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.253501
Станишевский А.Д.

Проверил: ассистент кафедры информатики
Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

1 Цель работы 3
2 Теоретические сведения 4
3 Ход работы 5
Заключение 6
Приложение А (обязательное) Листинг программного кода 7

1 ЦЕЛЬ РАБОТЫ

Целью работы является исследование методов защиты TCP-соединений и протоколов прикладного уровня от сетевых атак, а также разработка демонстрационного программного решения, иллюстрирующего принципы противодействия угрозам на этапе установления соединения и обработки данных.

В рамках работы реализовано серверное приложение с поддержкой защищенного и незащищенного режимов, что позволяет проводить сравнительный анализ эффективности механизмов безопасности. Программа моделирует сценарии атак, такие как TCP-flood, DoS/DDoS и аномальные запросы прикладного уровня, и демонстрирует методы их нейтрализации, включая ограничение количества одновременных TCP-соединений, контроль частоты запросов от клиентов на основе IP-адресов, валидацию данных прикладного уровня.

Особое внимание уделено реализации интерактивного консольного интерфейса, предоставляющего возможность выбора режима работы, мониторинга состояния сервера (активные соединения, статусы клиентов) и визуализации процессов обработки запросов. В процессе разработки изучены уязвимости TCP/IP и прикладных протоколов.

В процессе разработки были изучены основные методы защиты TCP-соединений, включая использование таймаутов, лимитов на количество запросов. Реализация этих механизмов позволила глубже осознать особенности обеспечения безопасности сетевых сервисов, их преимущества, такие как устойчивость к атакам и контроль нагрузки, а также ограничения, например, возможное снижение производительности при включении защитных мер.

Таким образом, в ходе выполнения работы были получены практические навыки реализации и анализа защитных механизмов для TCP-соединений, а также создана программа, демонстрирующая базовые подходы к обеспечению безопасности сетевых приложений. Разработанное решение может служить основой для дальнейшего изучения более сложных методов защиты информации и современных протоколов сетевой безопасности.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Протокол TCP (Transmission Control Protocol) является одним из основных протоколов в сети Интернет. Он обеспечивает надежную передачу данных между устройствами, гарантируя, что все пакеты данных будут доставлены в правильном порядке и без ошибок. TCP был разработан в 1970-х годах и стал стандартом в сетевых коммуникациях благодаря своей способности управлять сложными сетевыми взаимодействиями.

TCP работает на транспортном уровне модели OSI (Open Systems Interconnection) и тесно связан с протоколом IP (Internet Protocol), образуя вместе стек TCP/IP. Основная задача TCP – обеспечить надежную, последовательную и управляемую передачу данных между приложениями. Это достигается за счет использования различных механизмов, таких как установление соединения, контроль ошибок и управление перегрузкой.

Защита от атак при установке TCP-соединения и при использовании протоколов прикладного уровня требует комплексного подхода, учитывающего особенности каждого этапа взаимодействия в сети. На этапе установки TCP-соединения одной из распространённых угроз является атака SYN-флуд, при которой злоумышленник отправляет множество запросов на подключение (SYN-пакетов), но не завершает установку соединения, переполняя очередь полуоткрытых подключений на сервере.

Чтобы установить надежное соединение, TCP использует процесс, называемый термином трехстороннее рукопожатие (TCP three-way/triple handshake). Установленное соединение будет полнодуплексным, то есть оба канала могут передавать информацию одновременно, а также они синхронизируют (SYN) и подтверждают (ACK) друг друга. Обмен выполняется следующим образом:

1. Клиент отправляет сегмент с установленным флагом SYN. При этом сегменту присваивается произвольный порядковый номер (sequence number) в интервале от 1 до 232 (т.н. initial sequence number), относительно которого будет вестись дальнейший отсчет последовательности сегментов в соединении.

2. Сервер получает запрос и отправляет ответный сегмент с одновременно установленными флагами SYN+ACK, при этом записывает в поле «номер подтверждения» (acknowledgement number), полученный порядковый номер, увеличенный на 1 (что подтверждает получение первого сегмента), а также устанавливает свой порядковый номер, который, как и в SYN-сегменте, выбирается произвольно.

3. После получения клиентом сегмента с флагами SYN+ACK соединение считается установленным, клиент, в свою очередь, отправляет в ответ сегмент с флагом ACK, обновленными номерами последовательности, и не содержащий полезной нагрузки.

4. Начинается передача данных.

3 ХОД РАБОТЫ

Для удобства взаимодействия программа предоставляет консольный интерфейс, позволяющий пользователю выполнять следующие действия:

1 Просмотр статистики сервера.
2 Запуск сервера
3 Остановка сервера.
4 Включение или выключение защиты.
5 Отправить сообщение серверу.
6 Выход из приложения.

На рисунке 3.1 изображен результат работы программного продукта.

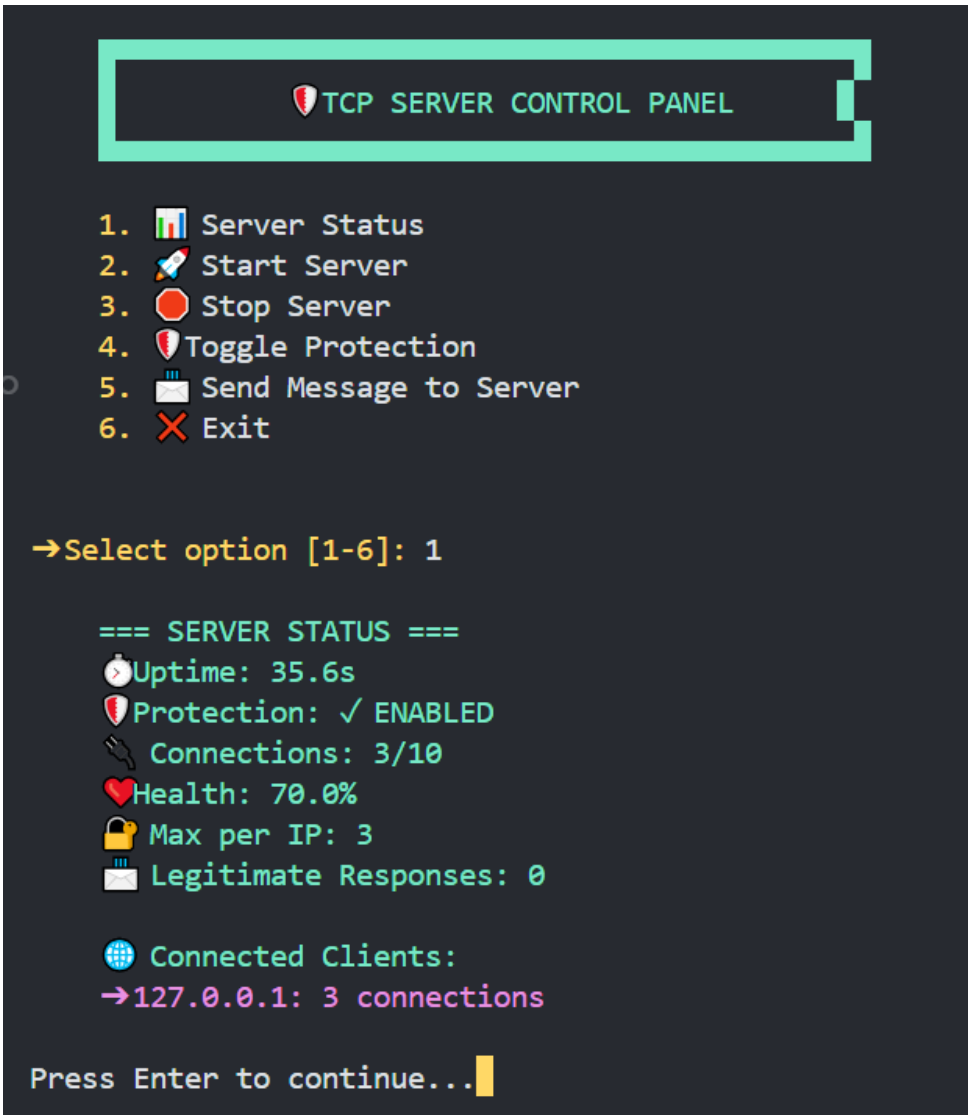

Рисунок 3.1 – Результат работы программного продукта

Таким образом, программа успешно справляется с поставленными задачами.

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была достигнута поставленная цель – изучение принципов защиты TCP-соединений и реализация механизмов противодействия атакам на примере создания защищенного сервера. Разработанная программа успешно демонстрирует основные этапы работы защитных механизмов, которые соответствуют ключевым принципам обеспечения безопасности сетевых приложений.

Принцип работы программы согласно реализованным защитным мерам:

1 Установка соединения: сервер использует трехэтапное рукопожатие (SYN, SYN+ACK, ACK) для установки TCP-соединения с клиентом. При этом в защищенном режиме активируются механизмы защиты от SYN-flood атак, такие как ограничение числа одновременных незавершенных соединений.

2 Контроль частоты запросов: после успешной установки соединения сервер отслеживает количество запросов от каждого IP-адреса. Если число запросов превышает заданный лимит в течение определенного времени, сервер блокирует дальнейшие подключения с этого IP, предотвращая DoS/DDoS атаки.

3 Контроль количества подключений с одного IP адреса: это позволяет предотвратить большое количество подключений от одного пользователя.

4 Мониторинг состояния системы: программа предоставляет возможность просмотра текущего состояния сервера, включая информацию о количестве активных соединений, статусах клиентов и детали обработки запросов. Это помогает анализировать эффективность защитных механизмов и выявлять потенциальные угрозы.

Программа поддерживает работу в двух режимах: защищенном и незащищенном. Защищенный режим активирует все описанные выше механизмы, тогда как незащищенный режим позволяет исследовать поведение системы без этих ограничений. Такой подход дает возможность сравнить устойчивость сервера к различным типам атак и продемонстрировать важность применения защитных мер.

Разработанная программа предоставляет удобный консольный интерфейс, позволяющий выполнять следующие действия: выбор режима работы сервера (защищенный или незащищенный), наблюдение за процессом установки соединений, моделирование атак (например, SYN-flood или перегрузка запросами), а также просмотр состояния системы, включая информацию о текущих соединениях и их параметрах.

Таким образом, в результате проделанной работы были получены практические навыки реализации и анализа механизмов защиты TCP-соединений, а также создана программа, которая может быть использована для демонстрации принципов обеспечения безопасности сетевых приложений в условиях различных угроз.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

```
class FancyServer:
    def __init__(self, host='localhost', port=9999):
        self.host = host
        self.port = port
        self.protection_enabled = False
        self.max_connections = 10
        self.max_connections_per_ip = 3
        self.request_limit = 5
        self.connections = 0
        self.connected_ips = defaultdict(int)
        self.requests = defaultdict(lambda: deque(maxlen=10))
        self.lock = threading.Lock()
        self.running = False
        self.start_time = time.time()
        self.legitimate_responses = 0
    def handle_client(self, client_socket):
        ip = client_socket.getpeername()[0]
        try:
            data = client_socket.recv(1024)
            if data:
                response = b"OK"
                if self.protection_enabled:
                    with self.lock:
                        now = time.time()
                        self.requests[ip].append(now)

                        if len(self.requests[ip]) >= self.request_limit:
                            time_diff = now - self.requests[ip][0]
                            if time_diff < 5.0:
                                response = b"BLOCKED"
                with self.lock:
                    load_factor = min(1.0, self.connections /
self.max_connections)
                delay = load_factor * 2
                time.sleep(delay)

                client_socket.send(response)
                with self.lock:
                    if response == b"OK":
                        self.legitimate_responses += 1
        except:
            pass
        finally:
            client_socket.close()
            with self.lock:
                self.connections -= 1
                self.connected_ips[ip] -= 1
                if self.connected_ips[ip] <= 0:
                    del self.connected_ips[ip]
    def start_server(self):
        self.running = True
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.bind((self.host, self.port))
        server_socket.listen(5)
        print(f"{Fore.GREEN} Server started on
{self.host}:{self.port}{Style.RESET_ALL}")
```