УДК 004.42

РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРОСТОГО ФАЕРВОЛА НА PYTHON ДЛЯ ЗАЩИТЫ ЛОКАЛЬНОГО ХОСТА

Кулдаулет Д.Д.
kuldauletd@gmail.com
Магистрант 2 курса образовательной программы «Информатика»
Актюбинский региональный университет имени К. Жубанова
г.Актобе, Республика Казахстан
Научный руководитель –**Шангытбаева Г.А.**
доцент, PhD

Аннотация

В данной статье рассматривается разработка простого фаервола на языке Python, который позволяет контролировать сетевые соединения локального хоста. Представлен анализ существующих инструментов для мониторинга и фильтрации сетевого трафика, а также приведён код для реализации базового фаервола. Проведены тестирования работы программы в среде Windows с использованием библиотеки Scapy и команд iptables/netsh. Особое внимание уделено вопросам эффективности фильтрации и возможностям дальнейшего развития проекта.

Ключевые слова: фаервол, Python, сетевая безопасность, мониторинг трафика, блокировка IP-адресов, Scapy, программные фаерволы.

Сетевые угрозы продолжают развиваться, и защита персональных устройств становится всё более актуальной задачей. В условиях постоянных киберугроз важной задачей является защита локального хоста от нежелательных соединений. Программные фаерволы играют ключевую роль в обеспечении безопасности, позволяя контролировать исходящий и входящий трафик. В данной работе предложена простая реализация фаервола на Python, которая позволяет мониторить активные соединения, перехватывать сетевые пакеты и блокировать доступ к определённым IP-адресам.

Фаерволы — это средства фильтрации сетевого трафика, предотвращающие несанкционированный доступ к системам. [1] Они могут работать на разных уровнях модели OSI, фильтруя трафик по IP-адресам, портам или содержимому пакетов. Основные типы фаерволов включают:

- Сетевые фаерволы – фильтруют пакеты на границе сети.
- Хостовые фаерволы – работают на уровне операционной системы устройства.
- Приложенческие фаерволы – анализируют трафик конкретных приложений.
- Гибридные фаерволы – совмещают несколько методов фильтрации.

В данной работе рассматривается программная реализация на уровне приложений, использующая средства Python для анализа и блокировки соединений. [2]

Для разработки фаервола были выбраны следующие инструменты:

- Psutil — для мониторинга сетевых соединений.
- Scapy — для перехвата и анализа пакетов.
- Iptables (Linux) / Netsh (Windows) — для блокировки IP-адресов.
- Npcap — драйвер для перехвата пакетов в Windows.

Применение этих инструментов позволяет создать базовый фаервол с возможностями мониторинга, анализа и блокировки нежелательного трафика.

Программа выполняет три основные функции:

1. Мониторинг соединений — отображает активные подключения.
2. Перехват пакетов — фиксирует исходящий и входящий трафик.
3. Блокировка IP — предотвращает соединения с указанными IP-адресами.

Пример кода для мониторинга соединений:

```
import psutil

def monitor_connections():
    print("Активные соединения:")
    for conn in psutil.net_connections():
        if conn.status == 'ESTABLISHED' and conn.raddr:
            print(f"{conn.laddr.ip}:{conn.laddr.port} -> {conn.raddr.ip}:{conn.raddr.port}")
```

Этот код позволяет отслеживать все установленные соединения и выводить информацию о них.

Пример кода для перехвата пакетов с использованием Scapy:

```
from scapy.all import sniff

def packet_sniffer(packet):
    print(f"Пакет: {packet.summary()}")

sniff(prn=packet_sniffer, store=0)
```

Этот код перехватывает пакеты и отображает их краткую информацию. Для работы в Windows необходимо установить Npcap.

Пример кода для блокировки IP-адресов:

```
import os

def block_ip(ip):
    if os.name == "nt":
        os.system(f"netsh advfirewall firewall add rule name='BlockIP' dir=out action=block remoteip={ip}")
    else:
        os.system(f"sudo iptables -A OUTPUT -d {ip} -j DROP")
    print(f"IP {ip} заблокирован!")
```

Этот код позволяет блокировать нежелательные IP-адреса с использованием встроенных системных команд.

Для проверки работоспособности фаервола было проведено несколько тестов:

1. Мониторинг активных соединений – запуск скрипта и проверка корректности вывода активных подключений.
2. Перехват сетевых пакетов – использование Scapy для анализа пакетов.
3. Блокировка IP – попытка установить соединение с заблокированным IP-адресом.

Результаты тестирования:

- При запуске функции мониторинга корректно отображались все активные соединения.
- Перехват пакетов работал стабильно после установки Npcap в Windows.
- Блокировка IP-адресов была успешно протестирована: после внесения IP в список блокировки соединение с этим адресом становилось невозможным.

Тестирование проводилось на Windows 10 и Ubuntu 22.04. В Windows использовалась утилита Netsh, в Ubuntu – Iptables. В обоих случаях фаервол успешно блокировал указанные IP-адреса и фиксировал сетевые соединения.

В ходе тестирования были выявлены следующие ключевые моменты:

- Мониторинг соединений позволяет в реальном времени отслеживать активные подключения.
- Перехват пакетов даёт возможность детально анализировать трафик.
- Блокировка IP-адресов оказалась эффективной, но требует административных прав. [3]

Эффективность фаервола на Python зависит от нескольких факторов, таких как

производительность системы, используемые библиотеки и алгоритмы фильтрации, а также возможности для масштабирования и адаптации к различным требованиям безопасности. В рамках этого анализа мы сосредоточимся на нескольких ключевых аспектах: производительности, безопасности, возможности масштабирования и гибкости, а также на области потенциальных улучшений и оптимизаций.

Одной из важнейших характеристик любого фаервола является его производительность, которая напрямую зависит от того, как быстро он может обрабатывать сетевые пакеты и как эффективно фильтровать трафик. Рассмотрим основные параметры производительности:

Когда фаервол реализован с использованием Python, важно учитывать его влияние на ресурсы системы, такие как CPU и память. Python, несмотря на свою удобность и широкую функциональность, не является самым быстрым языком с точки зрения обработки трафика в реальном времени. Это особенно важно при перехвате большого объема пакетов или при мониторинге большого числа активных соединений.

Для тестирования влияния фаервола на ресурсы можно измерить использование процессора и памяти в процессе работы программы. При увеличении числа активных соединений или при перехвате большего объема пакетов Python-программа может начать замедляться. Это можно протестировать с помощью инструментов, таких как **psutil**, который позволяет отслеживать использование ресурсов в реальном времени. [4]

Пример кода для мониторинга загрузки процессора:

```
import psutil
import time
def monitor_cpu_usage(interval=1):
    while True:
        cpu_usage = psutil.cpu_percent(interval=interval)
        print(f"CPU Usage: {cpu_usage}%")
        time.sleep(1)
```

Пример тестирования ресурсоемкости можно провести путем запуска фаервола в условиях, когда активно используются несколько соединений и перехватываются пакеты. В этом случае можно наблюдать, как меняется нагрузка на процессор и память, а также оценить, насколько стабильна работа фаервола при высокой нагрузке.

Python не оптимизирован для обработки сетевых пакетов в реальном времени, особенно при высоких нагрузках. Важно понять, как быстро фаервол может перехватывать и фильтровать трафик. Библиотека **Scapy** удобна, но она не предоставляет максимально быстрых решений для обработки большого потока пакетов. В случае с высокоскоростными сетями или большими объемами данных фаервол может не успевать обрабатывать пакеты своевременно, что может привести к задержкам.

Для улучшения этой ситуации можно рассмотреть возможность использования C-расширений или других оптимизированных библиотек, таких как **dpkt** или **pcapy**, которые могут быть быстрее в обработке сетевого трафика.

Кроме обработки трафика на уровне CPU и памяти, фаервол также может влиять на сетевую задержку. В процессе фильтрации пакетов фаервол должен анализировать их содержимое, что может занимать некоторое время. На практике это может привести к небольшой задержке, особенно при фильтрации сложных пакетов или при блокировке большого количества IP-адресов. Эти задержки могут быть критичны в ситуациях, где требуется минимальное время отклика, например, в онлайн-играх или при видеоконференциях.

Уровень безопасности

Фаервол, реализованный на Python, имеет несколько уязвимостей и ограничений с точки зрения безопасности, которые могут повлиять на его эффективность как средства защиты от атак. [5]

Сам Python как язык программирования имеет несколько особенностей, которые

могут ограничить уровень безопасности фаервола. Во-первых, Python-программы могут быть уязвимы к атакам, связанным с выполнением вредоносных скриптов или эксплуатации библиотек с уязвимостями. Например, библиотека **Scapy**, хотя и является мощным инструментом для работы с сетевыми пакетами, может быть использована для создания фишинговых атак или атак типа Man-in-the-Middle (MITM), если она не будет правильно настроена. [6]

Во-вторых, использование Python для фильтрации пакетов может открывать двери для атак через уязвимости в самом коде программы. Для повышения уровня безопасности можно использовать более безопасные решения, такие как ядра фаерволов на уровне операционной системы или специализированные программы, которые обеспечивают защиту от атак более эффективно.

Кроме того, работа с фаерволом на уровне Python подразумевает, что программа должна иметь соответствующие привилегии на операционной системе, чтобы вносить изменения в настройки сетевых интерфейсов или фильтровать пакеты. Например, на Windows необходимо наличие прав администратора для использования утилиты Netsh. На Linux – для работы с iptables. При отсутствии этих прав фаервол не будет эффективно блокировать трафик, и пользователи будут уязвимы.

Система фаервола на Python, даже при хорошем мониторинге и блокировке IP-адресов, может быть уязвима к более сложным методам обхода. Например, если злоумышленник использует прокси-сервера, Tor или VPN, он может скрыть свой настоящий IP-адрес, что делает блокировку менее эффективной. Для повышения уровня защиты необходимо будет интегрировать фаервол с более сложными системами безопасности, например, с системами IDS/IPS (Intrusion Detection and Prevention Systems), которые позволяют более эффективно выявлять аномалии в трафике.

Одним из важнейших факторов в оценке эффективности фаервола является его способность масштабироваться в зависимости от нужд пользователя. В нашем случае фаервол на Python подходит для защиты локального хоста, но для использования в корпоративных или распределенных сетях ему не хватает масштабируемости.

Когда речь идет о защите большого количества устройств или сложных сетевых инфраструктур, фаервол, работающий на Python, может не быть достаточно масштабируемым. Это связано с тем, что Python, несмотря на свою гибкость, не оптимален для работы с большим объемом трафика, и обработка пакетов может существенно замедляться при увеличении нагрузки. Для этого можно рассмотреть возможность использования многозадачности или многопоточности для обработки пакетов.

Фаервол на Python, в его текущей реализации, ориентирован на работу в локальных сетях. Для работы в облачных средах или с виртуализированными инфраструктурами потребуются дополнительные решения, такие как интеграция с облачными провайдерами для контроля трафика через их фаерволы или использование виртуальных частных сетей для защиты между виртуальными машинами.

Для повышения производительности фаервола можно использовать оптимизированные библиотеки для обработки сетевых пакетов, такие как **dpkt** или **pcapy**. Кроме того, можно рассмотреть использование многопоточности для обработки пакетов, что позволит более эффективно распределять нагрузку между ядрами процессора. [7]

Для повышения уровня безопасности и расширения функциональности фаервола можно интегрировать его с другими инструментами безопасности, такими как системы обнаружения вторжений (IDS) или системы предотвращения атак (IPS). Это позволит фаерволу не только блокировать пакеты, но и анализировать трафик на наличие подозрительных действий.

Для удобства пользователей можно разработать графический интерфейс, который будет показывать статистику по активным соединениям, заблокированным IP-адресам и событиям фильтрации. Такой интерфейс позволит не только повысить удобство управления фаерволом, но и улучшить восприятие программы.

Заключение

Разработка простого фаервола на языке Python для защиты локального хоста продемонстрировала основные принципы работы с фильтрацией сетевого трафика, мониторингом активных соединений и блокировкой IP-адресов. Реализованный фаервол предоставляет пользователю возможности для контроля исходящего и входящего трафика, использования различных инструментов и библиотек для фильтрации данных, таких как Psutil и Scapy, и может служить базовым средством защиты от несанкционированных сетевых соединений.

Основные выводы:

Производительность и ресурсоемкость: Несмотря на удобство Python для быстрой разработки и гибкости, данный язык не является оптимальным для работы с большими объемами сетевого трафика в реальном времени. Использование Python-программ для фильтрации пакетов может быть ограничено по производительности при высоких нагрузках. Тестирование показало, что фаервол справляется с мониторингом и блокировкой соединений в средних условиях, но в случае интенсивного трафика потребуется дополнительная оптимизация и, возможно, использование других более производительных решений.

Несмотря на удобство и гибкость, Python имеет несколько ограничений с точки зрения безопасности. Основные уязвимости связаны с самим языком и его библиотеками, которые могут быть использованы в целях обхода фильтрации или для создания атак. Необходимо учитывать, что Python-программа для фильтрации пакетов может требовать администраторских прав и быть уязвимой для атак на уровне операционной системы.

Масштабируемость и гибкость: В рамках защиты локального хоста фаервол демонстрирует удовлетворительную работу. Однако для использования в корпоративных сетях или в крупных распределенных системах необходимы более сложные решения, которые смогут справиться с более высокими требованиями к масштабируемости и обработке большого объема трафика.

Потенциальные улучшения: На основе проведенного тестирования и анализа можно выделить несколько направлений для улучшения фаервола:

- Оптимизация производительности с использованием многозадачности или многопоточности.
- Использование более эффективных библиотек, таких как dpkt или pcapy, для обработки пакетов.
- Разработка графического интерфейса для упрощения взаимодействия с пользователем.
- Интеграция с другими инструментами безопасности, такими как IDS/IPS системы, для расширения возможностей фаервола.

Простой фаервол, разработанный на Python, является хорошим начальным шагом в создании системы для защиты локальных устройств, однако для более серьезных задач безопасности потребуется значительно более сложная и высокопроизводительная система. В будущем можно рассмотреть возможность интеграции с облачными решениями и виртуальными частными сетями, а также разработку более универсальных и масштабируемых решений для защиты в крупных инфраструктурах.

Таким образом, созданный фаервол на Python выполняет свою задачу, но его возможности ограничены, и для более эффективной защиты в условиях реального использования необходимо продолжать оптимизировать и развивать его функционал.

Список использованной литературы

1. Strand, L. K. (2004). Adaptive distributed firewall using intrusion detection (Master's thesis).
2. Ziade, T. (2017). Python Microservices Development: Build, test, deploy, and scale

microservices in Python. Packt Publishing Ltd.

3. A. -D. Tudosi, "A Python-Based Approach for Monitoring and Troubleshooting Snort IDS in Distributed Firewall Environments," 2023 International Conference on Advanced Scientific Computing (ICASC), Cluj-Napoca, Romania, 2023, pp. 1-5, doi: 10.1109/ICASC58845.2023.10328028. keywords: {Firewalls (computing);Scientific computing;Scalability;Network security;Reliability engineering;Libraries;Servers;Distributed firewall;Intrusion Detection System (IDS);Monitoring;Script-based framework;Snort},

4. Lee, B. (2019). Building a secure network test environment using virtual machines.

5. Rayhan, A., Kinzler, R., & Rayhan, R. CYBERSECURITY BEST PRACTICES FOR PYTHON WEB APPLICATIONS.

6. Rehim, R. (2016). Effective python penetration testing. Packt Publishing Ltd.

7. Elghaly, Y. (2021). Learn Penetration Testing with Python 3. x: Perform Offensive Pentesting and Prepare Red Teaming to Prevent Network Attacks and Web Vulnerabilities (English Edition). BPB Publications.