросимова и А.В.Шнайдера рассмотрены проблемы обеспечения информационной безопасности в компьютерных сетях. [4]. А. Бражук погрузился в многофункциональный расширяемый генератор сетевых пакетов Scapy, работающий в интерпретируемой среде языка программирования Python [5].

1.3Цель исследования

Целью исследования является разработка и демонстрация простого инструмента для анализа сетевого трафика с использованием Python и библиотеки scapy, направленного на перехват и обработку TCP-пакетов с последующим извлечением IP-адресов источника и назначения. Работа направлена на изучение базовых принципов мониторинга сети, предоставление практического примера для начинающих специалистов по кибербезопасности и иллюстрацию применения анализа трафика для выявления потенциальных угроз в сетевой инфраструктуре.

2 Материалы и методы

В данном исследовании используется интерпретируемая среда Python 3.x для разработки скрипта анализа сетевого трафика. Применяется библиотека scapy для перехвата и обработки сетевых пакетов, а также стандартная библиотека logging для записи данных в файл.

2.1Подготовка

Необходимо установить Python версии 3.x на систему. Также понадобится библиотека scapy с помощью команды: `pip install scapy`. На операционной системе Windows дополнительно необходимо установить Npcap для поддержки перехвата пакетов. Следует создать новый файл с

именем `packet_sniffer.py` в текстовом редакторе или среде разработки, например, PyCharm или Visual Studio Code. И убедиться, что имеются права администратора для запуска скрипта, так как перехват сетевых пакетов требует повышенных привилегий. После подготовки среды необходимо создать проект (Рис. 1).



Рисунок 1 – Создание файла `packet_sniffer.py` в редакторе кода

3. Результаты

В процессе реализации инструмента для анализа сетевого трафика на языке Python первым шагом становится подготовка среды выполнения. На начальном этапе проекта производится импорт необходимых библиотек и настройка системы логирования. Для фиксации информации о перехваченных сетевых пакетах используется модуль `logging`, обеспечивающий запись данных в файл `traffic_log.txt` с указанием временных меток. Одновременно создаётся объект `Counter`, с помощью которого в дальнейшем осуществляется подсчёт количества пакетов, приходящихся на каждый IP-адрес источника. Инициализация этих компонентов происходит в начале работы программы и позволяет в дальнейшем накапливать статистику по сетевому трафику (рис. 2).

```
logging.basicConfig(  
    filename='traffic_log.txt',  
    level=logging.INFO,  
    format='%(asctime)s - %(levelname)s - %(message)s'  
)
```

Рисунок 2 - Инициализация модуля `logging` и объекта `Counter`

На следующем этапе внимание сосредотачивается на выборе сетевого интерфейса, через который будет осуществляться перехват трафика. Для этого реализуется функция `select_interface()`, выполняющая сбор доступных интерфейсов с помощью метода `get_if_list()`. Полученные данные отображаются пользователю в удобной форме: с указанием IP-адреса либо пометкой "Неактивен", если адрес отсутствует. Пользовательский ввод используется для выбора конкретного интерфейса. Дополнительно проводится проверка корректности выбора: исключаются интерфейсы с адресом 0.0.0.0, а также те, что относятся к классу `loopback`. При отсутствии подходящих интерфейсов программа завершает выполнение, предварительно зафиксировав ошибку как в лог-файле, так и в консоли (Рис. 3).

```
def select_interface(): usage
    """Выбор активного сетевого интерфейса."""
    interfaces = get_if_list()
    if not interfaces:
        error_msg = "Не найдено сетевых интерфейсов. Убедитесь, что Npcap установлен (Windows) или сетевое подключение активно."
        print(error_msg)
        logging.error(error_msg)
        sys.exit(1)

    print("Доступные сетевые интерфейсы:")
    active_interfaces = []
    for i, iface in enumerate(interfaces, 1):
        try:
            ip_addr = get_if_addr(iface)
            status = f"IP: {ip_addr}" if ip_addr and ip_addr != '0.0.0.0' else "Неактивен"
            if "Loopback" in iface:
                status += " (Loopback, не подходит для внешнего трафика)"
            active_interfaces.append((i, iface, ip_addr))
            print(f"{i}. {iface} ({status})")
        except Exception as e:
            print(f"{i}. {iface} (Ошибка определения IP: {e})")

    if not active_interfaces:
        error_msg = "Не найдено активных интерфейсов с IP-адресом."
        print(error_msg)
        logging.error(error_msg)
        sys.exit(1)
```

Рисунок 3 – Функция select_interface()

Ключевая часть проекта связана с обработкой перехваченных пакетов. Эта задача реализуется в функции `packet_callback`, которая вызывается для каждого сетевого пакета, поступающего на обработку. На этом этапе происходит фильтрация пакетов: анализируются только те из них, которые содержат слои IP и TCP. Из полученного пакета извлекаются IP-адреса отправителя и получателя, порты, а также иные атрибуты, такие как протокол и размер. Информация отображается в консоли и одновременно сохраняется в файл журнала. Также обновляется статистика по IP-адресам источника — соответствующий счётчик увеличивается. В дополнение к этому реализовано определение HTTP-пакетов по порту 80, для которых формируется специальная метка. Пакеты, не удовлетворяющие условиям (например, не содержащие слоёв IP или TCP), отбрасываются как нерелевантные, что позволяет снизить объём шумовых данных (рис. 4).

```

75 packet_callback(packet):
76     """Обработка каждого перехваченного пакета."""
77     if packet is None:
78         logging.warning("Получен пустой пакет (None). Пропускаем.")
79         return
80     try:
81         if packet.haslayer('IP'):
82             ip_src = packet['IP'].src
83             ip_dst = packet['IP'].dst
84             packet_size = len(packet) if packet else 0
85             protocol = packet['IP'].proto
86             proto_name = "TCP" if protocol == 6 else "UDP" if protocol == 17 else f"Протокол {protocol}"
87             timestamp = time.strftime(format: "%Y-%m-%d %H:%M:%S", time.localtime())
88
89             if packet.haslayer('TCP'):
90                 src_port = packet['TCP'].sport
91                 dst_port = packet['TCP'].dport
92                 tcp_flags = packet['TCP'].flags
93                 is_http = dst_port == 80 or src_port == 80
94                 is_https = dst_port == 443 or src_port == 443
95                 protocol_label = " (HTTP)" if is_http else " (HTTPS)" if is_https else ""
96                 log_message = f"[{timestamp}] Пакет: {ip_src}:{src_port} -> {ip_dst}:{dst_port}{protocol_label} | Флаги TCP: {tcp_flags} | Размер: {packet_size} байт"
97             else:
98                 log_message = f"[{timestamp}] Пакет: {ip_src} -> {ip_dst} | Протокол: {proto_name} | Размер: {packet_size} байт"
99
100         print(log_message)
101         logging.info(log_message)
102         ip_counter[ip_src] = ip_counter.get(ip_src, 0) + 1
103     except Exception as e:
104         logging.warning(f"Пакет без IP-слоя: {packet.summary()}")
105     logging.error(f"Ошибка при обработке пакета: {e}")
106     print(f"Ошибка при обработке пакета: {e}")

```

Рисунок 4 – Реализация функции packet_callback

Организация логики запуска приложения сосредоточена в функции main, которая инициирует процесс перехвата пакетов с использованием функции sniff из библиотеки scapy. На данном этапе задаются условия фильтрации трафика — по умолчанию перехватываются TCP-пакеты. Также может быть установлено ограничение на количество анализируемых пакетов, что удобно для демонстрационных целей. Для каждого перехваченного пакета вызывается функция обратного вызова, обеспечивающая его анализ (рис. 5).

```

def main():
    """Основная функция для перехвата и анализа пакетов."""
    try:
        # Выбор сетевого интерфейса
        iface = select_interface()
        ip_addr = get_if_addr(iface)
        print(f"Используемый интерфейс: {iface} (IP: {ip_addr})")
        logging.info(f"Запуск на интерфейсе: {iface} (IP: {ip_addr})")

        # Выбор режима перехвата
        print("Выберите режим перехвата:")
        print("1. Все пакеты (для отладки)")
        print("2. Только TCP (порты 80 и 443 для HTTP/HTTPS)")
        mode = input("Введите номер режима (1-2): ")
        filter_str = None if mode == "1" else "tcp and (port 80 or port 443)"

        print(f"Запуск перехвата (режим: {'Все пакеты' if mode == '1' else 'TCP HTTP/HTTPS'})... Нажмите Ctrl+C для остановки.")
        sniff(
            iface=iface,
            filter=filter_str,
            prn=packet_callback
        )
    except KeyboardInterrupt:
        print("\nЗавершено.")

```

Рисунок 5 – Реализация функции main и запуск перехвата

Важным аспектом надёжности работы программы является обработка исключительных ситуаций. Среди возможных проблем — запуск скрипта без

административных прав, ошибки при доступе к сетевым интерфейсам или отсутствие необходимых компонентов, таких как Npcap в операционной системе Windows. Для каждой из таких ситуаций предусмотрена система уведомлений пользователя: выводятся сообщения с пояснениями и рекомендациями, фиксируются соответствующие записи в лог-файле (рис. 6).

```
except PermissionError:
    error_msg = "Ошибка: Запустите скрипт с правами администратора (например, от имени администратора на Windows)."
    print(error_msg)
    logging.error(error_msg)
    sys.exit(1)
except Exception as e:
    error_msg = f"Ошибка при перехвате пакетов: {e}"
    print(error_msg)
    logging.error(error_msg)
    sys.exit(1)
except KeyboardInterrupt:
    print("\nПерехват остановлен пользователем.")
    print("Статистика по IP-адресам источников:")
    for ip, count in ip_counter.items():
        stats_message = f"IP: {ip} - {count} пакетов"
        print(stats_message)
        logging.info(stats_message)
```

Рисунок 6 – Реализация обработки ошибок

После запуска программы пользователь начинает работу с выбором сетевого интерфейса. На экране отображается список доступных интерфейсов с сопутствующей информацией — IP-адресами или статусом активности. Интерфейсы типа loopback отфильтровываются как не пригодные для анализа внешнего сетевого трафика. Следующим шагом становится выбор режима перехвата. Программа предлагает два варианта: первый предназначен для захвата всех пакетов и может использоваться, например, при отладке, второй — ограничивает анализ только TCP-пакетами на портах 80 и 443, что соответствует HTTP- и HTTPS-протоколам соответственно. Выбранный режим определяет фильтр, используемый при захвате данных (рис. 7).

```

Доступные сетевые интерфейсы:
1. \Device\NPF_{BE61ADAB-8F30-42D3-B077-038DD2A9E513} (Неактивен)
2. \Device\NPF_{D26864F1-E6C1-403E-8FE9-027C5AADD5DF} (Неактивен)
3. \Device\NPF_{151EC62C-429A-4DD5-AB2F-9C01355594E3} (Неактивен)
4. \Device\NPF_{36541355-175A-46D4-B83C-3FD8730B8993} (IP: 10.254.6.96)
5. \Device\NPF_{83CD721B-0EA1-41B1-BFB6-57B5E39C8AFD} (IP: 192.168.37.1)
6. \Device\NPF_{2430C2A2-B915-4BD5-A4DB-26A039EC4F1F} (IP: 192.168.147.1)
7. \Device\NPF_{83403816-6C7D-4F3D-80C8-81D78F634F25} (IP: 169.254.32.244)
8. \Device\NPF_{E45F5869-BF79-4716-8534-493F706636E2} (IP: 169.254.141.199)
9. \Device\NPF_{24515C30-83D9-194B-42BD-28DB49F7A0FC} (IP: 10.160.69.181)
10. \Device\NPF_{Loopback} (IP: 127.0.0.1 (Loopback, не подходит для внешнего трафика))

Выберите интерфейс для перехвата (рекомендуется интерфейс с IP, например, Wi-Fi или Ethernet):
Введите номер интерфейса (1-10): 4
Используемый интерфейс: \Device\NPF_{36541355-175A-46D4-B83C-3FD8730B8993} (IP: 10.254.6.96)
Выберите режим перехвата:
1. Все пакеты (для отладки)
2. Только TCP (порты 80 и 443 для HTTP/HTTPS)
Введите номер режима (1-2): 1

```

Рисунок 7 – Запуск программы

Начав перехват, программа переходит к отображению поступающих пакетов. Для каждого из них в консоли и лог-файле указывается подробная информация: временная метка, IP-адреса источника и получателя, номера портов, используемый протокол, размер передаваемых данных, а при необходимости — и метка, указывающая на тип трафика (HTTP/HTTPS). Сформированный файл `traffic_log.txt` может использоваться для последующего анализа сетевой активности (рис. 8).

```

2025-07-07 15:43:29,426 - WARNING - Пакет без IP-слоя: Ether / IPv6 / UDP / mDNS Qry b'googlecast._tcp.local.'
2025-07-07 15:43:29,901 - INFO - [2025-07-07 15:43:29] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 282 байт
2025-07-07 15:43:30,138 - INFO - [2025-07-07 15:43:30] Пакет: 195.26.224.120 -> 10.254.6.96 | Протокол: UDP | Размер: 234 байт
2025-07-07 15:43:30,181 - INFO - [2025-07-07 15:43:30] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 148 байт
2025-07-07 15:43:32,114 - INFO - [2025-07-07 15:43:32] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 250 байт
2025-07-07 15:43:32,355 - INFO - [2025-07-07 15:43:32] Пакет: 195.26.224.120 -> 10.254.6.96 | Протокол: UDP | Размер: 234 байт
2025-07-07 15:43:32,407 - INFO - [2025-07-07 15:43:32] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 151 байт
2025-07-07 15:43:33,952 - INFO - [2025-07-07 15:43:33] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 298 байт
2025-07-07 15:43:34,179 - INFO - [2025-07-07 15:43:34] Пакет: 195.26.224.120 -> 10.254.6.96 | Протокол: UDP | Размер: 234 байт
2025-07-07 15:43:34,228 - INFO - [2025-07-07 15:43:34] Пакет: 10.254.6.96 -> 195.26.224.120 | Протокол: UDP | Размер: 200 байт

```

Рисунок 8 – Файл `traffic_log`

4 Выводы

Выполненная работа по созданию инструмента для перехвата и анализа сетевого трафика средствами языка Python и библиотеки Scapy продемонстрировала эффективность предложенного подхода. В рамках проекта была успешно реализована логика взаимодействия с сетевыми интерфейсами, настройки фильтров, обработки поступающих пакетов и логирования результатов. Использование удобного синтаксиса Python и функциональных возможностей Scapy позволило обеспечить стабильную и расширяемую архитектуру, пригодную для последующего усовершенствования и адаптации под более сложные задачи анализа сетевой безопасности.

Библиографический список

1. Рубцов В. В. и др. Генератор трафика. Создание пакетов с соединением TCP на примере программы Scapy //Перспективы науки. 2021. №. 7. С. 18-22.
2. Деарт В. Ю., Маньков В. А., Краснова И. А. Анализ перспективных подходов и исследований по классификации потоков трафика для поддержания QoS методами ML в SDN-сетях //Вестник СибГУТИ. 2021. №. 1 (53). С. 3-22.
3. Овчаров В. А., Степанюк О. М., Подшибякин А. С. Методика аудита объектов информационной инфраструктуры с использованием технологии DNS-туннелирования //Известия Тульского государственного университета. Технические науки. 2022. №. 7. С. 46-56.
4. Абросимов Е. В., Шнайдер А. В., Казаков Ф. А. Технологии интеллектуального анализа сетевого трафика для обеспечения противодействия несанкционированного доступа // Актуальные научные исследования сборник статей III Международной научно-практической конференции. В 2 частях. Том Часть 1. Пенза, 2021
5. Бражук А. Утилита scapy. Основы конструирования сетевых пакетов //Системный администратор. 2011. №. 10. С. 66-69.