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 № 4-6

|  |  |
| --- | --- |
|  | Выполнил студент группы 053505  Слуцкий Никита Сергеевич |
|  | Проверил ассистент каф.информатики  Протько Мирослав Игоревич |

Минск 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc132111711)

[1 Теоретические сведения 5](#_Toc132111712)

[2 Выполнение работы 7](#_Toc132111713)

[3 Тестирование программного продукта 8](#_Toc132111714)

[Заключение 9](#_Toc132111715)

[ПРИЛОЖЕНИЕ А Листинг кода 10](#_Toc132111716)

# ВВЕДЕНИЕ

Целью данной лабораторной работы является выполнением нескольких объединённых заданий.

Реализовать приложение с графическим интерфейсом, удовлетворяющее следующим требованиям:

1. Приложение проводит аутентификацию пользователя.
2. Каждый пользователь программы должен относиться к какой-нибудь группе пользователей (роли), членам которой доступны различные функциональные возможности программы.
3. Программа должна принимать от пользователя некоторые данные и, возможно, после некоторой обработки, отображать их.
4. При этом должна осуществляться защита от как минимум 4-х типов возможных атак на приложение:
5. Атака «переполнение буфера».
6. Атака «SQL-инъекции».
7. Атака, эксплуатирующая ошибки канонизации.
8. Атака «XSS» (межсайтовое кодирование).
9. Принцип минимизации привилегий.
10. DoS-атака (отказ в обслуживании).

При разработке защиты нужно предположить, что приложение работает с базой данных, в которой сохраняет введённые пользователем данные.

**Задание II.**

Реализовать приложение-инсталлятор, позволяющее установить на компьютер пользователя приложение, реализованное в предыдущем пункте задания.

Требования к приложению:

1. Приложение-инсталлятор совместно с устанавливаемым приложением должно обеспечивать защиту программного продукта от несанкционированного тиражирования.
2. Приложение-инсталлятор должно иметь защиту от возможных атак на него.

**Задание III.**

Протестировать правильность работы систем защиты разработанных приложений посредством реализации тестовых атак выбранных 4-х типов.

**ЗАДАНИЕ:**

1. Реализовать на выбор 3 метода обфуксации программного кода приложения, разработанного в рамках лабораторных работ 4,5, позволяющие защитить ПО от несанкционированного использования в следующих комбинациях:

* По одному.
* Любые 2 на выбор из трех одновременно.
* Все три одновременно.

1. Протестировать работоспособность приложения с запутанным программным кодом.
2. Проверить и пояснить следующие свойства, которым должна удовлетворять запутанная программа:

* Запутывание должно быть *замаскированным*. То, что к программе были применены запутывающие преобразования, не должно бросаться в глаза.
* Запутывание не должно быть регулярным. Регулярная структура запутанной программы или её фрагмента позволяет человеку отделить запутанные части и даже идентифицировать алгоритм запутывания.

# 1 Теоретические сведения

Одним из основных способов защиты авторских прав разработчика **является запутывание (obfuscated) или обфуксация программного кода.**

**Обфуска́ция** (от лат. *obfuscare* — затенять, затемнять; и англ. *obfuscate* — делать неочевидным, запутанным, сбивать с толку) или **запутывание кода** — приведение исходного текста или исполняемого кода программы к виду, сохраняющему её функциональность, но затрудняющему анализ, понимание алгоритмов работы и модификацию при декомпиляции.

Обфуксация производится в следующих целях:

1. Затруднение декомпиляции/отладки и изучения программ с целью обнаружения функциональности.
2. Затруднение декомпиляции пропритарных программ с целью предотвращения обратной разработки или обхода DRM и систем проверки лицензий.
3. Оптимизация программы с целью уменьшения размера работающего кода и (если используется некомпилируемый язык) ускорения работы.
4. Демонстрация неочевидных возможностей языка и квалификации программиста (если производится вручную, а не инструментальными средствами).

«Запутывание» кода может осуществляться на уровне алгоритма, исходного текста и/или ассемблерного текста. Для создания запутанного ассемблерного текста могут использоваться специализированные компиляторы, использующие неочевидные или недокументированные возможности среды исполнения программы. Существуют также специальные программы, производящие обфускацию, называемые **обфускаторами** (англ. *obfuscator*).

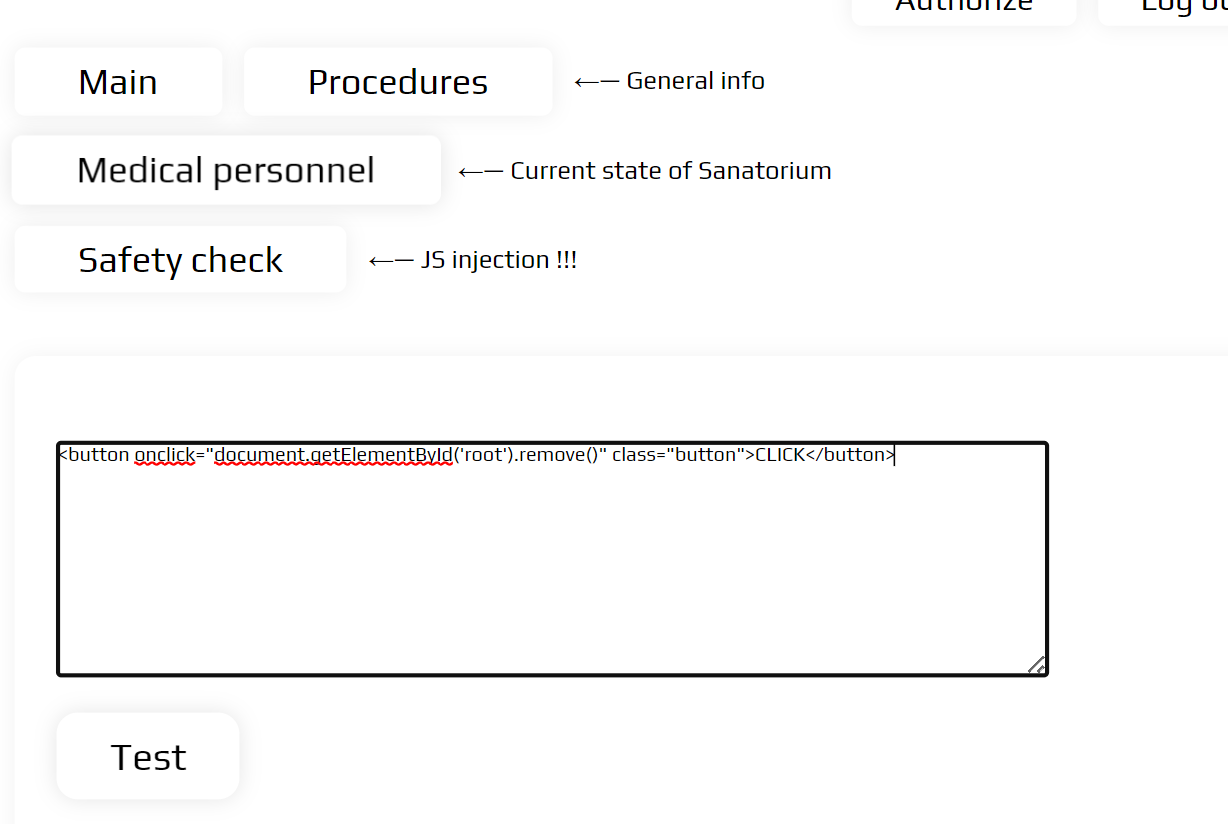
# 2 Выполнение работы

Для выполнения данной лабораторной работы была использована ОС Linux Ubuntu, язык программирования C++ (с применением синтаксиса современного C++ 17), компилятор GNU g++, а также системные вызовы Linux для создания сокетов, соединений.

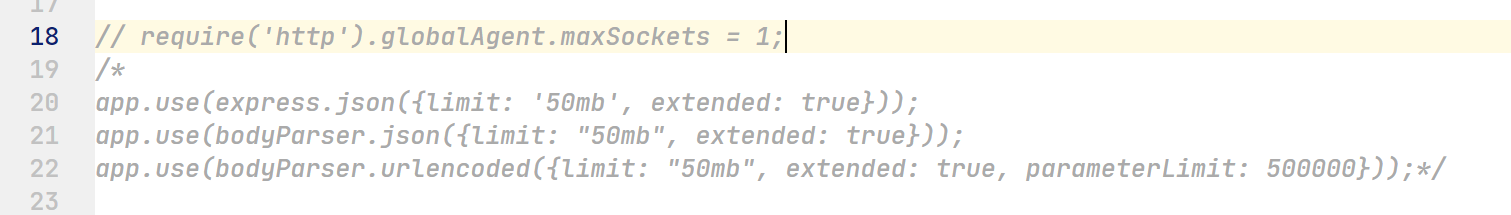
За теоретический источник информации взята книга «Computer Security: A Hands-on Approach» .

Для успешного выполнения работы также необходимы утилиты Linux, описанные в файле Readme в репозитории с лабораторной работой. Дальнейшее описание опускается во избежание дублирования уже написанной информации с ходом выполнения.

# 3 тестирование программного продукта







# Заключение

В результате выполнения лабораторной работы на практике была реализована связка клиента TCP и сервера TCP, а также продемонстрирована работа сканирующих утилит Linux для наблюдения за этой связкой.

Была использована утилита Linux для атаки на работающий сервер, а также написан код, который выполняет то же самое.

Цели лабораторной работы можно считать достигнутыми. Работа выполнена.

# ПРИЛОЖЕНИЕ А

**Листинг кода**

**call npm install**

**call ts-node installer.ts**

**cd app**

**cd client**

**call npm install**

**cd ..**

**cd server**

**call npm install**

**const decompress = require("decompress");**

**async function install(input, output) {**

**decompress(input, output)**

**.then((files) => {**

**console.log('successfully unzipped')**

**})**

**.catch((error) => {**

**console.error('Error occured')**

**console.log(error);**

**});**

**}**

**async function main() {**

**const input = 'app.zip'**

**const output = './'**

**await install(input, output)**

**}**

**import fs from 'fs'**

**var JavaScriptObfuscator = require('javascript-obfuscator');**

**const main = async () => {**

**const code = await fs.promises.readFile('./installer.js', 'utf-8')**

**console.log(code)**

**const obfuscatedCode = JavaScriptObfuscator.obfuscate(code, {**

**compact: false,**

**controlFlowFlattening: true,**

**controlFlowFlatteningThreshold: 1,**

**numbersToExpressions: true,**

**simplify: true,**

**stringArrayShuffle: true,**

**splitStrings: true,**

**stringArrayThreshold: 1**

**});**

**console.log(obfuscatedCode)**

***// await fs.promises.rm('installer-obfuscated.js')***

***// await fs.promises.open('installer-obfuscated.js')***

**await fs.promises.writeFile('./installer-obfuscated.js', obfuscatedCode.\_obfuscatedCode)**

**}**

**main()**