1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. Институт компьютерных наук и кибербезопасности

**ЛАБОРАТОРНАЯ РАБОТА № 6**

1. «**Защита программного обеспечения от нелегального использования**»
2. по дисциплине «Основы информационной безопасности»
3. Выполнил
4. студент гр. 5151001/40001 Волошкевич М.А.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Орел Е.М.

<*подпись*>

1. Санкт-Петербург
2. 2025

# Цель работы

Приобретение навыков по защите приложений от нелегального использования, по анализу исполняемых ходов в отсутствие исходных текстов и по применению способов защиты программ от дизассемблирования и отладки.

# Постановка задачи

1. Создать простую программу на языке С запрашивающую пароль
2. Пользуясь исходным текстом программы и утилитой HIEW изменить в исполняемом файле команду перехода при проверке правильности пароля. Проверить работу исполняемого файла при вводе любых данных.
3. Модифицировать программу, чтобы исключить открытое хранение пароля, используя XOR шифрование. Повторить пункт 2.
4. Модифицировать программу Nag4.exe, чтобы при работе с ней не активизировалось всплывающее окно.
5. Модифицировать программу Guard2.exe, чтобы проверка пароля не осуществлялась.
6. Для разработанной программы в п.1. реализовать обфускатор на уровне исходных кодов.

# Теоретические исследования

Простейшим способом проверки легальности копии программы является запрос у пользователя учетных данных (например, пароля, номера лицензии, ключа), выданных ему при покупке. Если пользователем введены корректные данные, функция проверки разрешает продолжить работу с программой.

Обойти проверку пароля можно изменив ассемблерную команду условного перехода, например «jz», на команду безусловного перехода «jmp» или на другие противоположные команды условного перехода, например «jnz».

Защитой от взлома является запрет на открытое хранение пароля во внешнем файле: доступ к файлу не дает никакой информации о самих паролях. Пароль хранится в зашифрованном виде. Простейшая схема сокрытия пароля – использование логического «ИСКЛЮЧАЮЩЕГО ИЛИ».

Для обхода механизмов защиты и для анализа поведения программ с целью выявления нарушений применяются программы-дизассемблеры и отладчики. В простейшем случае, как и при проверке пароля, нарушителю необходимо исправить один байт, что свидетельствует о чрезвычайной ненадежности такого способа защиты программ.

Для того чтобы противодействовать модификации исполняемого файла, разработчики используют проверку целостности программ (например, с помощью контрольного суммирования). Кроме того, существует множество методов по усложнению дизассемблирования и анализа алгоритмов программ.

Простейшим методом от дизассемблирования является применение оптимизирующего компилятора. В оптимизированном исполняемом файле нарушается прямой порядок вызова команд и использования регистров. Это несколько усложняет процесс анализа функций и передачи управления внутри кода программы. В современных компиляторах оптимизация исполняемого кода включена по умолчанию.

Традиционный способ защиты программ применение шифрования или упаковки распространяемого программного обеспечения. В этом случае собственно исполняемый код не соответствует тексту программы в открытом виде. Однако если программа во время исполнения сама себя дешифрует или распаковывает, то она же содержит код, выполняющий действия по дешифрации или распаковке. Причем этот код доступен злоумышленнику, и, следовательно, выполнив его, можно получить в памяти исходный исполняемый код. Поэтому в программах, требующих хорошего уровня защиты от дизассемблирования, используется метод неполного динамического шифрования дешифрования, таким образом, программа никогда не содержится в памяти, расшифрованной целиком.

В большинстве случаев получение исходного ассемблерного кода не дает достаточно информации о механизме защиты. Тогда с помощью специальных утилит-отладчиков можно запустить анализируемую исполняемую программу в режиме отладки. Простейшим методом защиты от запуска в режиме отладки является проверка наличия отладчика (например, функция isDebugPresent).

Существует метод защиты программного обеспечения, основанный на запутывании программного кода, обфускация (оbfuscate сбивать с толку). При этом исходный текст или исполняемый код программы приводится к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание и модификацию алгоритма при дизассемблировании. Для создания обфусцированного кода могут применяться специализированные компиляторы, использующие неочевидные и недокументированные возможности, или специальные программы-обфускаторы.

При программировании на интерпретируемых скрипт-языках (например, JavaScript, VBScript) пользователю доступен исходный текст программы. Обфускация в таком случае сводится к форматированию текста и замене имен переменных и функций, что направлено на то, чтобы сделать текст менее читаемым, и, соответственно, менее понятным нарушителям.

Обфускация на уровне машинного кода заключается в добавлении блоков с необязательным исполнением, в перемешивании и внесении "мусорных" функциональных блоков программы с целью осложнения анализа код, но не нарушения логики работы программы. Код независимых функциональных блоков вместо тиражирования может, наоборот, объединяться. Таким образом, граф исполнения программы принимает новый вид, в котором передача управления выполняется через новую цепочку и не очевидным образом. В этой связи негативное следствие обфускации-замедление работы программы.

При обфускации машинного кода также используется скрытие констант и данных, когда константа формируется во время исполнения и не встречается в открытом виде. Константные данные могут скрываться при помощи шифрования.

В настоящее время ведутся разработки обфускаторов, использующих виртуальные процессоры. В этом случае создается случайная виртуальная машина со случайным набором инструкций, и весь код программы трансформируется под нее.

Обфускация помогает сделать программу более защищенной от отладки и анализа, но ни один из существующих обфускаторов не гарантирует абсолютной невозможности восстановления логики работы программы.

# Описание решения

## Программа, запрашивающая пароль

На языке С была реализована программа, запрашивающая у пользователя пароль. Пароль хранится в отдельном файле, а проверка пароля реализована с помощью отдельной функции.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Введен неверный пароль



Рисунок 2 – Введен верный пароль

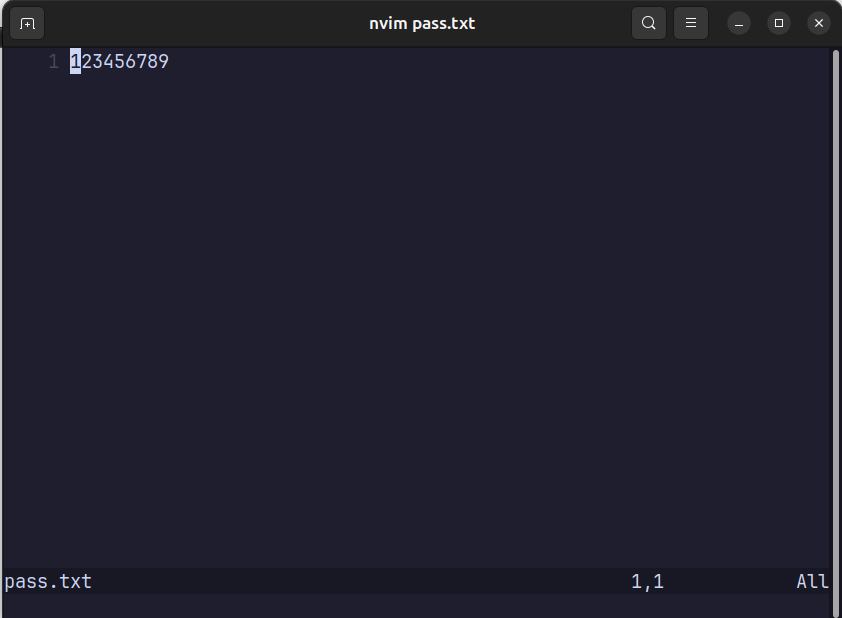


Рисунок 3 – Пароль, который хранится в файле

С помощью программы IDA был найден участок кода с командой перехода проверки на правильность пароля. Для того, чтобы при вводе любого пароля, программа давала доступ к программе, при переходе условного перехода «jz», на команду безусловного перехода «jmp»

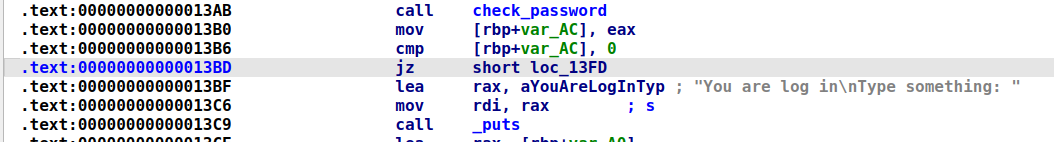


Рисунок 4 – Участок дизассемблированного кода, который содержит команду «jz»

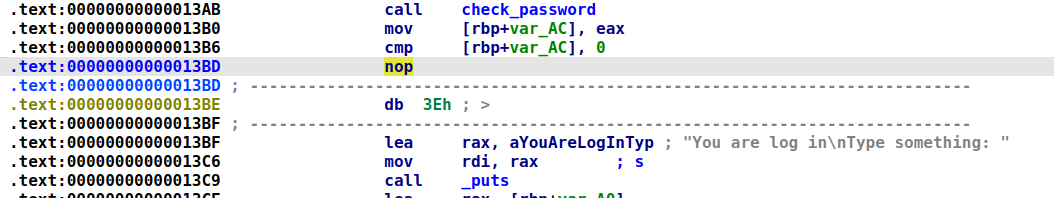


Рисунок 5 – Рисунок 3 Участок дизассемблированного кода, команду «jz» заменим на «nop»

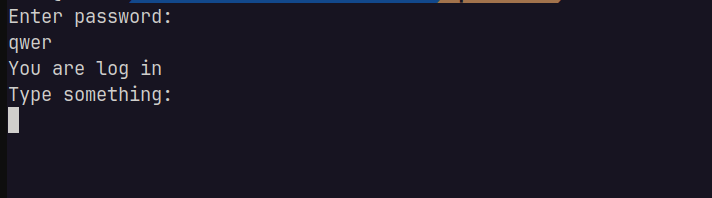


Рисунок 6 – Пароль верен при любых входных данных

Программа запроса пароля была модифицирована таким образом, чтобы исключить открытое хранение пароля. Для этого была создана процедура XOR(), строка пароля зашифрована путем применения побитового оператора XOR к каждому символу с использованием заданного ключа (в данном случае ConstXOR=100). Однако данный шифр не исправил ситуации. При выполнении действий, описанных выше, и изменения дизассемблированного кода при вводе любого набора символов, пароль считается верным.



Рисунок 7 – Пароль верен при любых входных данных, программа использует шифр XOR

## Работа с программой Nag4.exe

При запуске программы Nag4.exe открывается окно, в котором можно напечатать текст. Однако через 10 секунд открывается окно с отсчетом. Пока отсчет не закончится, программу нельзя закрыть, в ней ничего нельзя написать.

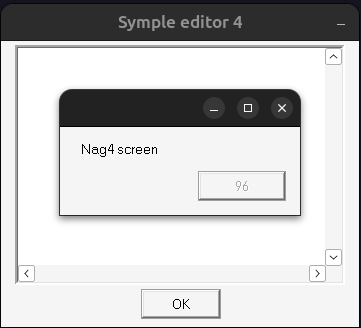


Рисунок 8 – Работа программы Nag4.exe

За вызов данного окна отвечает функция SetTimer. Ее вызов можно найти с помощью поиска в IDA. Как видно из Рисунка 9 вызов функции осуществляется по адресу 0040195А.

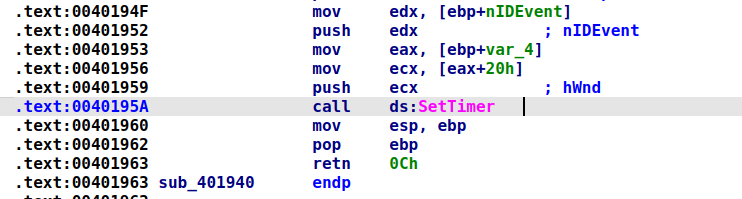


Рисунок 9 – При помощи поиска находим нужный участок кода. Функция SetTimer.

Изначально установлена команда ассемблера «call», то есть осуществляется переход в данную функцию. Если изменить эту команду на «nop». При вызове данной команды не происходит никаких действий.

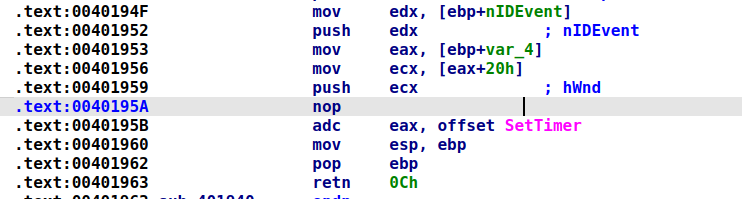


Рисунок 10 – При функции SetTimer была установлена команда "nop"

Теперь при работе с программой Nag4.exe окно со счетчиком не появляется.

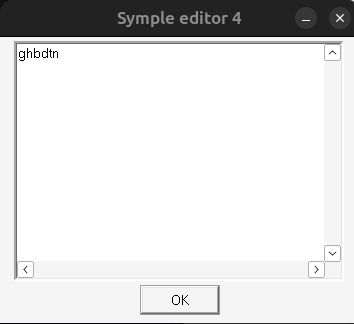


Рисунок 11 – Программа Nag4.exe работает без вызова всплывающего окна

## Работа с программой GUARD\_2.exe

Данная программа просит ввести пароль. При вводе неверного пароля программа выдает ошибку.

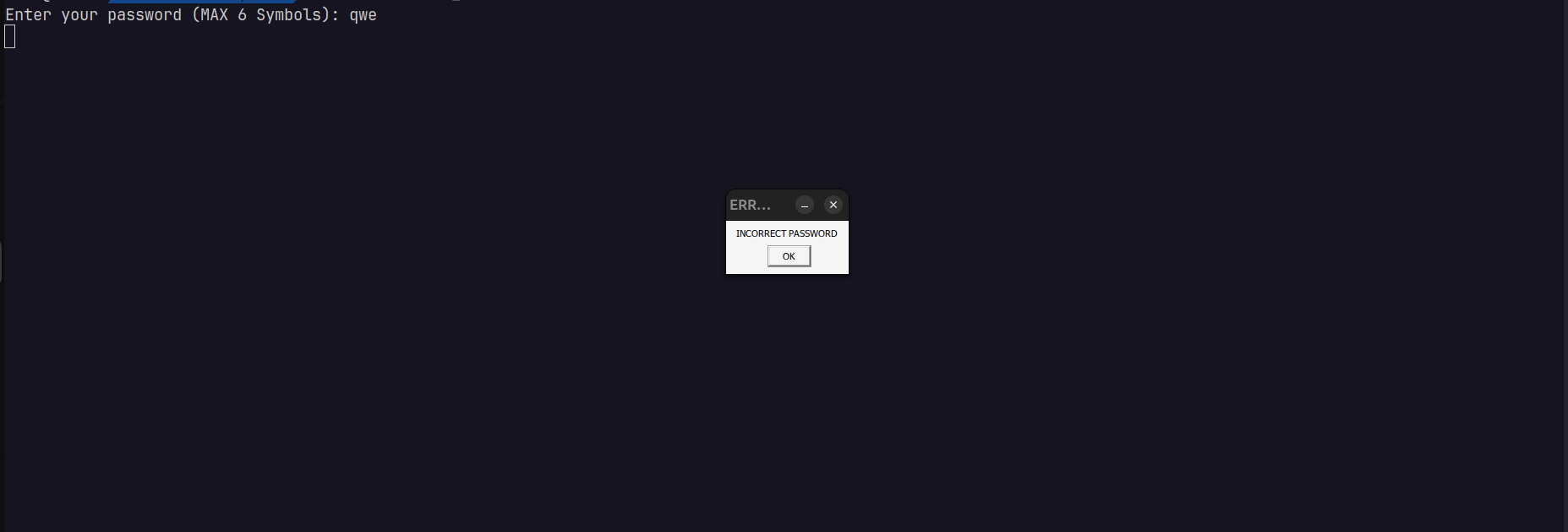


Рисунок 12 – Ввод неверного программе GUARD\_2.exe

С помощью утилиты Peid был определен упаковщик. Это видно на Рисунке 13 в строке «EP Section». Упаковка данного файла: UPX.

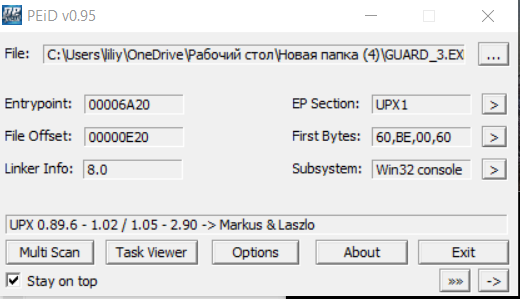


Рисунок 13 – Тип упаковки можно узнать с помощью утилиты Peid

С помощью утилиты upx была сделана распаковка данной программы. Для этого файла .exe был загружен в данную утилиту c флагом -d и сохранен. Таким образом выполнена распаковка файла.

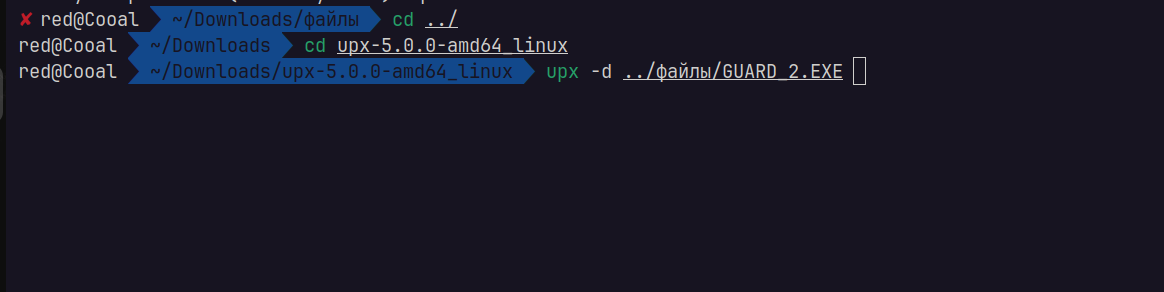


Рисунок 14 – С помощью утилиты upx была реализована распаковка

Ниже представлен участок кода с проверкой пароля.

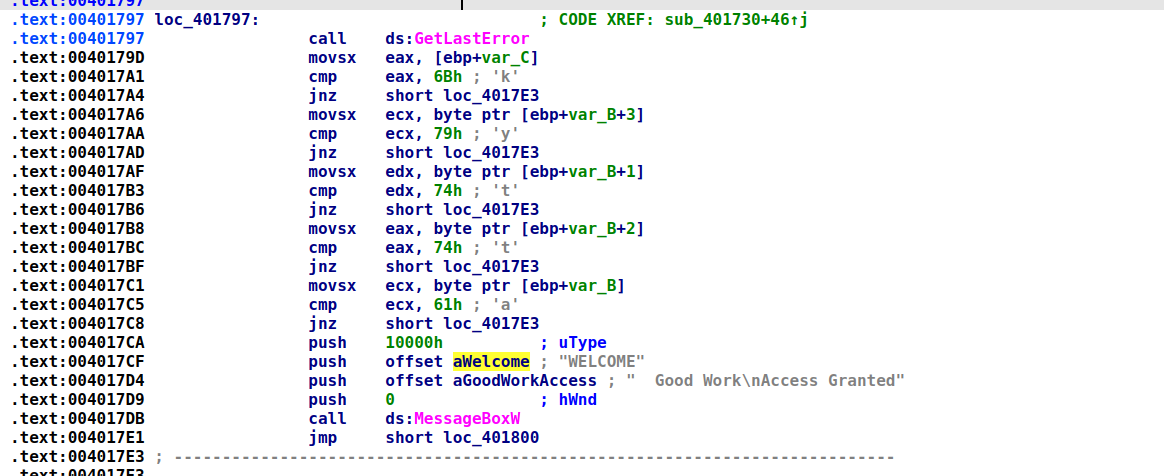


Рисунок 15 – Участок кода с проверкой пароля

Были изменены пять ассемблерных инструкций «jnz» на «nop».

Рисунок 16 – Изначальный код

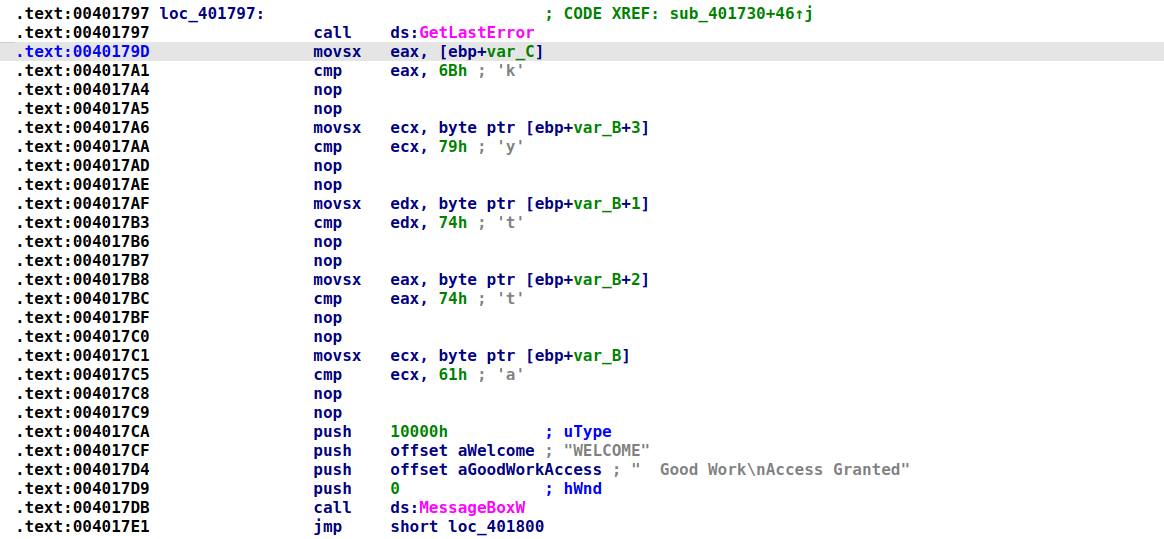


Рисунок 17 – Замена команд "jnz" на "nop"

Тогда при любых входных данных пароль верный.

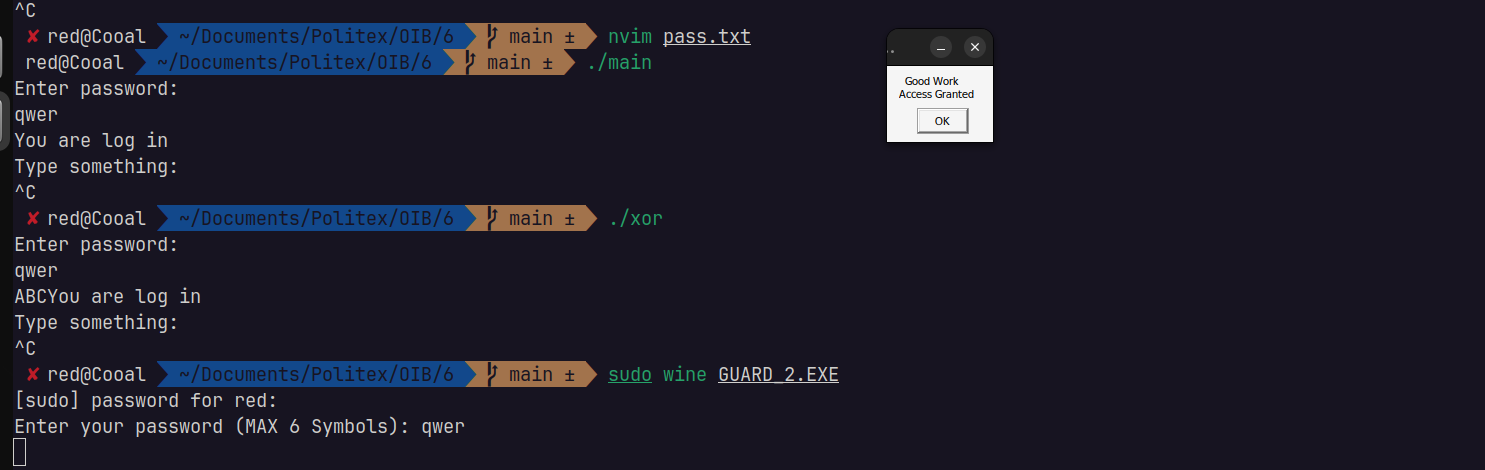


Рисунок 18 – При любых входных данных пароль считается верным

## Программа-обфускатор

Исходный код программы представлен в приложении Б.

Программа начинается с загрузки конфигурации из файла obfuscator.cfg с помощью функции load\_config(). В конфигурации указываются параметры обфускации, такие как пути к входному и выходному файлам, флаги удаления комментариев, пробелов, замены переменных, добавления "мусорного" кода и другие настройки.

### Удаление комментариев

Если в конфигурации установлен флаг remove\_comments, вызывается функция remove\_comments(), которая удаляет все однострочные (//) и многострочные (/\* ... \*/) комментарии, сохраняя при этом строковые литералы в кавычках. Для этого используется регулярное выражение с заменой через callback-функцию.

### Удаление пробелов, табуляции и переносов строки

При активации флага remove\_spaces функция remove\_spaces() обрабатывает код, удаляя лишние пробелы, табуляции и переносы строк. Строки, начинающиеся с # (например, директивы препроцессора), сохраняются без изменений, включая обязательный перенос строки. Остальные пробелы сжимаются, а ненужные пробелы вокруг операторов и пунктуации удаляются.

### Замена переменных и функций

Если включена опция replace\_variables, функция replace\_variables() находит все имена переменных и функций (кроме main и стандартных типов) и заменяет их на случайно сгенерированные имена с префиксом из конфигурации. Для корректной работы строковые литералы временно заменяются метками, а после обработки восстанавливаются.

### Перемешивание функций

При активации shuffle\_functions код анализируется на наличие функций с помощью find\_function(), после чего их порядок случайным образом изменяется (за исключением структур, которые остаются в начале).

### Добавление мусорного кода

Если в конфигурации разрешено add\_garbage, в код добавляются бессмысленные переменные, циклы и функции с помощью garbage\_var(), garbage\_loop() и garbage\_func(). Количество мусорных элементов регулируется настройками.

### Запись в файл

После всех преобразований обработанный код записывается в выходной файл, указанный в конфигурации с помощью открытия файла в режиме записи.

Программа была протестирована на работоспособность — после обфускации код продолжает выполнять свои функции, но его структура и имена переменных становятся неочевидными.

# Ответы на контрольные вопросы

## Какие методы применяются для защиты коммерческих программ от ввода некорректных учетных данных?

Для защиты коммерческих программ от ввода некорректных учетных данных часто используются методы запроса пароля и другие методы, которые требует аутентификацию пользователя, запрет на открытое хранение пароля.

## По каким признакам можно найти интересующий код сопоставления с паролем в программе?

Найти интересующий код сопоставления с паролем в программе можно с помощью команды CMP и команды условного перехода.

## Какие существуют основные методы защиты от дизассемблирования и отладки программ?

К основным методам защиты от дизассемблирования и отладки программ является применение оптимизирующего компилятора, проверки целостности программы, применение упаковки, применение шифрования, обфускация кода.

## Как реализуется обфускация с помощью виртуальных машин?

Обфускация с помощью виртуальных машин реализуется созданием случайно виртуальной машины со случайным набором инструкций, а весь код трансформируется под нее.

## Предложите методы усиления механизмов защиты программ от нелегального использования, исследованные в данной работе.

Чтобы усилить механизмы защиты программ от нелегального использования, можно шифровать данные учетной записи пользователя, использовать проверку на присутствие отладчика, использовать всплывающие окна, использовать большое количество ссылок на разные участки кода.

# Выводы

В ходе данной работы были изучены различные методы защиты программ от нелегального использования. При выполнении работы была разработана программа, требующая пароля. При модификации программы было изучено XOR-шифрование. При работе с программами были получены практические навыки по анализу исполняемого кода без исходного текста с помощью дизассемблера. При модификации программ были изучены некоторые команды дизассемблера, а также получены навыки работа с ними. Были изучены методы защиты программы, такие как проверка на присутствие отладчика, использование всплывающих окон.

Был также изучен метод обфусцирования. При разработке программы-обфускатора возникли проблемы с реализацией поиска и замены переменных, однако они были решены так, что наименование всех переменных записывалось, а после каждое слово сравнивалось с названием переменной. В случае совпадения заменялось на другое имя переменной.

ПРИЛОЖЕНИЕ А

Листинг программы «Ввод пароля, используя шифр XOR»

#include <stdio.h>

#include <string.h>

#define Const\_XOR 50

int check\_password(const char \* pass, char \* true\_pass)

{

for (int i = 0; i < strlen(true\_pass); i++)

{

true\_pass[i] ^= Const\_XOR;

}

int res = strncmp(pass, true\_pass, strlen(pass));

return res == 0 && strlen(pass) == strlen(true\_pass);

}

int main(int argc, char const \*argv[])

{

int is\_login = 0;

char pass[50];

char true\_pass[50];

FILE \*fptr;

fptr = fopen("passxor.txt", "r");

if (fptr == NULL)

{

printf("The file is not opened. The program will now exit.");

return 0;

}

while (fgets(true\_pass, 50, fptr) != NULL);

fclose(fptr);

printf("Enter password: \n");

scanf("%s", pass);

while (1)

{

is\_login = check\_password(pass, true\_pass);

if (is\_login)

{

char test[20];

printf("\nYou are log in\nType something: \n");

scanf("%s", test);

printf("%s\n", test);

}

else

{

printf("\nYou are not log in\nTry again:\n");

scanf("%s", pass);

}

}

return 0;

}

ПРИЛОЖЕНИЕ Б

Листинг программы «Программа-обфускатор»

import configparser

import re

import random

import string

c\_words = ["struct", "typedef", "unsigned", "signed", "enum", "union", "const", "return"]

data\_types = ["char", "float", "int", "long", "short", "double", "size\_t", "void", "define"]

def load\_config(config\_file):

config = configparser.ConfigParser()

config.read(config\_file)

return {

'input\_file': config.get('DEFAULT', 'input\_file', fallback='test.c'),

'output\_file': config.get('DEFAULT', 'output\_file', fallback='compressed.c'),

'remove\_comments': config.getboolean('DEFAULT', 'remove\_comments', fallback=True),

'remove\_spaces': config.getboolean('DEFAULT', 'remove\_spaces', fallback=True),

'replace\_variables': config.getboolean('DEFAULT', 'replace\_variables', fallback=True),

'shuffle\_functions': config.getboolean('DEFAULT', 'shuffle\_functions', fallback=True),

'name\_prefix': config.get('DEFAULT', 'name\_prefix', fallback='x\_'),

'add\_garbage': config.getboolean('DEFAULT', 'add\_garbage', fallback=False),

'garbage\_functions': config.getint('DEFAULT', 'garbage\_functions', fallback=5),

'garbage\_variables': config.getint('DEFAULT', 'garbage\_variables', fallback=5),

'garbage\_loops': config.getint('DEFAULT', 'garbage\_loops', fallback=5),

'garbage\_iterations': config.getint('DEFAULT', 'garbage\_iterations', fallback=10)

}

def remove\_comments(code):

pattern = r'''

("(?:\\.|[^"\\])\*")| # строки в двойных кавычках

('(?:\\.|[^'\\])\*')| # строки в одинарных кавычках

(//[^\n]\*|/\\*.\*?\\*/) # комментарии

'''

def replacer(match):

if match.group(1): return match.group(1)

if match.group(2): return match.group(2)

return ''

return re.sub(pattern, replacer, code, flags=re.VERBOSE | re.DOTALL)

def remove\_spaces(code):

lines = code.split('\n')

result = []

for line in lines:

line = line.strip()

if line.startswith('#'):

result.append(line + '\n')

else:

if line:

result.append(line.rstrip(' \t') + ' ')

code = ''.join(result)

all\_keywords = [re.escape(word) for word in c\_words + data\_types]

keyword\_pattern = r'(?:\b(?:' + '|'.join(all\_keywords) + r')\b\s\*)+'

# Корректно экранированные шаблоны

pattern = fr'''

(".\*?"|'.\*?') # 1. Строки в кавычках

| # ИЛИ

(//.\*?$|/\\*.\*?\\*/) # 2. Комментарии

| # ИЛИ

({keyword\_pattern} # 3. Последовательность ключевых слов

(?:\\*\s\*)\* # возможные указатели

\w+ # идентификатор

)

| # ИЛИ

([ ]+) # 4. Обычные пробелы

'''

def replacer(m):

if m.group(1): return m.group(1) # Сохраняем строки

if m.group(2): return m.group(2) # Сохраняем комментарии

if m.group(3): return m.group(3) # Сохраняем ключевые слова с пробелами

return ' ' if m.group(4) else '' # Сжимаем пробелы до одного

# Применяем замену

code = re.sub(pattern, replacer, code, flags=re.VERBOSE | re.DOTALL | re.MULTILINE)

# Дополнительная очистка пробелов вокруг операторов (с корректным экранированием)

operator\_pattern = r'''

[ ]\* # Любые пробелы

(

[=+\-/\*%&|^<>!]\*= # Операторы присваивания

|\+\+ |-- # Инкремент/декремент

|&& |\|\| # Логические операторы

|-> |\. # Операторы доступа

|[,;:()\[\]{}] # Пунктуация

)

[ ]\* # Любые пробелы

'''

code = re.sub(operator\_pattern, r'\1', code, flags=re.VERBOSE)

return code

def garbage\_var(config):

var\_type = random.choice(["int", "double", "float", "char"])

var\_name = generate\_variable(config)

var\_value = random.choice([

f'\'{random.choice(string.ascii\_lowercase)}\'',

f'{random.randint(0, 100)}',

f'{random.randint(100, 200) / 100 }'

])

return f"\n{var\_type} {var\_name} = {var\_value};"

def garbage\_loop(config):

var\_name = generate\_variable(config)

iterations = config["garbage\_iterations"]

loop\_type = random.choice(['for', 'while', 'do'])

if loop\_type == 'for':

return f"\nfor(int {var\_name} = 0; {var\_name} < {iterations}; {var\_name}++) {{}}\n"

elif loop\_type == 'while':

return f"\nint {var\_name} = 0; \nwhile({var\_name} < {iterations}){{ {var\_name}++; }}\n"

else:

return f"\nint {var\_name} = 0; \ndo{{{var\_name}++;}}while({var\_name} < {iterations});\n"

def garbage\_func(config):

func\_name = generate\_variable(config)

func\_type = random.choice(["int", "double", "void"])

var\_count = config["garbage\_variables"]

loop\_count = config["garbage\_loops"]

func\_vars = ""

for \_ in range(0, var\_count):

func\_vars += garbage\_var(config)

func\_loops = ""

for \_ in range(0, loop\_count):

func\_loops += garbage\_loop(config)

func\_return = ""

if func\_type != "void":

func\_return = f"return {random.randint(1, 100)};"

return f"\n{func\_type} {func\_name} () \n{{ {func\_vars} {func\_loops} {func\_return} }}\n"

def add\_garbage(code, config):

func = find\_function(code)

parts = []

parts.append(code[:func[0][1]])

for \_ in range(0, config["garbage\_functions"]):

parts.append(garbage\_func(config))

for i in func:

parts.append(code[i[1]:i[2]])

new\_code = ""

for part in parts:

new\_code += part

return new\_code

def find\_function(code):

functions = []

stack = 0

func\_name = None

func\_start = None

i = 0

n = len(code)

while i < n:

# Ищем начало объявления функции

if stack == 0:

match = re.match(

r'(?:[\w\s\*]+)\s+' # возвращаемый тип

r'(\w+)\s\*' # имя функции (группа 1)

r'\([^)]\*\)\s\*' # параметры

r'(?:{[^{}]\*})?', # возможное тело для однострочных функций

code[i:i+100] # ограничиваем длину проверки

)

if match:

func\_name = match.group(1)

func\_start = i

while i < n and code[i] != '{':

i += 1

if i >= n:

break

stack = 1

i += 1

continue

if stack > 0:

if code[i] == '{':

stack += 1

elif code[i] == '}':

stack -= 1

if stack == 0 and func\_name:

functions.append((func\_name, func\_start, i + 1))

func\_name = None

func\_start = None

i += 1

return functions

def shuffle\_functions(code):

funcs = find\_function(code)

parts = []

prev = 0

for func in funcs:

if prev != func[1]: parts.append([False, code[prev:func[1]]])

parts.append([True, code[func[1]:func[2]]])

prev = func[2]

if prev < len(code):

parts.append([False, code[prev:]])

to\_shuffle = [part for part in parts if part[0]]

random.shuffle(to\_shuffle)

result = []

shuffle\_idx = 0

structs = 1

for part in parts:

if part[0]:

result.append(to\_shuffle[shuffle\_idx][1])

shuffle\_idx += 1

else:

result.append(part[1])

if "struct" in part[1] and len(result) > 1:

temp = result[structs]

result[structs] = part[1]

result[len(result) - 1] = temp

structs += 1

return ''.join(result)

def generate\_variable(config):

var = config["name\_prefix"] + ''.join(random.choice(string.ascii\_letters) for \_ in range(10))

return var

def replace\_variables(code, config):

code = remove\_comments(code)

replacements = {}

# Запомнить все строковые литералы и заменить их временными метками

string\_literals = []

def replace\_string(match):

string\_literals.append(match.group(0))

return f'\_\_STRING\_LITERAL\_{len(string\_literals)-1}\_\_'

code\_without\_strings = re.sub(r'"(?:\\.|[^"\\])\*"', replace\_string, code)

type\_pattern = r'\b(?:unsigned|const|static|extern|auto)\b'

ptr\_pattern = r'\\*+\s\*'

identifier\_pattern = r'[a-zA-Z\_][a-zA-Z0-9\_]\*'

# Шаблон для структур

struct\_pattern = re.compile(

rf'\bstruct\s+({identifier\_pattern})\s\*'

rf'(?:\{{\s\*[^}}]\*\}})?'

rf'\s\*;?'

)

# Шаблон для объявлений переменных и функций

declaration\_pattern = re.compile(

rf'(?:{type\_pattern}\s+)\*'

rf'(?:{type\_pattern}\s+)\*'

rf'(?:{identifier\_pattern}\s+)+'

rf'(?:{ptr\_pattern})?'

rf'({identifier\_pattern})'

rf'(?:\s\*\[[^\]]\*\])?'

rf'(?:\s\*=\s\*[^;,]\*)?'

rf'(?:\s\*\([^)]\*\))?'

)

# Обрабатываем структуры

structs = []

for match in struct\_pattern.finditer(code\_without\_strings):

struct\_name = match.group(1)

if struct\_name not in replacements:

replacements[struct\_name] = generate\_variable(config)

structs.append(struct\_name)

data\_types.append(struct\_name)

# Обрабатываем другие объявления

for match in declaration\_pattern.finditer(code\_without\_strings):

original\_name = match.group(1)

if (original\_name not in replacements and original\_name != "main" and

not re.match(r'\b(?:int|char|float|double|void|short|long|signed|unsigned|const|static|extern|auto|struct|union|enum|typedef)\b', original\_name)):

replacements[original\_name] = generate\_variable(config)

for original, new in replacements.items():

code\_without\_strings = re.sub(rf'\b{original}\b', new, code\_without\_strings)

def restore\_strings(match):

index = int(match.group(1))

return string\_literals[index]

final\_code = re.sub(r'\_\_STRING\_LITERAL\_(\d+)\_\_', restore\_strings, code\_without\_strings)

return final\_code

def main():

config = load\_config("obfuscator.cfg")

file = open(config["input\_file"], "r")

code = file.read()

if config["remove\_comments"]: code = remove\_comments(code)

if config["replace\_variables"]: code = replace\_variables(code, config)

if config["shuffle\_functions"]: code = shuffle\_functions(code)

if config["add\_garbage"]: code = add\_garbage(code, config)

if config["remove\_spaces"]: code = remove\_spaces(code)

new\_file = open(config["output\_file"], "w")

new\_file.write(code)

main()

ПРИЛОЖЕНИЕ В

Листинг программы «Программа с вводом пароля после работы программы-обфускатора»

#include <stdio.h>

#include <string.h>

#define x\_uaVIUiHAvz 50

void x\_PxWFxwxWjX(){int x\_EDfBMArKAt=1.53;double x\_lgjLskPKus=1.04;int x\_fyHMUthWQy=1.54;float x\_eWfnAmRWBL=45;double x\_ZVSHgaQlmR='i';for(int x\_QeBFHdNZcz=0;x\_QeBFHdNZcz < 5;x\_QeBFHdNZcz++){}int x\_LRQnBDEPxl=0;do{x\_LRQnBDEPxl++;}while(x\_LRQnBDEPxl < 5);int x\_YtlfRNsXfj=0;while(x\_YtlfRNsXfj < 5){x\_YtlfRNsXfj++;}int x\_BvImlCUqMQ=0;while(x\_BvImlCUqMQ < 5){x\_BvImlCUqMQ++;}for(int x\_YhJUXWCgUT=0;x\_YhJUXWCgUT < 5;x\_YhJUXWCgUT++){}}double x\_fNETqkVNWA(){double x\_gLYxgQdZNI=1.96;int x\_QUcVFnOySW=1.03;double x\_WWHbxDmMHD=1.27;double x\_mRZBMpmySi=1.66;int x\_WbRiVnuFkT=1.0;int x\_CmBvIxgcOM=0;while(x\_CmBvIxgcOM < 5){x\_CmBvIxgcOM++;}for(int x\_zvXcWulJjs=0;x\_zvXcWulJjs < 5;x\_zvXcWulJjs++){}int x\_sAEDIDtYnn=0;while(x\_sAEDIDtYnn < 5){x\_sAEDIDtYnn++;}int x\_xyWGcnuCNG=0;while(x\_xyWGcnuCNG < 5){x\_xyWGcnuCNG++;}for(int x\_ewzgtSVLcJ=0;x\_ewzgtSVLcJ < 5;x\_ewzgtSVLcJ++){}return 47;}double x\_KaiOjvraAw(){int x\_wZWUUGVHeA='o';int x\_dLFIOTVNAh=0;int x\_FqhamAlrbP=1.63;double x\_iKsKnqYWxn=1.45;double x\_sdepAtJbmN='t';int x\_pJTGqzzNgx=0;while(x\_pJTGqzzNgx < 5){x\_pJTGqzzNgx++;}int x\_UmSVVGwgsO=0;while(x\_UmSVVGwgsO < 5){x\_UmSVVGwgsO++;}int x\_KRNObqdIMR=0;do{x\_KRNObqdIMR++;}while(x\_KRNObqdIMR < 5);int x\_jVjKopoPah=0;while(x\_jVjKopoPah < 5){x\_jVjKopoPah++;}for(int x\_GyVigStqWq=0;x\_GyVigStqWq < 5;x\_GyVigStqWq++){}return 93;}int x\_AUKjrucGWh(){int x\_pwgTqBRuCn='v';float x\_CAxcsbUgMy=72;double x\_GFFHWcRknn='b';char x\_iGSibEeXEp='h';int x\_qZBpnVqoss='q';for(int x\_efiTRIkyfb=0;x\_efiTRIkyfb < 5;x\_efiTRIkyfb++){}for(int x\_FVfvSwRMFI=0;x\_FVfvSwRMFI < 5;x\_FVfvSwRMFI++){}int x\_GspjuTaMXv=0;while(x\_GspjuTaMXv < 5){x\_GspjuTaMXv++;}for(int x\_cpaGKfWJnP=0;x\_cpaGKfWJnP < 5;x\_cpaGKfWJnP++){}int x\_KaXlhaxvpl=0;while(x\_KaXlhaxvpl < 5){x\_KaXlhaxvpl++;}return 5;}void x\_UycdOEmuyf(){float x\_eiJyyfNWbu='g';char x\_JUImYNDnEr='o';int x\_VNHqDwDNXy=1.77;float x\_hXdFtFuqHg='p';double x\_crevswQZXl=1.61;for(int x\_zcHuWAVOHr=0;x\_zcHuWAVOHr < 5;x\_zcHuWAVOHr++){}int x\_UqJbvXLLzO=0;do{x\_UqJbvXLLzO++;}while(x\_UqJbvXLLzO < 5);for(int x\_QCBXOARBsj=0;x\_QCBXOARBsj < 5;x\_QCBXOARBsj++){}for(int x\_qvKblHppTq=0;x\_qvKblHppTq < 5;x\_qvKblHppTq++){}for(int x\_SHojBnTYdG=0;x\_SHojBnTYdG < 5;x\_SHojBnTYdG++){}} int x\_izCQRCzrBp(const char \* x\_SPdJaNnwMe,char \* x\_ssKBsyeeuK){for(int x\_paebHujNWH=0;x\_paebHujNWH < strlen(x\_ssKBsyeeuK);x\_paebHujNWH++){x\_ssKBsyeeuK[x\_paebHujNWH]^=x\_uaVIUiHAvz;}int x\_RUZZXnCPRK=strncmp(x\_SPdJaNnwMe,x\_ssKBsyeeuK,strlen(x\_SPdJaNnwMe));return x\_RUZZXnCPRK==0&&strlen(x\_SPdJaNnwMe)==strlen(x\_ssKBsyeeuK);}int main(int argc,char const \*argv[]){int x\_FsFDVmGKTl=0;char x\_SPdJaNnwMe[50];char x\_ssKBsyeeuK[50];FILE \*x\_KYjfVdNATb;x\_KYjfVdNATb=fopen("passxor.txt","r");if(x\_KYjfVdNATb==NULL){printf("The file is not opened.The program will now exit.");return 0;}while(fgets(x\_ssKBsyeeuK,50,x\_KYjfVdNATb)!=NULL);fclose(x\_KYjfVdNATb);printf("Enter password:\n");scanf("%s",x\_SPdJaNnwMe);while(1){x\_FsFDVmGKTl=x\_izCQRCzrBp(x\_SPdJaNnwMe,x\_ssKBsyeeuK);if(x\_FsFDVmGKTl){char x\_nRWJhHzcGC[20];printf("\nYou are log in\nType something:\n");scanf("%s",x\_nRWJhHzcGC);printf("%s\n",x\_nRWJhHzcGC);}else{printf("\nYou are not log in\nTry again:\n");scanf("%s",x\_SPdJaNnwMe);}}return 0;}