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 №6

на тему

**ЗАЩИТА ПО ОТ НЕСАНКЦИОНИРОВАННОГО ИСПОЛЬЗОВАНИЯ**

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

Познакомиться с основными технологиями защиты программного обеспечения от несанкционированного использования. Получить навыки защиты разработанной программы от несанкционированного копирования.

# 2 БЛОК-СХЕМА АЛГОРИТМА

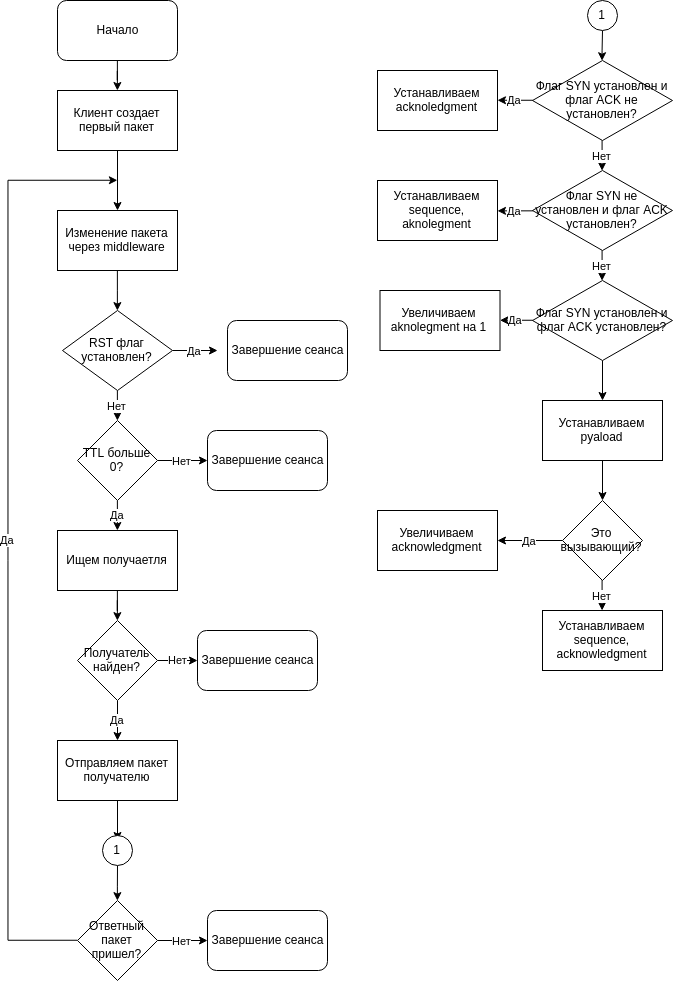
На рисунке 1 продемонстрирована блок-схема алгоритма.

Рисунок 1 – Блок-схема алгоритма программы

# 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

На рисунке 2 представлена часть кода после обфускации.

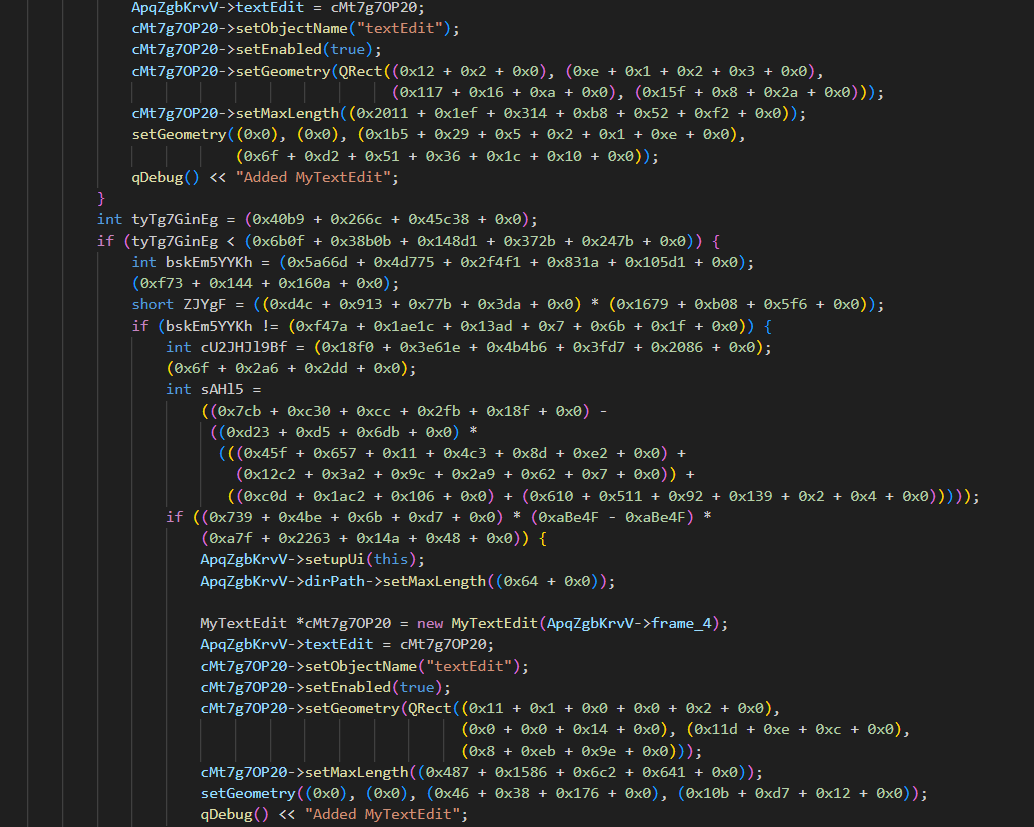


Рисунок 2 – Код после обфскации

# ВЫВОДЫ

В ходе выполнения лабораторной работы была разработана программа для обфускации исходного кода лабораторной 4.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

#include <algorithm>

#include <boost/regex.hpp>

#include <cstdlib>

#include <filesystem>

#include <format>

#include <fstream>

#include <functional>

#include <iostream>

#include <numeric>

#include <random>

#include <set>

#include <string>

#include <unordered\_map>

#include <unordered\_set>

#include <vector>

std::unordered\_set<std::string> NOT\_NAMES = {

"main",

"include",

"QSqlDatabase",

"std::string",

"std",

"optional",

"DB",

"QSqlQuery",

"size\_t",

"textEdit",

"MyTextEdit",

"int",

"QMimeData",

"QWidget",

"QTextEdit",

"Q\_OBJECT",

"user\_actions",

"Ui",

"QDirIterator",

"QDir",

"Files",

"QFileInfo",

"new",

"SLOT",

"Subdirectories",

"qint64",

"QMessageBox",

"warning",

"addItem",

"next",

"while",

"for",

"do",

"hasNext",

"canonicalFilePath",

"true",

"false",

"saveBtn",

"break",

"continue",

"critical",

"QListWidgetItem",

"QIODevice",

"ReadOnly",

"QTextStream",

"setPlainText",

"timeout",

"readAll",

"QTimer",

"setSingleShot",

"setInterval",

"connect",

"QObject",

"setEnabled",

"QString",

"QRect",

"startsWith",

"QStringList",

"deleteLater",

"start",

"currentItem",

"foundFiles",

"QFile",

"text",

"open",

"WriteOnly",

"toPlainText",

"close",

"insertFromMimeData",

"user\_actions",

"nullptr",

"private",

"public",

"signals",

"explicit",

"static",

"const",

"include",

"ifndef",

"endif",

"define",

"bool",

"login",

"setupUi",

"protected",

"setHostName",

"setDatabaseName",

"QMainWindow",

"qDebug",

"setUserName",

"setPassword",

"override",

"once",

"void",

"string",

"hasText",

"length",

"addDatabase",

"return",

"slots",

"sign\_in",

"atEnd",

"sign\_out",

"login\_in",

"password\_in",

"dirPath",

"size",

"SIGNAL",

"delete",

"isEmpty",

"auto",

"if",

"first",

"else",

"value\_or",

"this",

"emit",

"value",

"toInt",

"prepare",

"bindValue",

"exec",

"class",

"namespace",

"QApplication",

"MainWindow",

"char",

"QSize",

"QT\_BEGIN\_NAMESPACE",

"QT\_END\_NAMESPACE",

"setCentralWidget",

"show",

"hide",

"centralWidget",

"setParent",

"setMaxLength",

"frame\_4",

"setGeometry",

"setObjectName",

"on\_pushButton\_clicked",

"on\_searchFilesBtn\_clicked",

"on\_foundFiles\_itemClicked",

"on\_saveBtn\_clicked",

"on\_signOutBtn\_clicked",

};

std::random\_device rd;

std::mt19937 generator(rd());

std::uniform\_int\_distribution<> distribution(0, 1'000'000);

std::unordered\_map<std::string, std::string> VAR\_NAMES;

std::string generateRandomName(int length) {

std::string possibleChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890\_";

std::string randomName;

do {

for (int i = 0; i < length; i++) {

randomName += possibleChars[distribution(generator) % possibleChars.size()];

}

if (!std::isalpha(randomName[0]) && randomName[0] != '\_') {

randomName[0] = possibleChars[distribution(generator) % 52];

}

} while (VAR\_NAMES.contains(randomName));

return randomName;

}

std::string getCmpSign(int lhs, int rhs) {

if (lhs == rhs) {

std::vector<std::string> cmp = {"==", ">=", "<="};

return cmp[distribution(generator) % cmp.size()];

} else if (lhs > rhs) {

std::vector<std::string> cmp = {"!=", ">=", ">"};

return cmp[distribution(generator) % cmp.size()];

} else {

std::vector<std::string> cmp = {"!=", "<=", "<"};

return cmp[distribution(generator) % cmp.size()];

}

}

std::string getOppositeCmpSign(int lhs, int rhs) {

if (lhs == rhs) {

return "!=";

} else if (lhs > rhs) {

return "<";

} else {

return ">";

}

}

std::string generateExpr(bool nested = false, int level = 0, int maxLevel = 3) {

if (!nested || level > maxLevel) {

auto val = distribution(generator);

return std::to\_string(val % 12345);

} else {

bool nested1 = distribution(generator) % 2;

bool nested2 = distribution(generator) % 2;

std::vector<std::string> operators = {"+", "-", "\*"};

return "(" + generateExpr(nested1, level + 1) + " " + operators[distribution(generator) % operators.size()] +

" " + generateExpr(nested2, level + 1) + ")";

}

}

std::string generateAssignment(std::string var) {

std::vector<std::string> types = {"int", "long long", "unsigned long long", "short"};

return types[distribution(generator) % types.size()] + " " + var + " = " + generateExpr(1);

}

std::string generateDeadCode() {

std::string result;

result += generateExpr() + ";\n";

result += generateAssignment(generateRandomName(5)) + ";\n";

return result;

}

std::string generateIfElse(const std::string &varName, int varValue, size\_t level, size\_t maxLevel, bool isTrue,

const std::string &code, bool &wasInserted, std::vector<bool> path);

void generateIfElseHelper(std::function<std::string(int, int)> getCmpSignFunc, bool isTrue, const std::string &varName,

int varValue, size\_t level, size\_t maxLevel, const std::string &code, bool &wasInserted,

const std::vector<bool> &path, std::string &result, const std::vector<bool> &ifPath,

const std::vector<bool> &elsePath) {

bool isNextTrue = distribution(generator) % 2;

int ifVal = distribution(generator);

result += "if (" + varName + getCmpSignFunc(varValue, ifVal) + std::to\_string(ifVal) + ") {\n";

auto newVarName = generateRandomName(10);

auto newVarValue = distribution(generator);

result += "int " + newVarName + " = " + std::to\_string(newVarValue) + ";\n";

result += generateDeadCode();

result += generateIfElse(newVarName, newVarValue, level + 1, maxLevel, isNextTrue, code, wasInserted, ifPath);

if (isTrue == false) result += "} else {\n";

if (level == maxLevel && !wasInserted &&

static\_cast<size\_t>(std::count(path.begin(), path.end(), true)) == path.size()) {

result += code + '\n';

wasInserted = true;

} else {

result += "if (" + generateExpr() + "\* (0xaBe4F - 0xaBe4F) \* " + generateExpr() + ") {" + code + "}\n";

}

if (isTrue) result += "} else {\n";

newVarName = generateRandomName(10);

newVarValue = distribution(generator);

result += std::string((level + 1) \* 4, ' ') + "int " + newVarName + " = " + std::to\_string(newVarValue) + ";\n";

isNextTrue = distribution(generator) % 2;

result += generateIfElse(newVarName, newVarValue, level + 1, maxLevel, isNextTrue, code, wasInserted, elsePath);

result += "}\n";

}

std::string generateIfElse(const std::string &varName, int varValue, size\_t level, size\_t maxLevel, bool isTrue,

const std::string &code, bool &wasInserted, std::vector<bool> path) {

if (level > maxLevel) {

return "";

}

std::string result;

auto ifPath = path, elsePath = path;

ifPath.push\_back(isTrue);

elsePath.push\_back(!isTrue);

std::function<std::string(int, int)> getCmpSignFunc = (isTrue ? getCmpSign : getOppositeCmpSign);

generateIfElseHelper(getCmpSignFunc, isTrue, varName, varValue, level, maxLevel, code, wasInserted, path, result,

ifPath, elsePath);

return result;

}

std::string renameVars(const std::string &code) {

boost::regex pattern(R"((?<![<#\/])(?=(?:[^"']|"[^"]\*"|'[^']\*')\*$)\b([a-zA-Z\_]\w\*)\b(?![<>]))");

std::string result = boost::regex\_replace(

code, pattern,

[&](const boost::smatch &match) {

std::string varName = match.str(1);

if (NOT\_NAMES.contains(varName)) return match.str(0);

if (VAR\_NAMES.find(varName) == VAR\_NAMES.end()) {

VAR\_NAMES[varName] = generateRandomName(10);

}

return VAR\_NAMES[varName];

},

boost::match\_default | boost::format\_all);

return result;

}

std::string replaceNumbers(const std::string &code) {

boost::regex pattern(R"((?<!\.)\b(\d+)\b(?!\.))");

std::string result = boost::regex\_replace(

code, pattern,

[&](const boost::smatch &match) {

std::string result;

int number = std::stoi(match.str(1));

int cur = 0;

int rem = number + 1;

int maxOpCnt = distribution(generator) % 4 + 2;

while (cur != number) {

if (maxOpCnt == 0) {

result += "0x" + std::format("{:x}", rem - 1) + " + ";

break;

}

int tmp = distribution(generator) % rem;

result += "0x" + std::format("{:x}", tmp) + " + ";

cur += tmp;

rem -= tmp;

maxOpCnt--;

}

result += "0x0";

return "(" + result + ")";

},

boost::match\_default | boost::format\_all);

return result;

}

std::string insertIfElse(const std::string &code) {

boost::regex pattern(R"((?<!\w\s|=\s)(\{)([^{}]+)(\}))");

std::string result = boost::regex\_replace(

code, pattern,

[&](const boost::smatch &match) {

auto code = match.str(2);

bool wasInserted = false;

code = generateIfElse("a", distribution(generator) % 10000, 0, 4, distribution(generator) % 2, code,

wasInserted, {});

return match.str(1) + " int a = 0xaB1f \* 0xBc94 - 0x7e0db1EC;" + code + match.str(3);

},

boost::match\_default | boost::format\_all);

return result;

}

std::string obfuscate(const std::string &code) {

std::string result;

result = renameVars(code);

result = insertIfElse(result);

result = replaceNumbers(result);

return result;

}

std::vector<std::filesystem::path> getFileNames(const std::filesystem::path &dataDir) {

std::vector<std::filesystem::path> fileNames;

for (const auto &entry : std::filesystem::recursive\_directory\_iterator(dataDir)) {

if (entry.is\_regular\_file() && (entry.path().extension() == ".cpp" || entry.path().extension() == ".h")) {

std::filesystem::path filePath = std::filesystem::absolute(entry.path()).lexically\_normal();

fileNames.push\_back(filePath);

}

}

return fileNames;

}

std::string readFile(const std::filesystem::path &path) {

std::string comm = "clang-format -i " + path.string();

system(comm.c\_str());

auto file = std::ifstream(path);

std::string code;

std::string line;

while (std::getline(file, line)) {

code += line + '\n';

}

file.close();

return code;

}

int main() {

auto fileNames = getFileNames("lab-files");

for (const auto &path : fileNames) {

auto code = readFile(path);

std::string obfuscated\_code = obfuscate(code);

std::string from = "lab-files";

std::string to = "lab-files-copy";

size\_t replacePos = path.string().find(from);

auto outPath = std::filesystem::path(path.string().replace(replacePos, from.size(), to));

std::ofstream out(outPath);

out << obfuscated\_code;

out.close();

std::string comm = "clang-format -i " + outPath.string();

system(comm.c\_str());

}

return 0;

}