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.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc190686916)

[1 ФОРМУЛИРОВКА ЗАДАЧИ 4](#_Toc190686917)

[2 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ 5](#_Toc190686918)

[ЗАКЛЮЧЕНИЕ 7](#_Toc190686919)

ВВЕДЕНИЕ

В современных распределённых системах обеспечение безопасности обмена данными является одной из ключевых задач. С ростом количества кибератак защита серверных приложений становится всё более актуальной. В данной работе разработано серверное приложение, способное не только устанавливать защищённое соединение с клиентами посредством специального механизма handshake, но и эффективно противостоять различным видам атак. Программа предназначена для демонстрации базовых принципов защиты, применяемых для фильтрации подозрительных соединений и предотвращения атак типа ACK Flood, Slowloris и SYN Flood.

В работе подробно рассматриваются этапы установления защищённого соединения: от генерации уникального токена сервером до проверки корректности ответа от клиента. Дополнительно реализованы механизмы обнаружения аномальной активности, позволяющие своевременно разрывать соединения с клиентами, пытающимися перегрузить сервер или использовать уязвимости в процедуре handshake. Такая архитектура позволяет повысить надёжность работы приложения и снизить риск несанкционированного доступа.

Также следует отметить, что выбранный подход демонстрирует возможности простых методов защиты, которые могут быть использованы в реальных приложениях для повышения устойчивости серверной инфраструктуры к кибератакам. Программа служит хорошей иллюстрацией того, как можно объединить проверку подлинности и мониторинг активности для создания защищённого сервера.

1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель работы состоит в разработке серверного приложения, которое обеспечивает установление защищённого соединения с клиентами посредством механизма handshake и реализует защитные меры для противодействия сетевым атакам.

Задача включает в себя следующие основные аспекты:

1 **Установление защищённого соединения.** При каждом новом подключении сервер генерирует уникальный токен и отправляет его клиенту в виде сообщения CHALLENGE. Клиент должен корректно ответить, отправив соответствующее сообщение RESPONSE с полученным токеном. При совпадении токенов соединение считается успешно установленным, и клиент получает доступ к дальнейшему функционалу сервера.

2 **Обнаружение аномальной активности.** Приложение должно отслеживать поведение клиентов на предмет подозрительной активности. Если клиент отправляет слишком большое количество команд ACK за короткий период времени (ACK Flood), если передача данных замедляется до уровня, приводящего к истечению таймаута (Slowloris), или если наблюдается массовое создание незавершённых handshake (SYN Flood), сервер должен немедленно разорвать соединение.

3 **Защита от атак.** Встроенные в сервер защитные механизмы позволят обнаруживать и блокировать атаки на ранней стадии, предотвращая возможное истощение ресурсов системы. Это достигается за счёт ограничения количества незавершённых соединений, контроля времени отклика клиента и анализа количества отправляемых команд.

Таким образом, итоговое решение должно обеспечить корректное установление защищённого соединения с клиентом, а также демонстрировать эффективность реализованных защитных механизмов при попытках проведения атак.

2 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

В данной главе представлена практическая демонстрация работы серверного приложения. Описаны ключевые этапы его функционирования, а также проиллюстрированы результаты тестирования защитных механизмов. При запуске серверного приложения в консоли выводятся отладочные сообщения, информирующие о каждом этапе работы. После старта сервер начинает ожидать входящих подключений.

При корректном выполнении процедуры handshake клиент получает сообщение **HANDSHAKE SUCCESS**, после чего сервер отправляет приветственное сообщение с перечнем доступных команд (например, PING, ACK, quit). Результаты успешного соединения можно увидеть **на рисунке 1.**

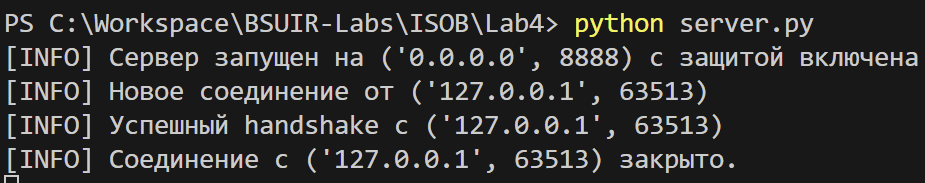


Рисунок 1 – Пример успешного соединения с сервером

На данном скриншоте из консоли показано, как клиент успешно проходит процедуру handshake: сервер отправляет сообщение **CHALLENGE**, клиент отвечает корректным сообщением **RESPONSE**, и сервер выводит сообщение **HANDSHAKE SUCCESS,** подтверждая установление защищённого соединения.

При моделировании ACK Flood атаки клиент за короткий промежуток времени отправляет множество команд **ACK**. Сервер, обнаружив превышение порога (ACK\_THRESHOLD), фиксирует аномалию и немедленно разрывает соединение. Результаты работы защитного механизма продемонстрированы на **рисунке 2.**

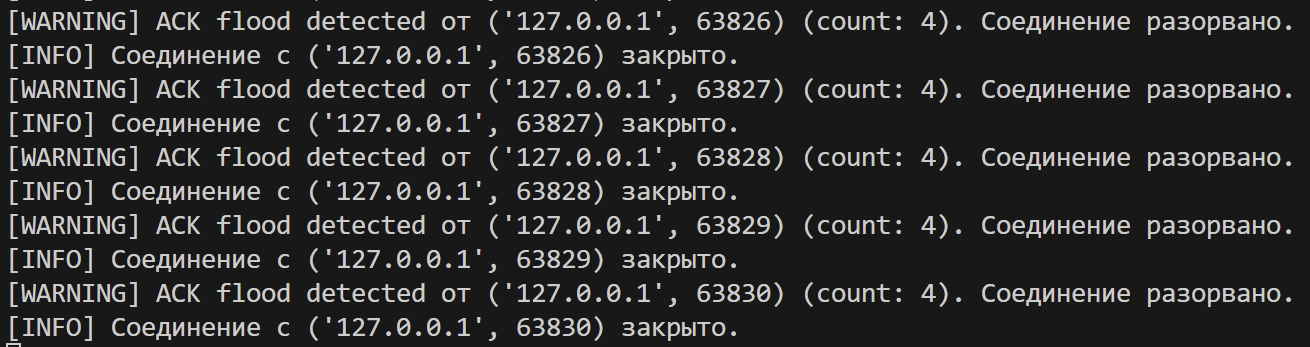


Рисунок 2 – Сценарий ACK Flood атаки

На данном скриншоте виден вывод отладочных сообщений, свидетельствующий о том, что сервер обнаружил избыточное количество команд **ACK** и принял решение разорвать соединение.

При моделировании Slowloris атаки клиент намеренно замедляет передачу ответа на **CHALLENGE**, отправляя данные по одному символу с большими интервалами. В результате истекает таймаут, и сервер завершает процедуру handshake. Результаты данного сценария можно увидеть **на рисунке 3.**

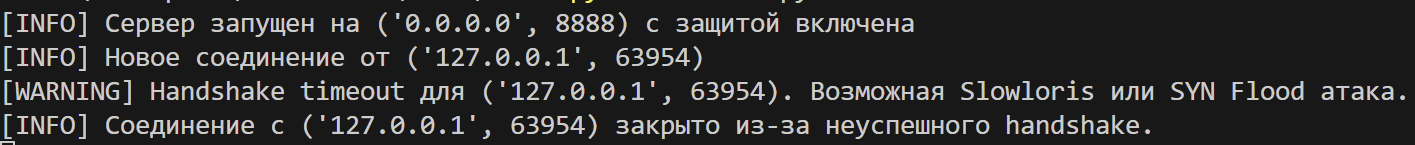


Рисунок 3 – Сценарий Slowloris атаки

Этот скриншот иллюстрирует, как истекает таймаут соединения из-за замедленной передачи данных. Сервер фиксирует превышение времени ожидания и завершает handshake, предотвращая возможность проведения атаки.

При моделировании SYN Flood атаки происходит массовое создание незавершённых handshake. Ограничение на количество таких соединений (MAX\_HALF\_OPEN\_CONNECTIONS) позволяет серверу обнаружить атаку и отклонить новые подключения. Вывод отладочных сообщений свидетельствует о том, что сервер блокирует входящие соединения при превышении допустимого лимита на рисунке 4.

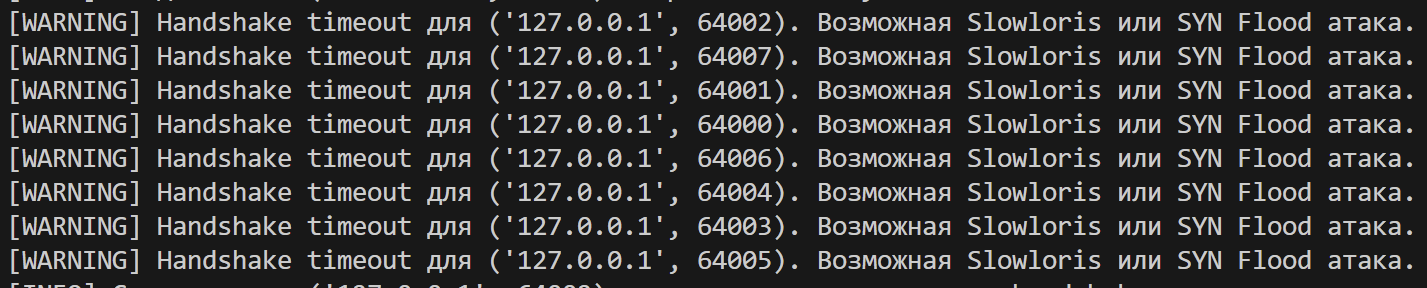


Рисунок 4 – Сценарий SYN Flood атаки

На данном скриншоте отображён вывод, подтверждающий, что сервер обнаружил чрезмерное количество незавершённых handshake и отказал в установлении новых соединений.

В результате тестирования было установлено, что при корректном поведении клиента сервер успешно устанавливает защищённое соединение и позволяет выполнять команды, а при обнаружении аномальной активности защитные механизмы срабатывают оперативно и без задержек.

# ЗАКЛЮЧЕНИЕ

В данной работе разработано серверное приложение, демонстрирующее возможность установления защищённого соединения с клиентом посредством механизма handshake и эффективной защиты от сетевых атак. Реализованная система успешно обеспечивает проверку подлинности клиента, генерируя уникальные токены и контролируя корректность его ответов. Дополнительно встроенные механизмы обнаружения атак типа ACK Flood, Slowloris и SYN Flood позволяют оперативно разрывать подозрительные соединения, что значительно повышает надёжность работы сервера.

Проведённое экспериментальное исследование показало, что при нормальной работе сервера клиенты получают доступ к функционалу, а попытки злоупотребления ресурсами приводят к немедленному разрыву соединения. Таким образом, предложенное решение демонстрирует практическую применимость простых, но эффективных методов защиты серверных приложений в условиях современных угроз.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг программного кода**

Листинг А.1 – файл *attack\_ack\_flood.py*

import asyncio

async def attack\_ack\_flood(i: int):

try:

reader, writer = await asyncio.open\_connection("127.0.0.1", 8888)

challenge = await asyncio.wait\_for(reader.readline(), timeout=5)

text = challenge.decode().strip()

if text.startswith("CHALLENGE"):

parts = text.split()

token = parts[1] if len(parts) > 1 else ""

writer.write(f"RESPONSE {token}\n".encode())

await writer.drain()

handshake\_result = await reader.readline()

print(f"[ACK Flood {i}] Handshake: {handshake\_result.decode().strip()}")

else:

print(f"[ACK Flood {i}] Received: {text}")

for j in range(10):

writer.write("ACK\n".encode())

await writer.drain()

await asyncio.sleep(0.01)

try:

resp = await asyncio.wait\_for(reader.readline(), timeout=1)

if not resp:

break

print(f"[ACK Flood {i}] Response: {resp.decode().strip()}")

except asyncio.TimeoutError:

break

writer.close()

await writer.wait\_closed()

except Exception as e:

print(f"[ACK Flood {i}] Ошибка: {e}")

async def main():

tasks = [asyncio.create\_task(attack\_ack\_flood(i)) for i in range(10)]

await asyncio.gather(\*tasks)

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

Листинг А.2 – файл *attack\_slowloris.py*

import asyncio

async def slowloris\_attack():

try:

reader, writer = await asyncio.open\_connection('127.0.0.1', 8888)

challenge = await asyncio.wait\_for(reader.readline(), timeout=5)

print(f"[Slowloris] Received: {challenge.decode().strip()}")

parts = challenge.decode().strip().split()

if len(parts) != 2 or parts[0] != "CHALLENGE":

print("[Slowloris] Неверный формат challenge.")

writer.close()

await writer.wait\_closed()

return

token = parts[1]

response = f"RESPONSE {token}\n"

print("[Slowloris] Отправляем ответ медленно...")

for ch in response:

writer.write(ch.encode())

await writer.drain()

await asyncio.sleep(1)

handshake\_result = await reader.readline()

print(f"[Slowloris] Handshake result: {handshake\_result.decode().strip()}")

await asyncio.sleep(10)

writer.close()

await writer.wait\_closed()

except Exception as e:

print(f"[Slowloris] Ошибка: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

asyncio.run(slowloris\_attack())

Листинг А.3 – файл *attack\_syn\_flood.py*

import asyncio

async def attack\_connection(i: int):

try:

reader, writer = await asyncio.open\_connection("127.0.0.1", 8888)

challenge = await asyncio.wait\_for(reader.readline(), timeout=5)

print(f"[SYN Flood {i}] Received: {challenge.decode().strip()}")

await asyncio.sleep(10)

writer.close()

await writer.wait\_closed()

except Exception as e:

print(f"[SYN Flood {i}] Ошибка: {e}")

async def main():

tasks = [asyncio.create\_task(attack\_connection(i)) for i in range(20)]

await asyncio.gather(\*tasks)

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

Листинг А.4 – файл *client\_correct.py*

import asyncio

async def proper\_client():

try:

reader, writer = await asyncio.open\_connection('127.0.0.1', 8888)

challenge = await asyncio.wait\_for(reader.readline(), timeout=5)

print(f"[Client] Received: {challenge.decode().strip()}")

if challenge.decode().startswith("CHALLENGE"):

token = challenge.decode().split()[1]

writer.write(f"RESPONSE {token}\n".encode())

await writer.drain()

handshake\_result = await reader.readline()

print(f"[Client] Handshake result: {handshake\_result.decode().strip()}")

else:

print("[Client] Нет handshake, защита отключена.")

welcome = await reader.readline()

print(f"[Client] {welcome.decode().strip()}")

writer.write("PING\n".encode())

await writer.drain()

pong = await reader.readline()

print(f"[Client] {pong.decode().strip()}")

writer.write("quit\n".encode())

await writer.drain()

goodbye = await reader.readline()

print(f"[Client] {goodbye.decode().strip()}")

writer.close()

await writer.wait\_closed()

except Exception as e:

print(f"[Client] Ошибка: {e}")

if \_\_name\_\_ == '\_\_main\_\_':

asyncio.run(proper\_client())

Листинг А.5 – файл *server.py*

import asyncio

import random

import string

import time

import argparse

import logging

logging.basicConfig(level=logging.INFO, format="[%(levelname)s] %(message)s")

parser = argparse.ArgumentParser(

description="Запуск защищённого сервера. По умолчанию защита включена."

)

parser.add\_argument(

"--disable-protection",

action="store\_true",

help="Запустить сервер без защитных механизмов.",

)

args = parser.parse\_args()

PROTECTION\_ENABLED = not args.disable\_protection

MAX\_HALF\_OPEN\_CONNECTIONS = 50

half\_open\_connections = set()

ACK\_THRESHOLD = 3

ACK\_WINDOW = 1

async def handle\_client(reader: asyncio.StreamReader, writer: asyncio.StreamWriter):

peername = writer.get\_extra\_info("peername")

logging.info(f"Новое соединение от {peername}")

if PROTECTION\_ENABLED and len(half\_open\_connections) >= MAX\_HALF\_OPEN\_CONNECTIONS:

writer.write("Server busy. Try again later.\n".encode())

await writer.drain()

writer.close()

await writer.wait\_closed()

logging.warning(

f"Отклонено соединение от {peername} (слишком много незавершённых handshake). Возможная SYN Flood атака."

)

return

handshake\_success = False

token = "".join(random.choices(string.ascii\_uppercase + string.digits, k=6))

if PROTECTION\_ENABLED:

try:

half\_open\_connections.add(peername)

writer.write(f"CHALLENGE {token}\n".encode())

await writer.drain()

response = await asyncio.wait\_for(reader.readline(), timeout=5.0)

response = response.decode().strip()

if response == f"RESPONSE {token}":

handshake\_success = True

writer.write("HANDSHAKE SUCCESS\n".encode())

await writer.drain()

logging.info(f"Успешный handshake с {peername}")

else:

writer.write("HANDSHAKE FAILED\n".encode())

await writer.drain()

logging.warning(

f"Неверный ответ от {peername}: {response}. Возможная попытка атаки."

)

except asyncio.TimeoutError:

writer.write("HANDSHAKE TIMEOUT\n".encode())

await writer.drain()

logging.warning(

f"Handshake timeout для {peername}. Возможная Slowloris или SYN Flood атака."

)

finally:

half\_open\_connections.discard(peername)

else:

handshake\_success = True

writer.write("Protection disabled. Connection accepted.\n".encode())

await writer.drain()

logging.info(

f"Защита отключена: соединение от {peername} принято без handshake"

)

if not handshake\_success:

writer.close()

await writer.wait\_closed()

logging.info(f"Соединение с {peername} закрыто из-за неуспешного handshake.")

return

writer.write(

"Добро пожаловать на защищённый сервер.\n"

"Доступные команды: PING, ACK, quit\n".encode()

)

await writer.drain()

ack\_count = 0

ack\_window\_start = time.monotonic()

while True:

try:

data = await asyncio.wait\_for(reader.readline(), timeout=30.0)

except asyncio.TimeoutError:

writer.write("Connection timed out due to inactivity.\n".encode())

await writer.drain()

logging.info(

f"Соединение с {peername} разорвано из-за таймаута бездействия."

)

break

if not data:

break

message = data.decode().strip()

if message.upper() == "ACK":

current\_time = time.monotonic()

if current\_time - ack\_window\_start > ACK\_WINDOW:

ack\_count = 0

ack\_window\_start = current\_time

ack\_count += 1

if ack\_count > ACK\_THRESHOLD:

writer.write("ACK flood detected. Connection dropped.\n".encode())

await writer.drain()

logging.warning(

f"ACK flood detected от {peername} (count: {ack\_count}). Соединение разорвано."

)

break

writer.write("ACK received.\n".encode())

await writer.drain()

continue

if message.lower() == "quit":

writer.write("Goodbye!\n".encode())

await writer.drain()

break

elif message.upper() == "PING":

writer.write("PONG\n".encode())

await writer.drain()

else:

writer.write(f"Unknown command: {message}\n".encode())

await writer.drain()

writer.close()

await writer.wait\_closed()

logging.info(f"Соединение с {peername} закрыто.")

async def main():

server = await asyncio.start\_server(handle\_client, "0.0.0.0", 8888)

addrs = ", ".join(str(sock.getsockname()) for sock in server.sockets)

mode = "включена" if PROTECTION\_ENABLED else "отключена"

logging.info(f"Сервер запущен на {addrs} с защитой {mode}")

async with server:

await server.serve\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

try:

asyncio.run(main())

except KeyboardInterrupt:

logging.info("Сервер остановлен.")