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 №7

на тему

**ЗАЩИТА ПО ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ**

Выполнил: студент гр.253504

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc190789350)

[1 ФОРМУЛИРОВКА ЗАДАЧИ 4](#_Toc190789351)

[2 ПРИМЕР ОБФУСКАЦИИ КОДА 5](#_Toc190789352)

[3 РЕЗУЛЬТАТЫ И ОЦЕНКА 6](#_Toc190789353)

[ЗАКЛЮЧЕНИЕ 7](#_Toc190789354)

ВВЕДЕНИЕ

Современные программные системы сталкиваются с множеством угроз, включая несанкционированный доступ и взлом исходного кода. Одним из методов защиты является обфускация — процесс изменения исходного кода таким образом, чтобы его было трудно понять и проанализировать. Это позволяет значительно усложнить задачу для злоумышленников, пытающихся извлечь и использовать код для атак.

Цель данной работы — продемонстрировать процесс обфускации исходного кода программы traceroute, написанной на языке C. Обфускация делает программу сложной для анализа и позволяет защитить ее от атак, направленных на извлечение логики работы приложения. В данном случае, основное внимание уделяется замене читаемых имен переменных и функций на трудночитаемые, а также изменению структуры кода для предотвращения его несанкционированного использования.

1 ФОРМУЛИРОВКА ЗАДАЧИ

Задача данной работы состоит в том, чтобы продемонстрировать обфускацию исходного кода программы traceroute с целью повышения безопасности кода и защиты от анализа злоумышленниками. Для этого были проведены следующие шаги:

1 Замена имен переменных и функций на трудночитаемые символы.

2 Использование макросов для выноса значений и ключевых элементов программы.

3 Усложнение структуры кода для предотвращения его быстрого анализа.

Обфускация не должна менять функциональность программы, а лишь усложнять ее восприятие. Программа должна выполнять ту же задачу трассировки маршрута до удаленного хоста, но ее исходный код должен стать трудным для понимания.

2 ПРИМЕР ОБФУСКАЦИИ КОДА

Изначально программа traceroute имеет понятные имена переменных и функций, такие как first\_ttl, max\_ttl, parseCommandLine и так далее. Однако после проведения обфускации:

1 Имена переменных были заменены на случайные символы, такие как a1, a2, a3.

2 Все строки и константы, такие как сообщения об ошибках и инструкция, были вынесены в макросы с трудными для восприятия названиями (a7, a8, a9 и т. д.).

3 Функции также были переименованы в трудночитаемые идентификаторы, например, isValidTTL было заменено на a21, parseCommandLine — на a28.

Обфуцированный код на рисунке 1.

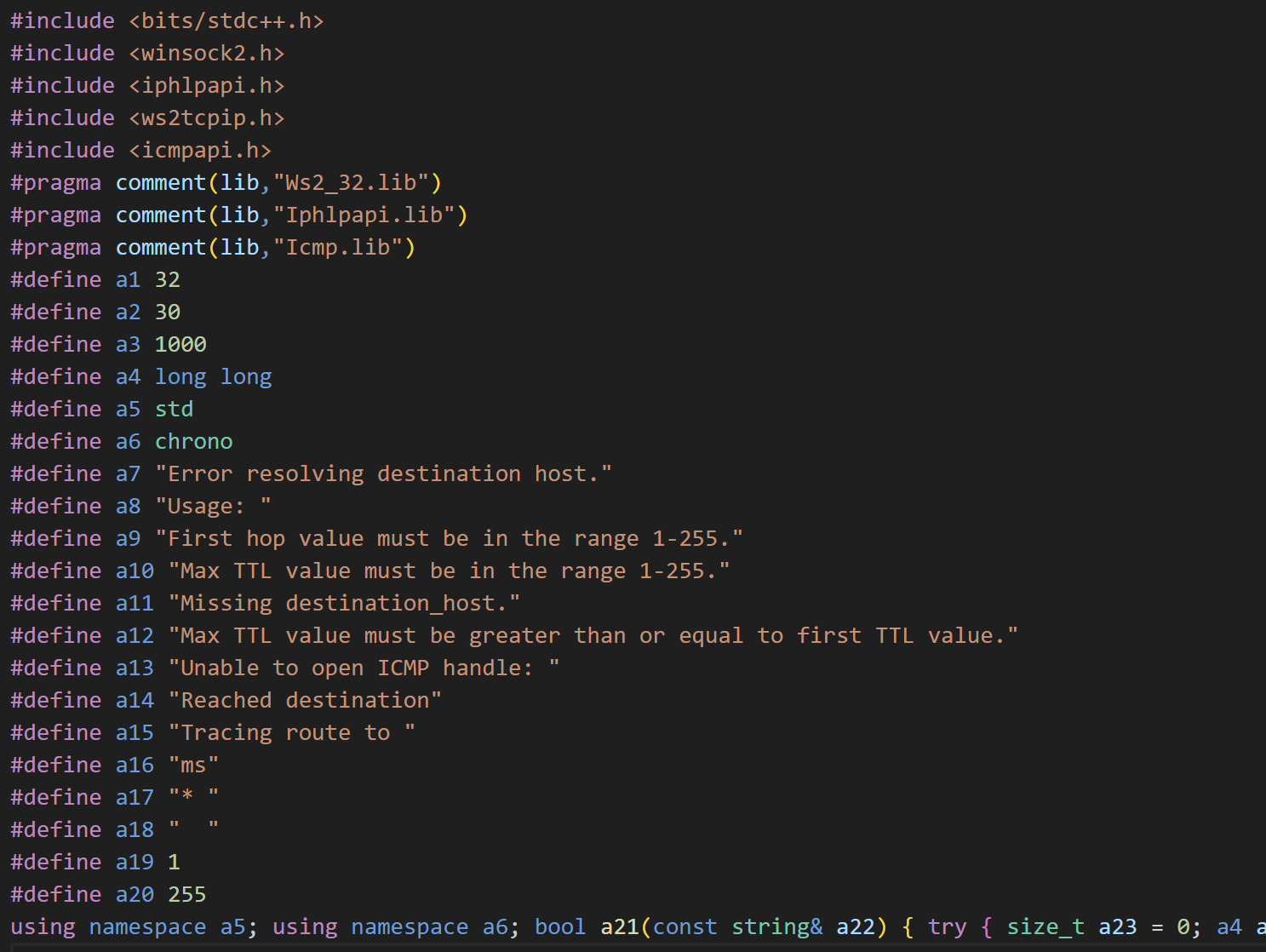


Рисунок 1 – Код после обусфакции

Такой код будет сложнее читать и анализировать, что затрудняет любые попытки извлечь его логику или найти уязвимости. Однако функциональность программы сохраняется.

3 РЕЗУЛЬТАТЫ И ОЦЕНКА

После обфускации код программы продолжает выполнять ту же задачу, что и до обфускации, но теперь его сложнее анализировать. Все переменные и функции имеют случайные имена, и вся логика программы скрыта за этими идентификаторами. Например:

1 Имена таких функций, как traceRoute и resolveHostname, были изменены на сложные идентификаторы (a40, a35), что усложняет их идентификацию.

2 Сообщения об ошибках и строковые значения были заменены на макросы с нелогичными именами, что делает код трудным для анализа.

3 Структура кода была изменена, чтобы она не имела очевидной связи между действиями и переменными.

Этот процесс значительно повышает безопасность программы, так как она становится трудной для обратного инжиниринга, несмотря на сохранение функциональности.

# ЗАКЛЮЧЕНИЕ

В ходе работы была успешно выполнена обфускация программы traceroute, написанной на языке C. Результаты показали, что даже базовые методы обфускации, такие как замена имен переменных и функций, могут значительно усложнить анализ кода и повысить безопасность приложения.

Обфускация доказала свою эффективность как метод защиты программного обеспечения от несанкционированного доступа. Этот метод затрудняет задачу для злоумышленников, которые пытаются изучить исходный код с целью его эксплуатации. Важно подчеркнуть, что обфускация является важным элементом комплексной стратегии защиты программного обеспечения, позволяющим защитить код от обратного инжиниринга и атак.

Таким образом, проведенная работа демонстрирует значимость обфускации для повышения безопасности программного обеспечения в условиях современных угроз и подтверждает, что даже простые методы защиты, такие как изменение имен переменных и функций, могут существенно затруднить несанкционированное использование программы.

ПРИЛОЖЕНИЕ А

**(обязательное)**

**Листинг программного кода**

Листинг А.1 – файл *first.cpp*

#include <bits/stdc++.h>

#include <winsock2.h>

#include <iphlpapi.h>

#include <ws2tcpip.h>

#include <icmpapi.h>

#pragma comment(lib, "Ws2\_32.lib")

#pragma comment(lib, "Iphlpapi.lib")

#pragma comment(lib, "Icmp.lib")

#define PACKET\_SIZE 32

#define MAX\_HOPS 30

#define TIMEOUT\_SEC 1000

#define ll long long

using namespace std;

using namespace chrono;

bool isValidTTL(const string &ttl\_str)

{

try

{

size\_t pos = 0;

ll ttl\_value = stoi(ttl\_str, &pos);

if (pos != ttl\_str.size())

{

return false;

}

return (ttl\_value >= 1 and ttl\_value <= 255);

}

catch (const invalid\_argument &e)

{

return false;

}

}

void printHelp(const char \*programName)

{

cerr << "Usage: " << programName << " [-f first\_ttl] [-m max\_ttl] <destination\_host>" << endl;

cerr << "Options:" << endl;

cerr << " -f, --first-ttl=VALUE Start from the first\_ttl hop (instead from 1)" << endl;

cerr << " -m, --max-ttl=VALUE Set the max number of hops (max TTL to be reached). Default is 30" << endl;

;

cerr << " -h, --help Read this help and exit" << endl;

}

bool parseCommandLine(ll argc, char \*argv[], ll &first\_ttl, ll &max\_ttl, string &destination\_host)

{

for (ll i = 1; i < argc; ++i)

{

if (strcmp(argv[i], "-f") == 0 and i + 1 < argc)

{

if (!isValidTTL(argv[i + 1]))

{

cerr << "First hop value must be in the range 1-255." << endl;

return false;

}

first\_ttl = stoi(argv[++i]);

}

else if (strcmp(argv[i], "-m") == 0 and i + 1 < argc)

{

if (!isValidTTL(argv[i + 1]))

{

cerr << "Max TTL value must be in the range 1-255." << endl;

return false;

}

max\_ttl = stoi(argv[++i]);

}

else if (strcmp(argv[i], "-h") == 0 or strcmp(argv[i], "--help") == 0)

{

printHelp(argv[0]);

return false;

}

else

{

destination\_host = argv[i];

}

}

if (destination\_host.empty())

{

cerr << "Missing destination\_host." << endl;

printHelp(argv[0]);

return false;

}

if (max\_ttl < first\_ttl)

{

cerr << "Max TTL value must be greater than or equal to first TTL value." << endl;

return false;

}

return true;

}

bool resolveHostname(const string &hostname, struct sockaddr\_in &address)

{

struct addrinfo hints = {0};

struct addrinfo \*result = nullptr;

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_RAW;

hints.ai\_protocol = IPPROTO\_ICMP;

if (getaddrinfo(hostname.c\_str(), nullptr, &hints, &result) != 0)

{

cerr << "Error resolving destination host." << endl;

return false;

}

address = \*reinterpret\_cast<struct sockaddr\_in \*>(result->ai\_addr);

freeaddrinfo(result);

return true;

}

void traceRoute(const struct sockaddr\_in &addr, ll first\_ttl, ll max\_ttl)

{

HANDLE icmpHandle = IcmpCreateFile();

if (icmpHandle == INVALID\_HANDLE\_VALUE)

{

cerr << "Unable to open ICMP handle: " << GetLastError() << endl;

return;

}

char sendData[PACKET\_SIZE] = {0};

DWORD replySize = sizeof(ICMP\_ECHO\_REPLY) + PACKET\_SIZE;

char \*replyBuffer = new char[replySize];

for (ll ttl = first\_ttl; ttl <= max\_ttl; ++ttl)

{

cout << ttl << " ";

for (ll i = 0; i < 3; ++i)

{

IP\_OPTION\_INFORMATION optionInfo = {0};

optionInfo.Ttl = ttl;

DWORD result = IcmpSendEcho(icmpHandle, addr.sin\_addr.S\_un.S\_addr, sendData, sizeof(sendData), &optionInfo, replyBuffer, replySize, TIMEOUT\_SEC);

if (result != 0)

{

PICMP\_ECHO\_REPLY echoReply = reinterpret\_cast<PICMP\_ECHO\_REPLY>(replyBuffer);

struct in\_addr replyAddr;

replyAddr.S\_un.S\_addr = echoReply->Address;

cout << inet\_ntoa(replyAddr) << " (" << echoReply->RoundTripTime << " ms) ";

if (echoReply->Status == IP\_SUCCESS and echoReply->Address == addr.sin\_addr.S\_un.S\_addr)

{

cout << "Reached destination" << endl;

delete[] replyBuffer;

IcmpCloseHandle(icmpHandle);

return;

}

}

else

{

cout << "\* ";

}

this\_thread::sleep\_for(milliseconds(500));

}

cout << endl;

}

delete[] replyBuffer;

IcmpCloseHandle(icmpHandle);

}

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

{

ll first\_ttl = 1;

ll max\_ttl = MAX\_HOPS;

string destination\_host;

if (!parseCommandLine(argc, argv, first\_ttl, max\_ttl, destination\_host))

{

return 1;

}

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)

{

cerr << "WSAStartup failed." << endl;

return 1;

}

struct sockaddr\_in dest\_addr;

if (!resolveHostname(destination\_host, dest\_addr))

{

WSACleanup();

return 1;

}

cout << "Tracing route to " << destination\_host << " [" << inet\_ntoa(dest\_addr.sin\_addr) << "]" << endl;

traceRoute(dest\_addr, first\_ttl, max\_ttl);

WSACleanup();

return 0;

}

// g++ -std=c++17 -finput-charset=UTF-8 main.cpp -o build/main -lws2\_32 -liphlpapi -licmp

Листинг А.2 – файл *second.cpp*

#include <bits/stdc++.h>

#include <winsock2.h>

#include <iphlpapi.h>

#include <ws2tcpip.h>

#include <icmpapi.h>

#pragma comment(lib, "Ws2\_32.lib")

#pragma comment(lib, "Iphlpapi.lib")

#pragma comment(lib, "Icmp.lib")

#define a1 32

#define a2 30

#define a3 1000

#define a4 long long

#define a5 std

#define a6 chrono

#define a7 "Error resolving destination host."

#define a8 "Usage: "

#define a9 "First hop value must be in the range 1-255."

#define a10 "Max TTL value must be in the range 1-255."

#define a11 "Missing destination\_host."

#define a12 "Max TTL value must be greater than or equal to first TTL value."

#define a13 "Unable to open ICMP handle: "

#define a14 "Reached destination"

#define a15 "Tracing route to "

#define a16 "ms"

#define a17 "\* "

#define a18 " "

#define a19 1

#define a20 255

using namespace a5;

using namespace a6;

bool a21(const string &a22)

{

try

{

size\_t a23 = 0;

a4 a24 = stoi(a22, &a23);

if (a23 != a22.size())

return false;

return (a24 >= a19 && a24 <= a20);

}

catch (const invalid\_argument &a25)

{

return false;

}

}

void a26(const char \*a27) { cerr << a8 << a27 << " [-f first\_ttl] [-m max\_ttl] <destination\_host>" << endl

<< "Options:" << endl

<< " -f, --first-ttl=VALUE Start from the first\_ttl hop (instead from 1)" << endl

<< " -m, --max-ttl=VALUE Set the max number of hops (max TTL to be reached). Default is " << a2 << endl

<< " -h, --help Read this help and exit" << endl; }

bool a28(a4 a29, char \*a30[], a4 &a31, a4 &a32, string &a33)

{

for (a4 a34 = a19; a34 < a29; ++a34)

{

if (strcmp(a30[a34], "-f") == 0 && a34 + a19 < a29)

{

if (!a21(a30[a34 + a19]))

{

cerr << a9 << endl;

return false;

}

a31 = stoi(a30[++a34]);

}

else if (strcmp(a30[a34], "-m") == 0 && a34 + a19 < a29)

{

if (!a21(a30[a34 + a19]))

{

cerr << a10 << endl;

return false;

}

a32 = stoi(a30[++a34]);

}

else if (strcmp(a30[a34], "-h") == 0 || strcmp(a30[a34], "--help") == 0)

{

a26(a30[0]);

return false;

}

else

{

a33 = a30[a34];

}

}

if (a33.empty())

{

cerr << a11 << endl;

a26(a30[0]);

return false;

}

if (a32 < a31)

{

cerr << a12 << endl;

return false;

}

return true;

}

bool a35(const string &a36, struct sockaddr\_in &a37)

{

struct addrinfo a38 = {0};

struct addrinfo \*a39 = nullptr;

a38.ai\_family = AF\_INET;

a38.ai\_socktype = SOCK\_RAW;

a38.ai\_protocol = IPPROTO\_ICMP;

if (getaddrinfo(a36.c\_str(), nullptr, &a38, &a39) != 0)

{

cerr << a7 << endl;

return false;

}

a37 = \*reinterpret\_cast<struct sockaddr\_in \*>(a39->ai\_addr);

freeaddrinfo(a39);

return true;

}

void a40(const struct sockaddr\_in &a41, a4 a42, a4 a43)

{

HANDLE a44 = IcmpCreateFile();

if (a44 == INVALID\_HANDLE\_VALUE)

{

cerr << a13 << GetLastError() << endl;

return;

}

char a45[a1] = {0};

DWORD a46 = sizeof(ICMP\_ECHO\_REPLY) + a1;

char \*a47 = new char[a46];

for (a4 a48 = a42; a48 <= a43; ++a48)

{

cout << a48 << a18;

for (a4 a49 = 0; a49 < 3; ++a49)

{

IP\_OPTION\_INFORMATION a50 = {0};

a50.Ttl = a48;

DWORD a51 = IcmpSendEcho(a44, a41.sin\_addr.S\_un.S\_addr, a45, sizeof(a45), &a50, a47, a46, a3);

if (a51 != 0)

{

PICMP\_ECHO\_REPLY a52 = reinterpret\_cast<PICMP\_ECHO\_REPLY>(a47);

struct in\_addr a53;

a53.S\_un.S\_addr = a52->Address;

cout << inet\_ntoa(a53) << " (" << a52->RoundTripTime << " " << a16 << ") ";

if (a52->Status == IP\_SUCCESS && a52->Address == a41.sin\_addr.S\_un.S\_addr)

{

cout << a14 << endl;

delete[] a47;

IcmpCloseHandle(a44);

return;

}

}

else

{

cout << a17;

}

this\_thread::sleep\_for(milliseconds(500));

}

cout << endl;

}

delete[] a47;

IcmpCloseHandle(a44);

}

int main(int a29, char \*a30[])

{

a4 a54 = a19;

a4 a55 = a2;

string a56;

if (!a28(a29, a30, a54, a55, a56))

{

return 1;

}

WSADATA a57;

if (WSAStartup(MAKEWORD(2, 2), &a57) != 0)

{

cerr << "WSAStartup failed." << endl;

return 1;

}

struct sockaddr\_in a58;

if (!a35(a56, a58))

{

WSACleanup();

return 1;

}

cout << a15 << a56 << " [" << inet\_ntoa(a58.sin\_addr) << "]" << endl;

a40(a58, a54, a55);

WSACleanup();

return 0;

}