Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 6

Выполнил студент гр. 753503

Муха Д.С.

Проверил

Протько М.И.

**Минск, 2020**

Содержание

[1. Введение 3](#_Toc39239481)

[2. Постановка задач 4](#_Toc39239482)

[3. Теоретические сведения 5](#_Toc39239483)

[4. Результат выполнения работы программы 8](#_Toc39239484)

[5. Вывод 9](#_Toc39239485)

[Примечание. Код программ 10](#_Toc39239486)

# Введение

Целью данных лабораторных работ было:

1. Знакомство с основными технологиями защиты программного

обеспечения от несанкционированного использования.

1. Получение навыков защиты разработанной программы от несанкционированного копирования

# Постановка задач

I. 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:

a) По одному.

b) Любые 2 на выбор из трех одновременно.

c) Все три одновременно.

II. Протестировать работоспособность приложения с запутанным программным кодом.

III. Проверить и пояснить следующие свойства, которым должна удовлетворять запутанная программа:

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

# Теоретические сведения

В результате выполнения лабораторной работы №5 были созданы web-приложения, обеспечивающие аутентификацию пользователей, с разделяемым доступом к информационным ресурсам, которые обеспечивают защиту от основных удаленных атак.

Вместе с тем для любых коммерческих приложений не менее актуальным является решение задачи защиты авторских прав разработчика либо пользователя системы.

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).

**Преобразования форматирования**

К преобразованиям форматирования относятся удаление комментариев, переформатирование программы, удаление отладочной информации, изменение имён идентификаторов.

Удаление комментариев и переформатирование программы применимы, когда запутывание выполняется на уровне исходного кода программы. Эти преобразования не требуют только лексического анализа программы. Хотя удаление комментариев - одностороннее преобразование, их отсутствие не затрудняет сильно обратную инженерию программы, так как при обратной инженерии наличие хороших комментариев к коду программы является скорее исключением, чем правилом. При переформатировании программы исходное форматирование теряется безвозвратно, но программа всегда может быть переформатирована с использованием какого-либо инструмента для автоматического форматирования программ (например, indent для программ на Си).

Удаление отладочной информации применимо, когда запутывание выполняется на уровне объектной программы. Удаление отладочной информации приводит к тому, что имена локальных переменных становятся невосстановимы.

Изменение имён локальных переменных требует семантического анализа (привязки имён) в пределах одной функции. Изменение имён всех переменных и функций программы помимо полной привязки имён в каждой единице компиляции требует анализа межмодульных связей. Имена, определённые в программе и не используемые во внешних библиотеках, могут быть изменены произвольным, но согласованным во всех единицах компиляции образом, в то время как имена библиотечных переменных и функций меняться не могут. Данное преобразование может заменять имена на короткие автоматически генерируемые имена (например, все переменные программы получат имя v<номер> в соответствии с их некоторым порядковым номером). С другой стороны, имена переменных могут быть заменены на длинные, но бессмысленные (случайные) идентификаторы в расчёте на то, что длинные имена хуже воспринимаются человеком.

**Преобразования потока управления**

Преобразования потока управления изменяют граф потока управления одной функции. Они могут приводить к созданию в программе новых функций.

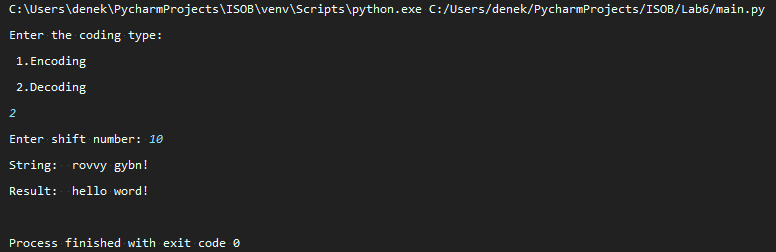
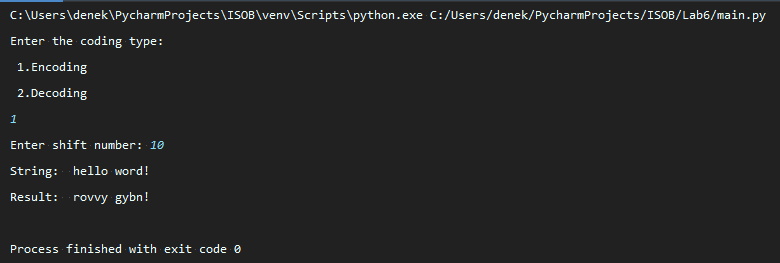
**Внесение недостижимого кода** (adding unreachable code). Если в программу внесены непрозрачные предикаты видов PF или PT, ветки условия, соответствующие условию "истина" в первом случае и условию "ложь" во втором случае, никогда не будут выполняться. Фрагмент программы, который никогда не выполняется, называется недостижимым кодом. Эти ветки могут быть заполнены произвольными вычислениями, которые могут быть похожи на действительно выполняемый код, например, собраны из фрагментов той же самой функции. Поскольку недостижимый код никогда не выполняется, данное преобразование влияет только на размер запутанной программы, но не на скорость её выполнения. Общая задача обнаружения недостижимого кода, как известно, алгоритмически неразрешима. Это значит, что для выявления недостижимого кода должны применяться различные эвристические методы, например, основанные на статистическом анализе программы.

**Внесение избыточного кода** (adding redundant code) [5], п. 6.2.6. Избыточный код, в отличие от мёртвого кода выполняется, и результат его выполнения используется в дальнейшем в программе, но такой код можно упростить или совсем удалить, так как вычисляется либо константное значение, либо значение, уже вычисленное ранее. Для внесения избыточного кода можно использовать алгебраические преобразования выражений исходной программы или введение в программу математических тождеств. Например, можно воспользоваться комбинаторным тождеством и заменить везде в программе использование константы 256 на цикл, который вычисляет сумму биномиальных коэффициентов по приведённой формуле.

# Результат выполнения работы программы

Использование для данной лабораторной работы кода программы, созданного в рамках предыдущих заданий, затруднено, поскольку он либо выполняет связь с базой данных, либо управляет визуальными компонентами, либо является недостаточно длинным. Это мешает грамотной демонстрации. Поэтому было решено использовать код программы, созданной в рамках первой лабораторной работы.

Демонстрация работы программы после применения обфускации



# Вывод

В рамках данной лабораторной работы были изучены основные технологии защиты программного обеспечения от несанкционированного использования. Получены навыки защиты разработанной программы от несанкционированного копирования. Изучены алгоритмы обфускации.

Так же были реализованы 3 метода обфуксации программного кода:

* Метод №1 – переименование переменных.
* Метод №2 – переименование переменных + внесение избыточного кода.
* Метод №3 – переименование переменных + внесение избыточного кода + внесение недостижимого кода

# Примечание. Код программ

dictionary = "abcdefghijklmnopqrstuvwxyz"  
  
*def* read\_file(file\_name):  
 abc = *open*(file\_name)  
 fff = abc.read().lower()  
 *return* fff  
  
  
*def* encrypt\_caesar(strr, n):  
  
 flag = *False  
 for* a *in* strr:  
 *if* a == ',' *or* a == '.':  
 flag = *True  
 break* lolkek = ""  
 *for* i *in* strr:  
 num = dictionary.find(i)  
 *if* num != -1:  
 *if* i == ' ':  
 lolkek += ' '  
 *elif* i == 'A' *and* flag:  
 lolkek -= ' '  
 *else*:  
 lolkek += dictionary[(dictionary.index(i) + n) % *len*(dictionary)]  
 *else*:  
 lolkek += i  
  
 *print*("Result: ", lolkek)  
 kek = *open*('DecodeCaesar.txt', 'w')  
 kek.write(lolkek)  
  
  
*def* decrypt\_caesar(de\_strr, n):  
 lolkek = ""  
 flag = *False  
 for* a *in* de\_strr:  
 *if* a == ',' *or* a == '.':  
 flag = *True  
 break  
  
 for* i *in* de\_strr:  
 num = dictionary.find(i)  
 *if* num != -1:  
 *if* i == ' ':  
 lolkek += ' '  
 *elif* i == 'C' *or* flag:  
 lolkek -= ' '  
 *else*:  
 lolkek += dictionary[(dictionary.index(i) - n) % *len*(dictionary)]  
 *else*:  
 lolkek += i  
 *print*("Result: ", lolkek)  
 lol = *open*('Caesar.txt', 'w')  
 lol.write(lolkek)  
  
  
*def* main():  
 typeCoding = *int*(*input*("Enter the coding type:\n 1.Encoding\n 2.Decoding\n"))  
 *if* typeCoding == 1:  
 strr = read\_file('Caesar.txt')  
 n = *int*(*input*("Enter shift number: "))  
 *print*("String: ", strr)  
 encrypt\_caesar(strr, n)  
 *elif* typeCoding == 2:  
 strr = read\_file('DecodeCaesar.txt')  
 n = *int*(*input*("Enter shift number: "))  
 *print*("String: ", strr)  
 decrypt\_caesar(strr, n)  
  
*if* \_\_name\_\_ == "\_\_main\_\_":  
 main()