Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЁТ

к лабораторной работе №4

на тему

**ЗАЩИТА ОТ АТАКИ ПРИ УСТАНОВКЕ TCP-СОЕДИНЕНИЯ И ПРОТОКОЛОВ ПРИКЛАДНОГО УРОВНЯ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Герчик А.В. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc189743996)

[2 Реализация программного средства 4](#_Toc189743997)

[Заключение 6](#_Toc189743998)

[Список использованных источников 7](#_Toc189743999)

[Приложение А (обязательное) Исходный код программы 8](#_Toc189744000)

**1 ПОСТАНОВКА ЗАДАЧИ**

В рамках данной лабораторной работы необходимо реализовать сервер, который будет защищен от атак при установке *TCP* соединения [1] и в соответствии с заданным протоколом прикладного уровня. Основной целью работы является проработка методов обеспечения безопасности коммуникаций в сети, а также изучение влияния различных атак на процесс установления соединения. В условиях современного интернета, где количество кибератак продолжает расти, особую значимость приобретает защита серверов от угроз, связанных с несанкционированным доступом и злоупотреблениями.

Для достижения этой цели потребуется разработать серверное приложение, которое будет обрабатывать входящие соединения с учетом современных стандартов безопасности. Важными аспектами реализации являются механизмы аутентификации пользователей, шифрование данных, а также внедрение средств защиты от атак, таких как *SYN*-флуд [2]. Это позволит создать надежный сервер, который не только будет выполнять свои основные функции, но и эффективно защищаться от потенциальных угроз.

Использование средств программирования высокого уровня, таких как *Python* или *Java*, обеспечит удобство и гибкость в разработке серверной логики. При этом необходимо учитывать особенности выбранного языка, чтобы оптимизировать производительность и безопасность приложения. Также потребуется интеграция с сетевыми утилитами операционной системы, что позволит проводить мониторинг состояния сервера и анализировать сетевой трафик в реальном времени.

В качестве инструмента для анализа сетевого трафика рекомендуется использовать *WireShark*. Этот мощный инструмент позволит наблюдать за процессом установления соединений, выявлять подозрительные активности и анализировать пакеты данных. С помощью *WireShark* можно будет детально рассмотреть, как сервер реагирует на различные типы атак, а также оценить эффективность внедренных защитных механизмов [3].

Параллельно с разработкой сервера будет проводиться тестирование на устойчивость к различным видам атак. Важно не только выявить уязвимости, но и предложить решения для их устранения. Для этого могут быть использованы различные сценарии атак, включая *SYN*-флуд, *DDoS*-атаки, а также попытки подмены или перехвата данных. Результаты тестирования помогут лучше понять механизмы защиты и улучшить навыки работы с сетевыми протоколами.

**2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

В данной лабораторной работе был разработан *TCP*-сервер, реализованный с использованием языка программирования *Python* и сетевых функций операционной системы *Linux*. Сервер предназначен для обработки входящих соединений от клиентов, обеспечивая базовую функциональность с элементами безопасности и многопоточности. Важной частью реализации является использование механизма *SYN* *cookies*, что позволяет защитить сервер от атак типа *SYN*-флуд.

Сервер использует библиотеку «*socket*», что позволяет работать с низкоуровневыми сетевыми протоколами. Основные параметры сервера, такие как адрес и порт, задаются в начале кода. В данном случае сервер настроен на прослушивание всех интерфейсов («0.0.0.0») и использует порт «12345», что делает его доступным для внешних подключений.

Для обеспечения одновременной обработки нескольких клиентов сервер реализует многопоточность с помощью модуля «*threading*». Максимальное количество одновременных соединений ограничено значением «*MAX\_CONNECTIONS*», которое равно 5. Это позволяет избежать перегрузки сервера и обеспечивает более стабильную работу при высоких нагрузках.

Каждое новое подключение обрабатывается в отдельном потоке, что позволяет серверу продолжать принимать новые соединения, пока обрабатываются уже установленные. Функция «*handle\_client*» отвечает за взаимодействие с клиентом: она принимает данные от клиента, выводит их на экран и отправляет обратно в виде эхо-ответа (рисунок 2.1). В случае возникновения таймаута или других ошибок, соединение корректно закрывается, и информация об этом выводится в лог.

Код сервера включает обработку исключений, что позволяет отлавливать возможные ошибки и предупреждать о проблемах с подключением. Например, если таймаут соединения превышен, сервер выводит предупреждение, а при других ошибках сообщает о них с указанием адреса клиента.

Одним из ключевых аспектов реализации является использование механизма *SYN cookies* (рисунок 2.2) [4]. Этот метод позволяет защитить сервер от атак типа *SYN*-флуд, при которых злоумышленник пытается исчерпать ресурсы сервера, отправляя большое количество запросов на установление соединения без завершения. При включении *SYN* *cookies* сервер генерирует специальный код в ответ на *SYN*-запросы, который позволяет ему проверить легитимность соединения, не занимая при этом ресурсы системы до завершения трехстороннего рукопожатия.

В дополнение к этому, сервер поддерживает динамическое управление активными соединениями: при превышении максимального количества подключений новые подключения будут закрываться с выводом соответствующего предупреждения. Серверный код завершается корректным закрытием сокета в случае завершения работы, что предотвращает утечки ресурсов.

В результате, реализованный сервер представляет собой простую, но эффективную систему для обработки клиентских запросов в условиях многопользовательской среды, с учетом основных принципов сетевой безопасности и защиты от атак. Использование *SYN* *cookies* добавляет дополнительный уровень защиты, что делает сервер более устойчивым к потенциальным угрозам.

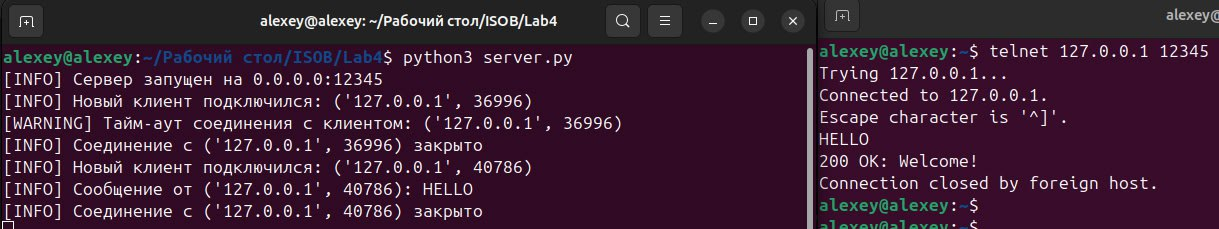


Рисунок 2.1 – Работа сервера

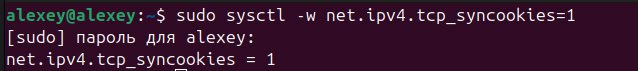


Рисунок 2.2 – Включение SYN cookies

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы был успешно разработан и реализован *TCP*-сервер на языке *Python* с использованием сетевых функций операционной системы *Linux*. Основной задачей стало создание надежного сервера, способного обрабатывать множественные соединения и защищенного от атак, особенно от *SYN*-флуда, с помощью механизма *SYN* *cookies*.

Сервер продемонстрировал способность эффективно управлять входящими запросами, обеспечивая многопоточность для обработки клиентов в реальном времени. Реализованные функции обработки данных, а также механизмы обработки ошибок и таймаутов, подтвердили свою эффективность и надежность в условиях тестирования.

Внедрение *SYN* *cookies* стало ключевым элементом в обеспечении безопасности сервера, что позволило предотвратить возможные атаки и повысить его устойчивость к перегрузкам. Это также подчеркнуло важность применения современных методов защиты в разработке сетевых приложений.

Полученные результаты подтвердили правильность выбранной архитектуры и подходов к реализации, что позволяет рекомендовать данный сервер как основу для дальнейших исследований и разработок в области сетевой безопасности. В будущем возможно расширение функциональности сервера, добавление новых методов защиты и оптимизация кода для повышения его производительности. Таким образом, проведенная работа не только помогла закрепить теоретические знания, но и предоставила практический опыт в разработке надежных и безопасных сетевых приложений.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Протоколы семейства TCP/IP. Теория и практика [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/ruvds/articles/759988/.

[2] Устройство TCP/Реализация SYN-flood атаки [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/782728/.

[3] Wireshark – подробное руководство по началу использования [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/735866/.

[4] Настройка сетевого стека Linux для высоконагруженных систем [Электронный ресурс]. – Режим доступа: https://habr.com/ru/companies/otus/ar ticles/550820/.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

import socket

import threading

HOST = '0.0.0.0' # Слушать все интерфейсы

PORT = 12345 # Порт сервера

MAX\_CONNECTIONS = 5 # Максимальное количество одновременных соединений

active\_connections = []

def handle\_client(client\_socket, client\_address):

"""

Обработка подключения клиента.

"""

print(f"[INFO] Подключен клиент: {client\_address}")

try:

client\_socket.settimeout(10)

while True:

data = client\_socket.recv(1024)

if not data:

break

print(f"[{client\_address}] Получено: {data.decode('utf-8')}")

response = f"Echo: {data.decode('utf-8')}"

client\_socket.sendall(response.encode('utf-8'))

except socket.timeout:

print(f"[WARNING] Таймаут соединения с {client\_address}")

except Exception as e:

print(f"[ERROR] Ошибка при обработке клиента {client\_address}: {e}")

finally:

print(f"[INFO] Соединение с {client\_address} закрыто")

active\_connections.remove(client\_socket)

client\_socket.close()

def start\_server():

"""

Запуск TCP-сервера.

"""

global active\_connections

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(MAX\_CONNECTIONS)

print(f"[INFO] Сервер запущен на {HOST}:{PORT}")

try:

while True:

client\_socket, client\_address = server\_socket.accept()

if len(active\_connections) >= MAX\_CONNECTIONS:

print(f"[WARNING] Превышено максимальное количество соединений: {MAX\_CONNECTIONS}")

client\_socket.close()

continue

active\_connections.append(client\_socket)

client\_thread = threading.Thread(target=handle\_client, args=(client\_socket, client\_address))

client\_thread.daemon = True

client\_thread.start()

except KeyboardInterrupt:

print("[INFO] Сервер остановлен пользователем")

finally:

server\_socket.close()

if \_\_name\_\_ == "main":

start\_server()