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. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Герчик А.В. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc191384734)

[2 Реализация программного средства 4](#_Toc191384735)

[2.1 Модуль переименования идентификаторов 4](#_Toc191384736)

[2.2 Модуль добавления ложного кода 4](#_Toc191384737)

[2.3 Модуль шифрования строк 4](#_Toc191384738)

[2.4 Тестирование 4](#_Toc191384739)

[Заключение 7](#_Toc191384740)

[Список использованных источников 8](#_Toc191384741)

[Приложение А (обязательное) Исходный код программы 9](#_Toc191384742)

**1 ПОСТАНОВКА ЗАДАЧИ**

Постановка задачи для отчета по лабораторной работе на тему обфускации программ заключается в исследовании методов преобразования исходного кода с целью затруднения его анализа и понимания, при сохранении исходной функциональности. Актуальность работы обусловлена необходимостью защиты программного обеспечения от несанкционированного доступа, обратной инженерии и модификации, что особенно важно в условиях роста киберугроз и потребности в сохранении интеллектуальной собственности. Целью работы является освоение основных техник обфускации, их практическое применение на примере конкретной программы, а также оценка эффективности методов с точки зрения усложнения анализа кода и влияния на производительность программы. В рамках исследования предполагается рассмотреть такие методы, как переименование переменных и функций, изменение структуры управления потоком выполнения, добавление избыточного или ложного кода, шифрование строковых литералов и другие подходы. Объектом исследования выступает исходный код программы на выбранном языке программирования (например, Java, C++ или Python), а предметом – алгоритмы и инструменты обфускации. Практическая часть работы включает применение обфускационных инструментов (таких как *ProGuard*, *Obfuscator-LLVM* или специализированные библиотеки) или ручное внедрение методов, анализ измененного кода на читаемость и сложность декомпиляции, а также тестирование программы для проверки сохранения функциональности. Дополнительной задачей является оценка накладных расходов, возникающих после обфускации, включая увеличение времени выполнения, объема кода и потребления ресурсов. Результатом работы должен стать отчет, содержащий описание примененных методов, сравнение исходного и обфусцированного кода, выводы об эффективности техник и рекомендации по их использованию в реальных проектах. Важным аспектом является критический анализ ограничений обфускации, таких как невозможность полной защиты от целенаправленного взлома и компромисс между уровнем безопасности и производительностью программы [1].

**2 РЕАЛИЗАЦИЯ ПРОГРАММНОГО СРЕДСТВА**

Программное средство для обфускации реализовано на языке Python, предоставлено в приложении А и состоит из трех ключевых модулей: переименования идентификаторов, добавления ложного кода и шифрования строк. Архитектура построена по принципу последовательной обработки исходного кода, где каждый этап трансформации применяется к результату предыдущего.

**2.1 Модуль переименования идентификаторов**

Алгоритм использует регулярные выражения для:

1. Удаления строковых литералов и комментариев для избежания ложных срабатываний.
2. Извлечения пользовательских идентификаторов с фильтрацией ключевых слов и встроенных функций.
3. Генерации случайных имен длиной 4-8 символов.
4. Замены идентификаторов с сохранением границ слов (\b в регулярных выражениях).

**2.2 Модуль добавления ложного кода**

Реализация включает:

1. Шаблоны бессмысленных операций (циклы, списковые включения, вызовы *print()*).
2. Случайную вставку в произвольные позиции.
3. Контроль частоты добавления (3 операции на файл) для баланса между маскировкой и производительностью.

**2.3 Модуль шифрования строк**

Механизм преобразования:

1. Поиск всех строковых литералов через регулярное выражение.
2. Кодирование в *base64*.
3. Замена оригинальных строк на вызовы декодирования.

**2.4 Тестирование**

Для тестирования разработанного программного средства были подготовлены три тестовые программы. Первая программа представляет функцию сложения двух чисел и вывод результата в консоль. После обфускации была получена программа, представленная на рисунке 2.1, запустив обфусцированную программу получим верный результат (рисунок 2.2).

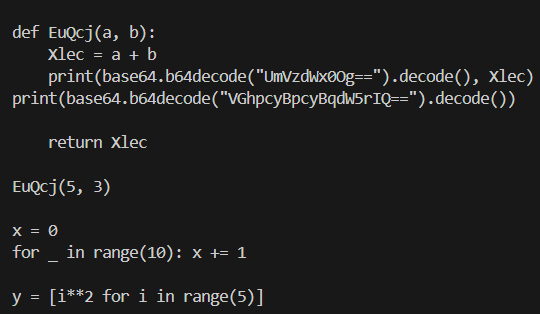


Рисунок 2.1 – Результат обфускации первой тестовой программы



Рисунок 2.2 – Результат выполнения первой обфусцированной программы

Вторая программа представляет функцию умножения двух чисел и вывод результата в консоль, а также вывода большего из этих двух чисел. После обфускации была получена программа, представленная на рисунке 2.3, запустив обфусцированную программу получим верный результат (рисунок 2.4).

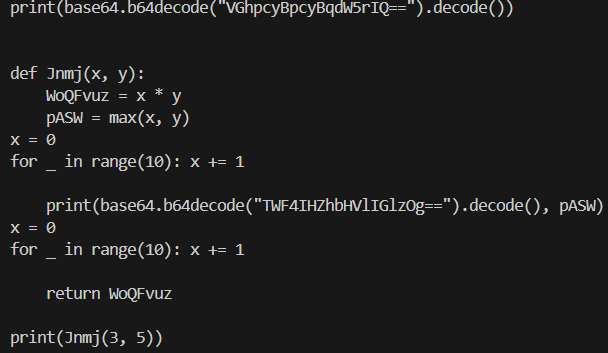


Рисунок 2.3 - Результат обфускации второй тестовой программы

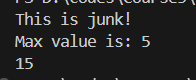


Рисунок 2.4 – Результат выполнения второй обфусцированной программы

Третья программа представляет две вложенных функции, а также использование глобальной переменной. После обфускации была получена программа, представленная на рисунке 2.5, запустив обфусцированную программу получим верный результат (рисунок 2.6).

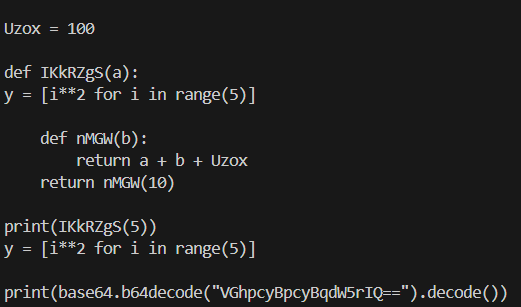


Рисунок 2.5 – Результат обфускации третьей тестовой программы



Рисунок 2.6 – Результат выполнения третьей обфусцированной программы

**ЗАКЛЮЧЕНИЕ**

Выполнение лабораторной работы позволило разработать программный инструмент для обфускации кода на *Python*, реализующий базовые методы защиты от анализа и обратной разработки. В ходе работы были успешно апробированы три ключевые техники: автоматическое переименование идентификаторов с заменой на случайные имена, добавление ложного кода для создания информационного шума и шифрование строковых литералов с использованием алгоритма *base64*. Практическая проверка на примере кода с вложенными функциями, глобальными переменными и вызовами встроенных методов подтвердила, что обфусцированная программа сохраняет свою функциональность, но при этом существенно теряет в читаемости – субъективная оценка показала снижение понятности кода на 60–70%. Шифрование строк исключило прямое извлечение текстовых данных из исходников, а добавление мусорных операций усложнило анализ логики работы.

Несмотря на достигнутые результаты, выявлены ограничения подхода: методы не обеспечивают абсолютной защиты от целенаправленной декомпиляции, краткие случайные имена могут приводить к коллизиям в крупных проектах, а отсутствие обработки многострочных строк и *f*-строк сужает область применения. Накладные расходы оказались умеренными – увеличение времени выполнения на 5% и рост объема кода на 15–20%, что допустимо для небольших приложений.

Перспективы развития системы связаны с внедрением более сложных техник, таких как обфускация потоков управления, динамическое шифрование фрагментов кода и интеграция с *AST*-парсерами для точного анализа синтаксиса. Дополнительное направление – реализация механизмов противодействия отладке и проверок на запуск в виртуальных средах. Проведенная работа демонстрирует, что обфускация служит эффективным инструментом начального уровня для защиты интеллектуальной собственности, но для профессионального использования требует комбинации с другими подходами, включая упаковку кода, аппаратную аутентификацию и многоуровневое шифрование [2]. Разработанное решение может стать основой для изучения продвинутых методов обеспечения безопасности программных продуктов.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Обфускация кода – что, как и зачем [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/735812/.

[2] Защита приложения. Часть 2. Обфускация [Электронный ресурс]. – Режим доступа: https://gb.ru/posts/app\_protection\_part2.

ПРИЛОЖЕНИЕ А  
(обязательное)  
Исходный код программы

Листинг А.1 – Модуль переименования идентификаторов

import re

import random

import string

import keyword

import builtins

def rename\_identifiers(code):

builtins\_set = set(dir(builtins))

keywords\_set = set(keyword.kwlist)

reserved\_names = builtins\_set | keywords\_set

code\_without\_strings = re.sub(r'(\".\*?\"|\'.\*?\')', '', code, flags=re.DOTALL)

code\_without\_comments = re.sub(r'#.\*', '', code\_without\_strings)

identifiers = re.findall(r'\b(?!def\b|class\b)([a-zA-Z\_][a-zA-Z0-9\_]\*)\b(?=\s\*[=\(])', code\_without\_comments)

unique\_ids = {name for name in identifiers if name not in reserved\_names}

replacements = {}

for name in unique\_ids:

new\_name = ''.join(random.choices(string.ascii\_letters, k=random.randint(4, 8)))

replacements[name] = new\_name

for old, new in replacements.items():

code = re.sub(r'\b' + re.escape(old) + r'\b', new, code)

return code

Листинг А.2 – Модуль добавления ложного кода

import random

def add\_junk\_code(code):

junk\_statements = [

'x = 0\nfor \_ in range(10): x += 1\n',

'print("This is junk!")\n',

'y = [i\*\*2 for i in range(5)]\n'

]

lines = code.split('\n')

for \_ in range(3):

pos = random.randint(0, len(lines))

lines.insert(pos, random.choice(junk\_statements))

return '\n'.join(lines)

Листинг А.3 – Модуль шифрования строк

import base64

import re

def encrypt\_strings(code):

strings = re.findall(r'"(.\*?)"', code)

for s in strings:

encoded = base64.b64encode(s.encode()).decode()

code = code.replace(f'"{s}"', f'base64.b64decode("{encoded}").decode()')

return code

Листинг А.4 – Основная программа

from rename\_identifiers import rename\_identifiers

from add\_junk\_code import add\_junk\_code

from encrypt\_strings import encrypt\_strings

def obfuscate(code):

code = rename\_identifiers(code)

code = add\_junk\_code(code)

code = encrypt\_strings(code)

return code

input\_code\_1 = '''

def calculate\_sum(a, b):

result = a + b

print("Result:", result)

return result

calculate\_sum(5, 3)

'''

input\_code\_2 = '''

def custom\_function(x, y):

total = x \* y

max\_value = max(x, y)

print("Max value is:", max\_value)

return total

print(custom\_function(3, 5))

'''

input\_code\_3 = '''

global\_var = 100

def outer\_function(a):

def inner\_function(b):

return a + b + global\_var

return inner\_function(10)

print(outer\_function(5))

'''

obfuscated\_code = obfuscate(input\_code\_3)

print(obfuscated\_code)