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**

Выполнил:

студент гр. 153503

Бобко И.В.

Проверил:

Лещенко Е.А.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc157598705)

[2 Теоретические сведения 4](#_Toc157598706)

[3 Полученные результаты 5](#_Toc157598712)

[Выводы 7](#_Toc157598713)

[Приложение А (обязательное) Листинг кода 8](#_Toc157598714)

# **1 ЦЕЛЬ РАБОТЫ**

Цель лабораторной работы – познакомиться с основными технологиями защиты программного обеспечения от несанкционированного использования, а также получить навыки защиты разработанной программы от несанкционированного копирования. Защиту ПО требуется осуществить при помощи реализации на выбор 3 методов обфускации программного кода.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Обфускация (от лат. obfuscare — затенять, затемнять; и англ. obfuscat*e* — 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).

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В ходе выполнения данной лабораторной работы было разработано приложение обфускатор, которому на вход подаётся путь к файлу типа js для обфускации. Обфускатор производит запутывание исходного кода следующими способами: переименованием переменных, добавлением недостижимого и мертвого кода, а также подстановкой кода функции в месте ее вызова. Эти способы можно использовать по отдельности или комбинировать любым образом.

Сам обфускатор работает не с исходным кодом программы непосредственно, а с его абстрактным синтаксическим деревом. Благодаря этому можно удобно взаимодействовать со всеми узлами-переменными, а также добавлять мертвый код в подходящие места программы, не вызывая в ней синтаксических ошибок.

Так, при переименовании переменных обфускатор получает список переменных посредством обращения к узлам, содержащим присваивание. После этого каждая переменная из списка будет переименована в новый, случайно сгенерированный набор символов (рисунок 1).



Рисунок 1 – Переименование переменных

В качестве мертвого и недостижимого кода добавляются циклы while и вложенные функции вместе с неиспользуемыми переменными. Количество добавленных «бесполезных» блоков прямо пропорционально размеру входной программы – в 5 раз меньше, чем узлов в абстрактном синтаксическом дереве (рисунок 2).



Рисунок 2 – Добавление мертвого и недостижимого кода

Для запутывания структур данных, было произведена группировка констант и различных данных в массив, в котором также лежит мусор (рисунок 3).



Рисунок 3 – Запутывание данных

# **ВЫВОДЫ**

В ходе выполнения лабораторной работы были реализованы 3 метода обфускации, в результате работы которых исходный код обфусцируемой программы становится более запутанным, но при этом сохраняет работоспособность.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *obfuskator.js*

const fs = require('node:fs');

const readline = require('readline');

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

function obfuscateVariables(code) {

const variableRegex = /\b[a-zA-Z\_][a-zA-Z0-9\_]\*\b/g;

let obfuscatedCode = code;

let variableCounter = 0;

obfuscatedCode = obfuscatedCode.replace(variableRegex, match => {

const obfuscatedName = `\_0x${(variableCounter++).toString(16)}`;

return obfuscatedName;

});

return obfuscatedCode;

}

const jso = require('javascript-obfuscator');

function formattingConversions(data) {

return jso.obfuscate(data, {

compact: false,

controlFlowFlattening: false,

controlFlowFlatteningThreshold: 0,

numbersToExpressions: false,

simplify: true,

sourceMapMode: 'inline',

stringArrayWrappersChainedCalls: false,

identifierNamesGenerator:'hexadecimal',

deadCodeInjection: false,

deadCodeInjectionThreshold: 0,

renameProperties: false,

renameVariables: false,

stringArrayShuffle: false,

splitStrings: false,

stringArray: false,

stringArrayThreshold: 0

});

}

function obfuscateAndAddDeadCode(code, deadCode) {

function obfuscateVariables(code) {

const variableRegex = /\b[a-zA-Z\_][a-zA-Z0-9\_]\*\b/g;

let obfuscatedCode = code;

let variableCounter = 0;

obfuscatedCode = obfuscatedCode.replace(variableRegex, match => {

const obfuscatedName = `\_0x${(variableCounter++).toString(16)}`;

return obfuscatedName;

});

return obfuscatedCode;

}

function addDeadCode(code, deadCode) {

return deadCode + '\n' + code;

}

const obfuscatedCode = obfuscateVariables(code);

const codeWithDeadCode = addDeadCode(obfuscatedCode, deadCode);

return codeWithDeadCode;

}

function transformationsOfDataStructures(data) {

return jso.obfuscate(data, {

compact: false,

controlFlowFlattening: false,

controlFlowFlatteningThreshold: 0,

numbersToExpressions: false,

simplify: true,

sourceMapMode: 'inline',

stringArrayWrappersChainedCalls: false,

identifierNamesGenerator:'hexadecimal',

deadCodeInjection: false,

deadCodeInjectionThreshold: 0,

renameProperties: false,

renameVariables: false,

stringArrayShuffle: false,

splitStrings: false,

stringArray: true,

stringArrayThreshold: 1

});

}

function compressFile(filePath) {

fs.readFile(filePath, 'utf8', (err, data) => {

if (err) {

console.error('Ошибка при чтении файла:', err);

return;

}

const compressedData = data.replace(/(\/\\*[\s\S]\*?\\*\/|\/\/.\*)|\s/g, '');

const compressedFilePath = filePath.replace('.js', '\_compressed.js');

});

}

function controlFlowTransformations(data) {

return jso.obfuscate(data, {

compact: false,

controlFlowFlattening: true,

controlFlowFlatteningThreshold: 0.5,

numbersToExpressions: false,

simplify: true,

sourceMapMode: 'inline',

stringArrayWrappersChainedCalls: false,

identifierNamesGenerator:'hexadecimal',

deadCodeInjection: true,

deadCodeInjectionThreshold: 1,

renameProperties: false,

renameVariables: false,

stringArrayShuffle: false,

splitStrings: false,

stringArray: true,

stringArrayThreshold: 1

});

}

rl.question('Enter the path to the file for obfuscation: ', (filePath) => {

fs.readFile(filePath, 'utf-8', (err, data) => {

if (err) {

console.error('File reading error:', err);

rl.close();

return;

}

//const obfuscationResult = formattingConversions(data);

//const obfuscationResult = transformationsOfDataStructures(data);

const obfuscationResult = controlFlowTransformations(data);

const obfuscatedFilePath = filePath.replace(/\.js$/, '\_obfuscated.js');

fs.writeFile(obfuscatedFilePath, obfuscationResult.getObfuscatedCode(), (err) => {

if (err) {

console.error('Error writing an obfuscated file:', err);

rl.close();

return;

}

console.log('The file has been successfully obfuscated and saved as', obfuscatedFilePath);

rl.close();

});

});

});