1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт кибербезопасности и защиты информации

**ЛАБОРАТОРНАЯ РАБОТА № 3**

1. «**Защита программного обеспечения от нелегального использования**»
2. по дисциплине «Основы информационной безопасности»
3. Выполнил
4. студент гр. №4851003/00002 Скрипко И.А.

<*подпись*>

1. Преподаватель
2. асс. преподавателя Панков И.Д.

<*подпись*>

1. Санкт-Петербург
2. 2021
3. **Цель работы**

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

1. **Постановка задачи**
2. Создать простую программу на языке С, запрашивающую пароль, пароль должен храниться в отдельном файле в открытом виде. С помощью утилиты HIEW отыскать в исполняемом файле команду перехода при проверке правильности пароля и изменить ее таким образом, чтобы при вводе любого пароля программа принимала пользователя как авторизованного. Затем, включив опцию оптимизации компилятора, скомпилировать программу и повторить действия с HIEW редактором. Далее модифицировать программу таким образом, чтобы исключить открытое хранение пароля, и повторить действия с HIEW редактором.
3. Запустить полученную от преподавателя программу Nag2.exe и зафиксировать наблюдаемые в работе программы эффекты. Затем, с помощью программы дизассемблера, модифицировать код таким образом, чтобы надпись «ОК» появлялась сразу же после активации. После этого, с помощью программы дизассемблера, модифицировать код таким образом, чтобы работа программы больше не блокировалась.
4. Запустить полученную от преподавателя программу Guard2.exe, выявить с помощью какой утилиты запакован исполняемый файл, с помощью отладчика найти команды, отвечающие за проверку пароля и изменить код так, чтобы выполнялся обход проверки.
5. Для разработанной в первом пункте программы написать программу-обфускатор.
6. **Теоретические исследования**
   1. **Методы, применяемые для защиты коммерческих программ от ввода некорректных учетных записей**

* Запрашивание пароля или ключа активации у пользователя.
* Вывод nag screen («изводящее» окно) пользователю, который не зарегистрирован или который не купил программу.
* Контроль целостности исполняемого кода. Например, подсчет суммы байт программы для проверки на то, были ли они изменены.
* Шифрование кода.
* Обфускация.
  1. **Признаки, по которым можно найти интересующий код сопоставления с паролем в программе**

Проверка пароля происходит с помощью условного перехода, часто при вводе некорректного ввода программа оповещает пользователя об этом, поэтому можно найти место с условным переходом и с уведомлением пользователя о неверно введенном пароле.

* 1. **Основные методы защиты от дизассемблирования и отладки программ**

Одним из основных методов является дублирование данных, в таком случае в последствии будет возможность сравнить программу с исходной и узнать, были ли внесены в нее изменения. Также это можно сделать с помощью контрольной суммы (значение, рассчитанное по набору данных путем применения определенного алгоритма). Данный метод можно обойти с помощью удалением из программы мест проверки.

Другим основным способом является шифрование кода. Суть метода заключается в том, чтобы видоизменить программу таким образом, чтобы исполняемый код не соответствовал тексту программы в открытом виде. Но в таком случае программа во время исполнения должна сама себя дешифровать. Такую защиту можно обойти, найдя функционал дешифровки и получить код в открытом виде.

Еще одним важным методом является обфускация – запутывание исполняемого кода. Суть метода состоит в том, чтобы привести код программы к виду, сохраняющему ее функциональность, но затрудняющему анализ, понимание и модификацию алгоритма. Данный метод можно обойти с помощью замен названий функций, переменных, строковых констант, удалением «мусорного» кода.

* 1. **Реализация обфускации с помощью виртуальных машин**

Для этого создается случайная виртуальная машина со случайным набором инструкций и весь код трансформируется под нее.

* 1. **Методы усиления механизмов защиты программ от нелегального использования**

Написание проверок паролей, алгоритмов для шифрования кода или выводимой на экран пользователя информации в отдельных функциях для усложнения анализа работы программы – будет использоваться много ссылок и отыскать все нужные части программы будет сложнее, также выводимый пользователю текст не будет храниться в открытом виде и понять, что в данный момент происходит, будет сложнее.

1. **Описание решения**
   1. **Программа проверки пароля**

Была написала простая программа на языке С для проверки пароля. Если пользователь вводит верный пароль, то программа оповещает его об этом и завершает работу, в противном случае пользователю сообщается, что пароль был введен неправильно и предлагается ввести его еще раз. Затем, с помощью утилиты HIEW был найден условный переход – были найдены уведомления пользователя о правильности/неправильности введенного им пароля. На рисунке 1 выделен момент условного перехода. Значение jz делает переход по адресу, если содержимое равно нулю. Данное значение было изменено на значение jnz (рисунок 2) – переход по адресу, если содержимое не равно нулю. Таким образом, программа стала принимать все неверные пароли.

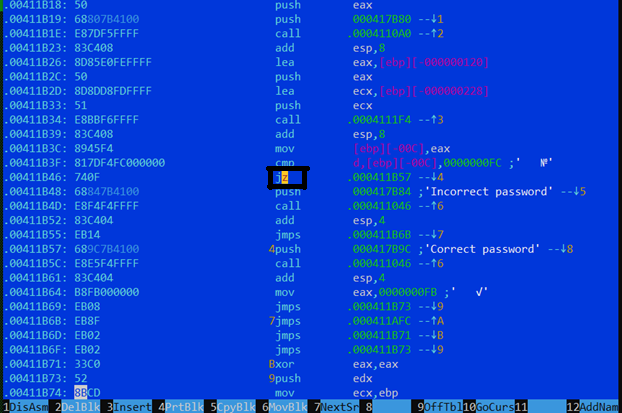


Рисунок 1 – Момент условного перехода

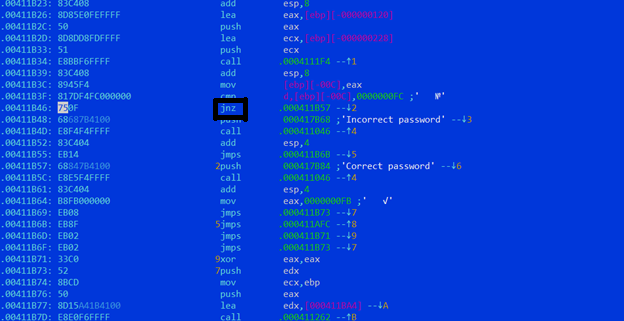


Рисунок 2 – Измененный момент условного перехода

Далее была включения оптимизация компилятора (рисунок 3). После данного действия программа снова начала работать корректно и не принимала неверные пароли. Но повторив действия с HIEW-редактором, снова начала принимать все неверные пароли.

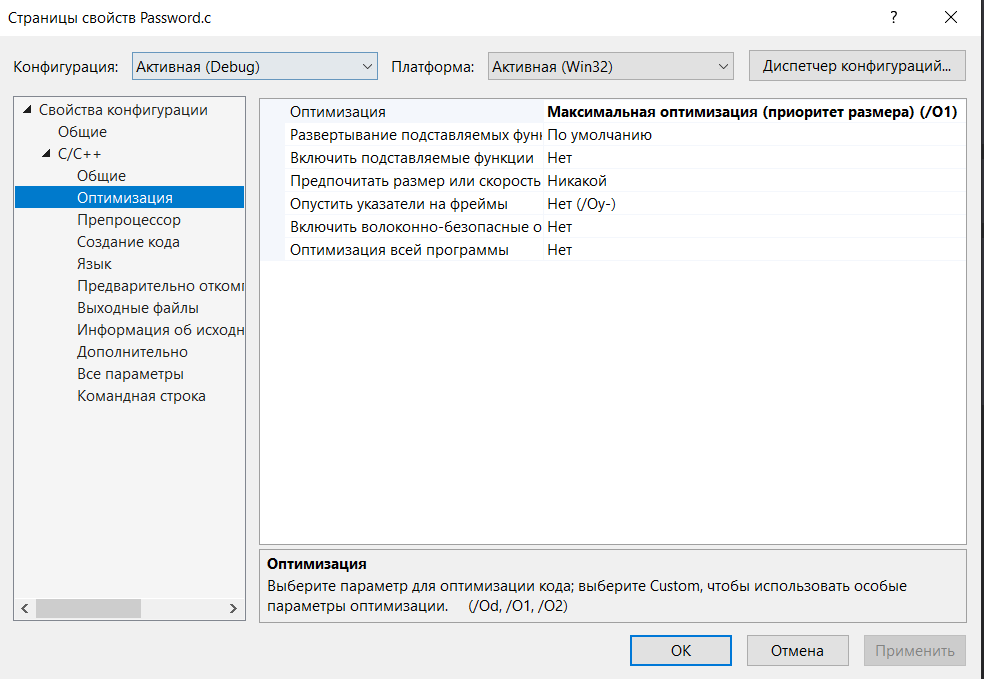


Рисунок 3 – Включение оптимизации компилятора

Затем программа была модернизирована с помощью простой шифровки *пароль* **XOR** *константа.* Далее повторились действия с HIEW-редактором и программа принимала все неверные пароли.

* 1. **Программа Nag2.exe**

С помощью программы IDA Pro была найдена функция, которая отвечает за счетчик, выводимого на кнопке всплывающего окна (рисунок 4). Затем был просмотрен псевдокод данной функции (рисунок 5). Счетчик в программе шел от 100 до 0, в псевдокоде видно, что переменная «v2» увеличивается до 100, а на кнопку выводится значения «100-v2». После того как «v2» станет больше 100, с помощью функции ShowWindow устанавливается новое состояние показываемого окна. Таким образом, был сделан вывод, что изменив условие 10-ой строчки псевдокода можно будет сразу нажимать на кнопку «ОК», не дожидаясь, когда счетчик станет равным нулю. Байт был найден и изменен в программе HIEW (рисунок 6).

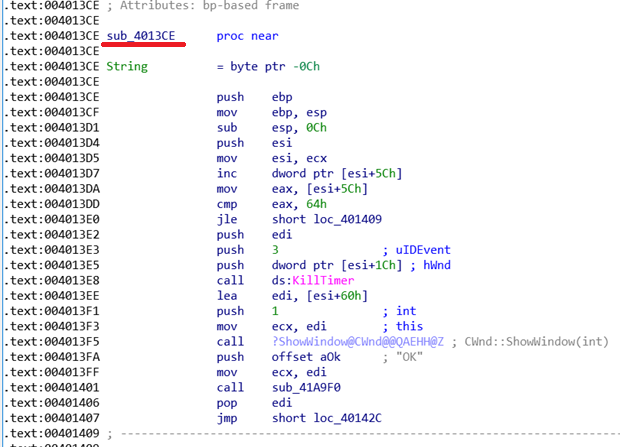


Рисунок 4 – Функция, отвечающая за счетчик всплывающего окна

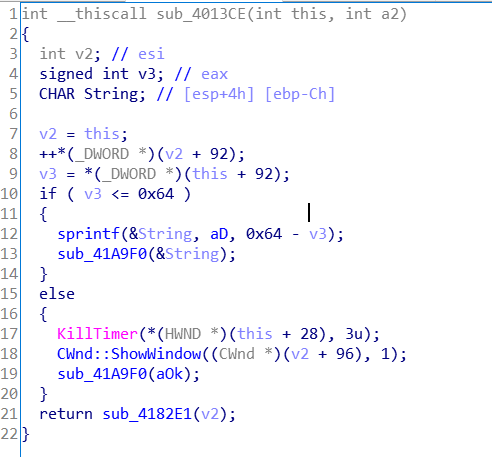


Рисунок 5 – Псевдокод функции всплывающего окна

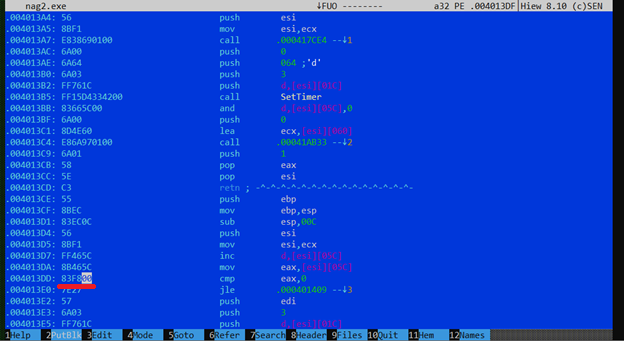


Рисунок 6 – Изменение байта в HIEW-редакторе

Далее была найдена функция активации таймера SetTimer (рисунок 7), просмотрен псевдокод (рисунок8), найден и изменен (рисунок 9) подаваемый в нее параметр IpTimerFunc (указатель на функцию, которая будет уведомлена по истечении времени ожидания) таким образом, чтобы по истечении времени данная функция не узнавала, что время истекло. Соответственно, не будет вызываться окно и прерывать работу программы.

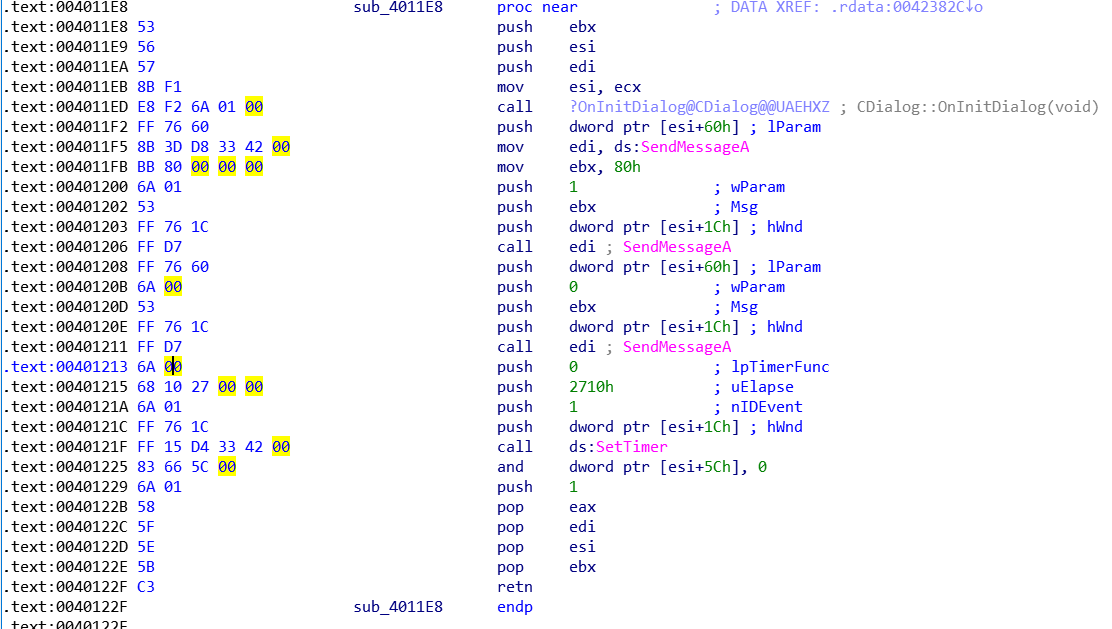


Рисунок 7 – Функция активации таймера SetTimer

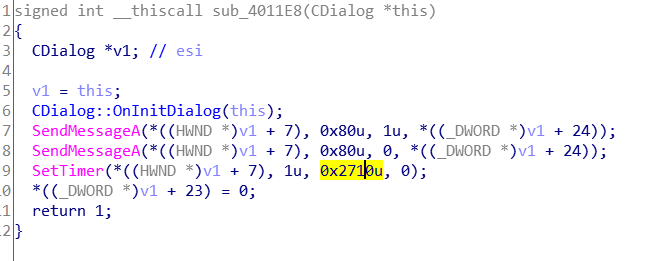


Рисунок 8 – Псевдокод функции с таймером

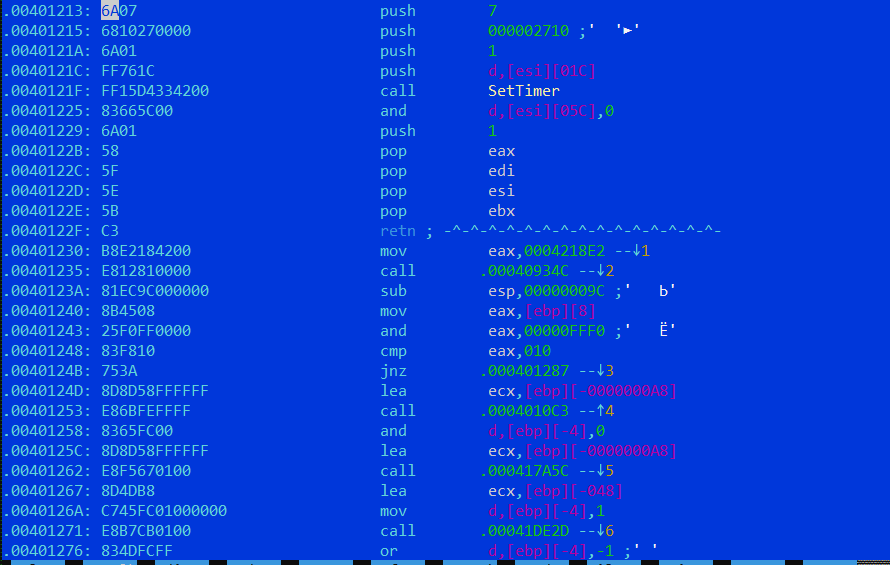


Рисунок 9 – Изменение параметра функции SetTimer

* 1. **Программа Guard2.exe**

Была запущена программа Guard2.exe. Результат работы программы представлен на рисунке 10. Далее, с помощью программы PEiD была выяснена утилита, с помощью которой был упакован исполняемый файл (рисунок 11). Затем программа была распакована с помощью утилиты UPX (рисунок 12). Далее с помощью программы IDA Pro был найден исполняемый код (рисунки 13-15).

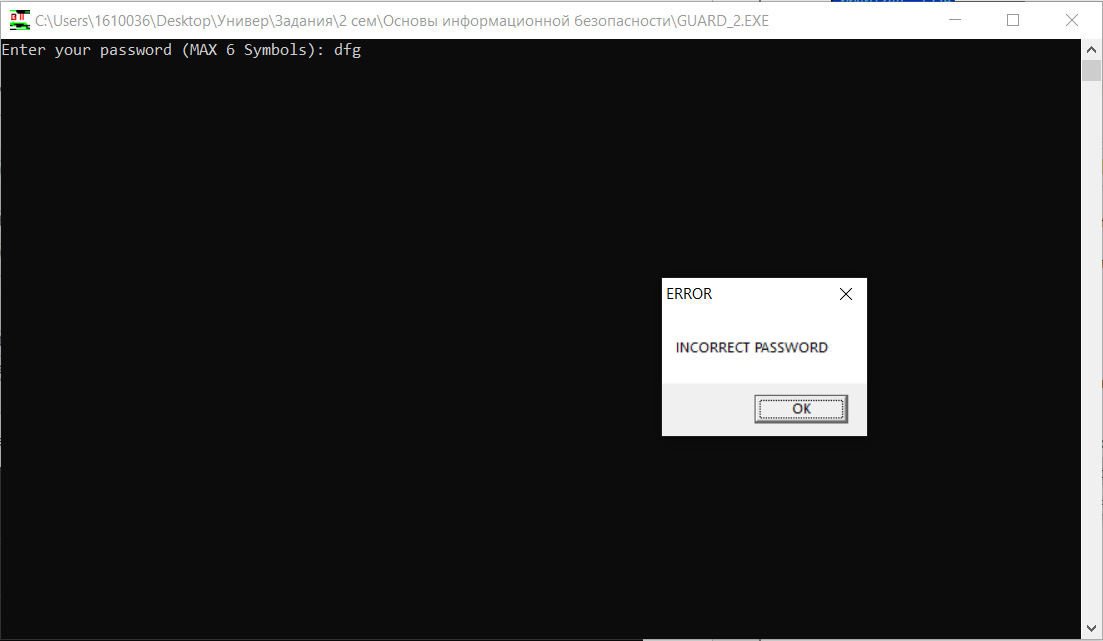


Рисунок 10 – Работа программы Guard2.exe

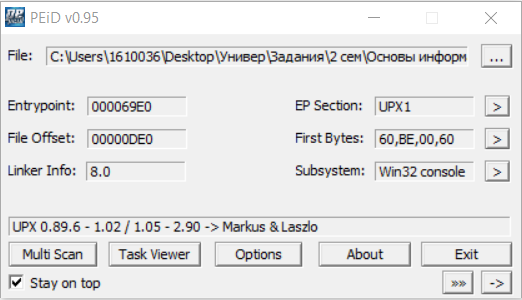


Рисунок 11 – Определение упаковщика

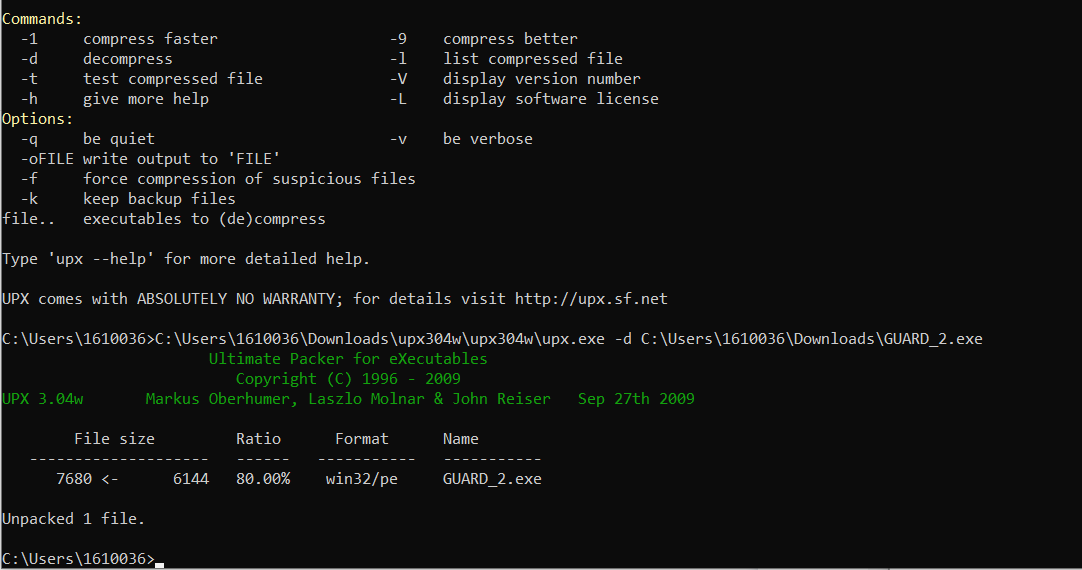


Рисунок 12 – распаковка файла

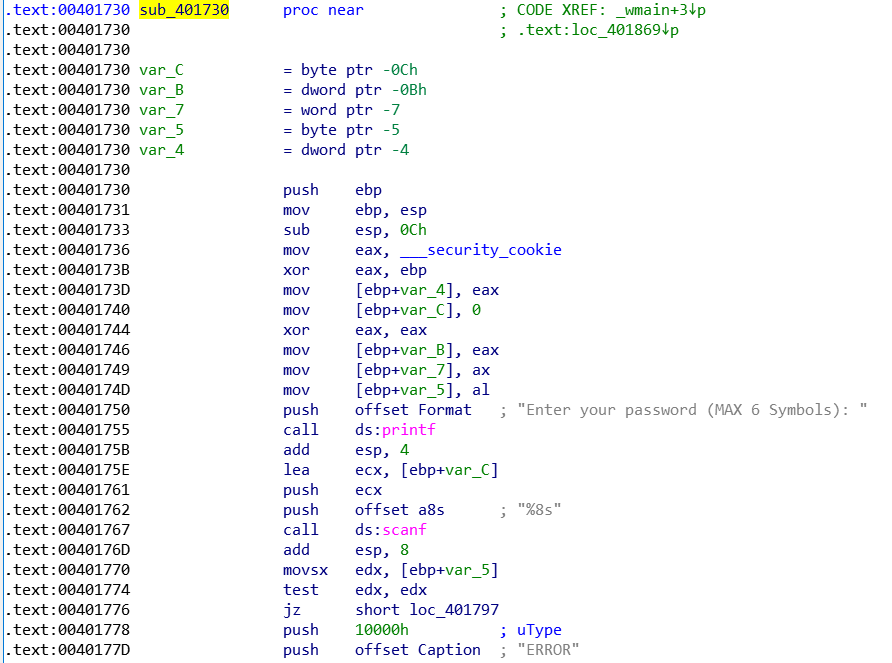


Рисунок 13 – Момент проверки пароля

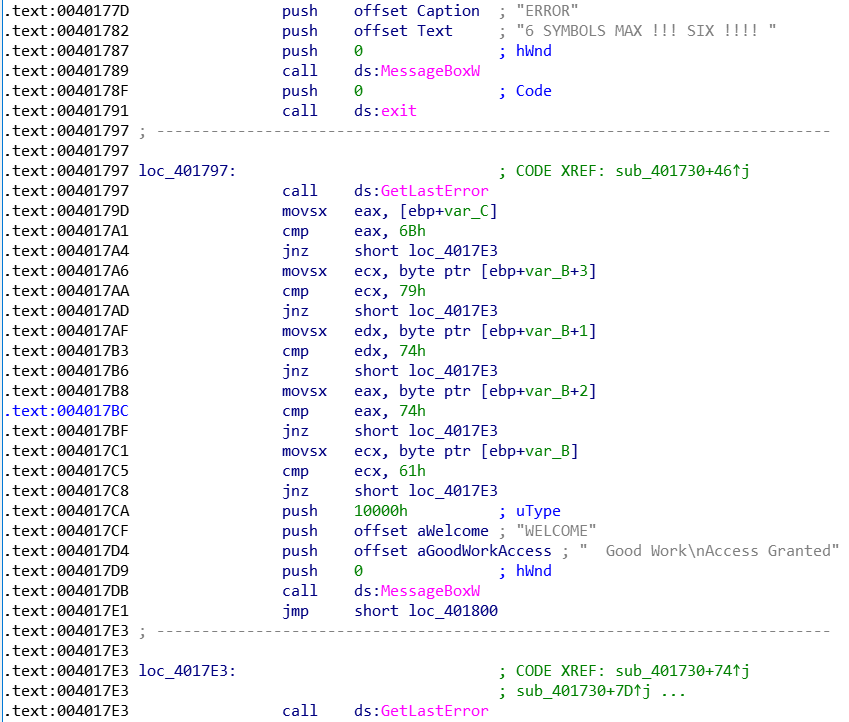


Рисунок 14 – Момент проверки пароля

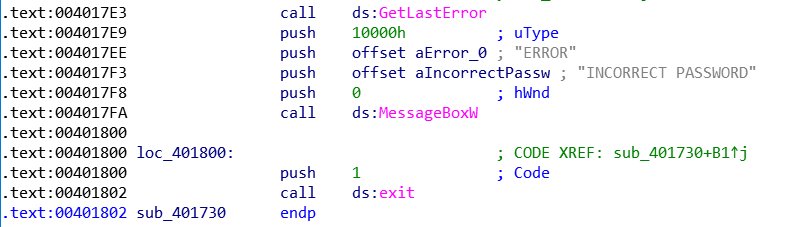


Рисунок 15 – Момент проверки пароля

Был найден момент перехода после ввода пароля (рисунок 16). Затем, с помощью утилиты HIEW был изменен исполняемый код таким образом, чтобы он принимал все неверно-введенные пароли (рисунок 17).



Рисунок 16 – Момент перехода после ввода пароля

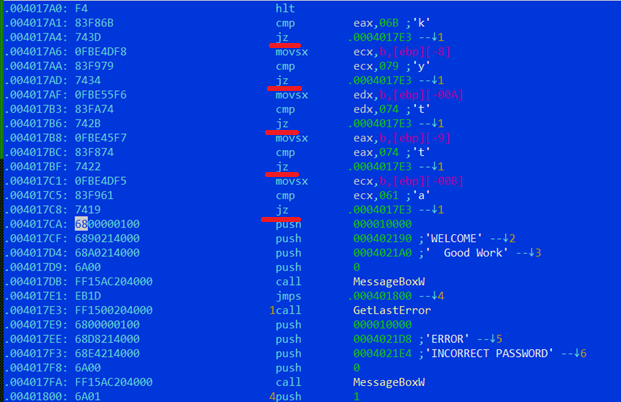


Рисунок 17 – Измененный переход после ввода пароля

* 1. **Обфускатор**

Программа принимает на вход имя файла с кодом, считывает код и сохраняет в массив символов. Затем вызывает 5 функций: changed\_name (функция для изменения имен переменных), delete\_comments (функция для удаления комментариев их кода), delete\_new\_line (функция для удаления переходов на новую строку), delete\_tabs (функция для удаления табуляции), change\_text (функция для изменения текста, выводимого программой). Changed\_name не меняет имени функции main. Delete\_new\_line начинает работу после всех заголовочных файлов. Changed\_text не меняет в тексте спецификаторы и не меняет подключение заголовочного файла.

После вызова пяти функций создается файл, куда будет вставлен код, прошедший обфускацию. Сначала в файле объявляются мусорные переменный как глобальные переменный, затем считываются заголовочные файлы, после чего переписывается модифицированный код, при этом после знака «{», если это начало действий блока, вписываются различные действия над мусорными переменными до тех пор, пока не закончится код или не закончится «мусор». Результат работы обфускатора представлен в пункте 5.

1. **Тестирование и результаты работы программы**

Результаты работы программы приведены на рисунках 18-27.

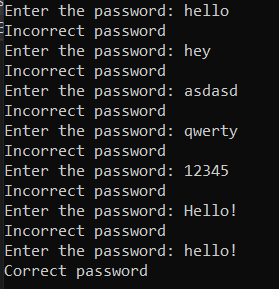


Рисунок 18 – Проверка пароля до редактирования в HIEW



Рисунок 19 – Проверка пароля после редактирования в HIEW

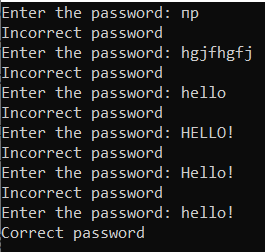


Рисунок 20 – Проверка пароля после включения оптимизатора

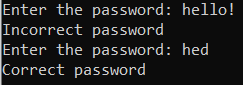


Рисунок 21 – Проверка пароля после включения оптимизатора

и редактирования в HIEW

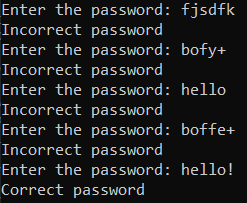


Рисунок 22 – Проверка пароля без открытого хранения пароля



Рисунок 23 – Проверка пароля без открытого хранения

пароля после изменения в HIEW

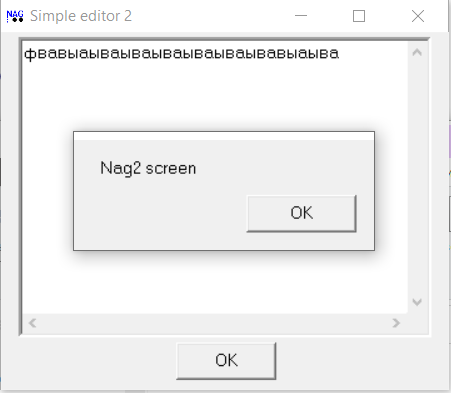


Рисунок 24 – Работа Nag2.exe после редактирования счетчика

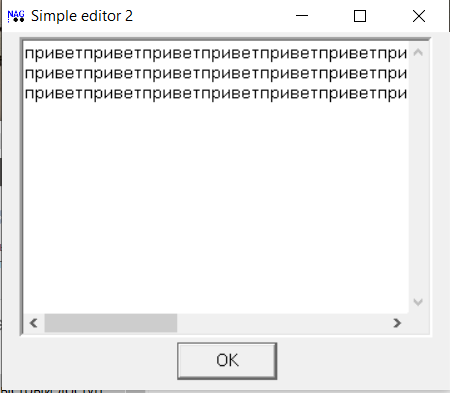


Рисунок 25 – Работа программы Nag2.exe после редактирования функции активации таймера

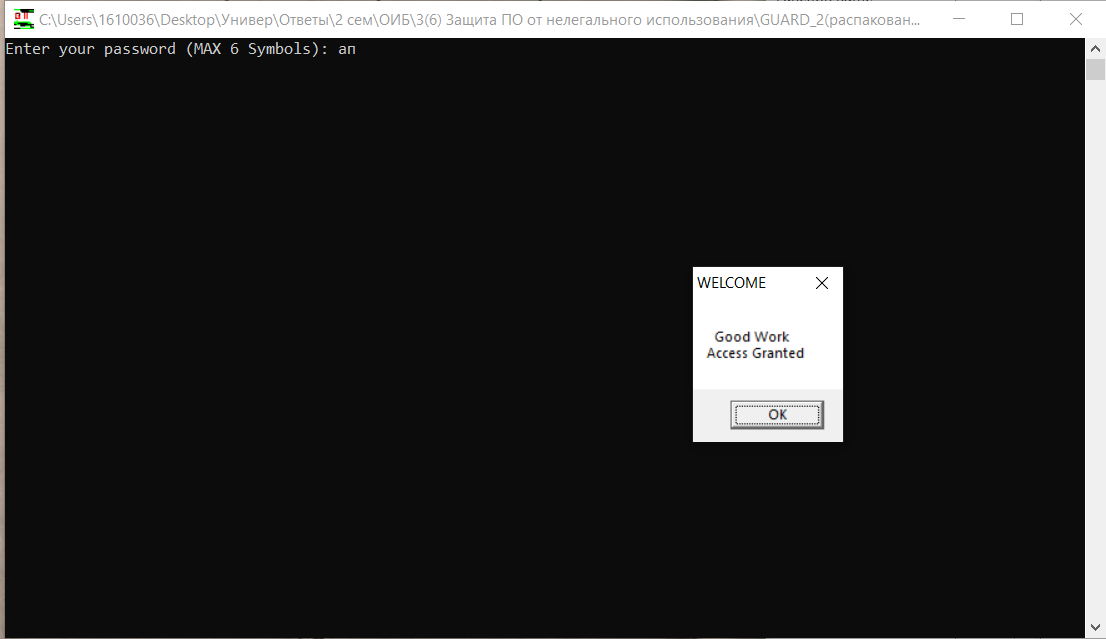


Рисунок 26 – Результат работы программы GUARD\_2.exe после исправления кода

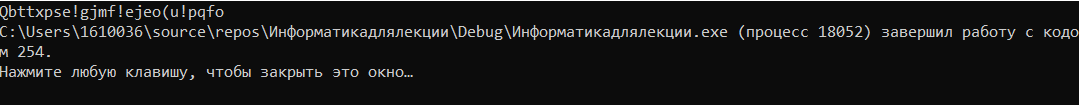


Рисунок 27 – результат работы программы, прошедшей обфускацию

Код программы до обфускации представлен в приложении А.

Код программы после обфускации:

int csalkmcksl=1;char\* sdhfsdfjk;int hdgbh=15;char ajdfkdshbhf='d';

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Password.h"

#include <stdio.h>

#include <string.h>

int difdl(char\* gpvgtarcuuyqtf, char\* tgcnarcuuyqtf) {hdgbh+=7; if (strcmp(gpvgtarcuuyqtf, tgcnarcuuyqtf) == 0) return CORRECT\_PASSWORD; else return ERROR\_PASSWORD; } int main() {if (ajdfkdshbhf) ajdfkdshbhf=0; int sftvmu = 0; FILE\* qbttxpse=NULL; char tgcnarcuuyqtf[MAX\_SIZE\_OF\_PASSWORD]={'\0'}; char gpvgtarcuuyqtf[MAX\_SIZE\_OF\_PASSWORD] = { '\0' }; qbttxpse = fopen("rcuuyqtf/uyu", "r"); if (qbttxpse == NULL) {csalkmcksl++; printf("Qbttxpse!gjmf!ejeo(u!pqfo"); return ERROR\_OPEN\_FILE; } fgets(tgcnarcuuyqtf, MAX\_SIZE\_OF\_PASSWORD, qbttxpse); tgcnarcuuyqtf[strlen(tgcnarcuuyqtf)] = '\0'; fclose(qbttxpse); while (sftvmu != CORRECT\_PASSWORD) {sdhfsdfjk=NULL; printf("Foufs!uif!rcuuyqtf;!"); scanf("%s", gpvgtarcuuyqtf); sftvmu = difdl(gpvgtarcuuyqtf, tgcnarcuuyqtf); if (sftvmu != CORRECT\_PASSWORD) printf("Jodpssfdu!rcuuyqtf]o"); else { printf("Dpssfdu!rcuuyqtf]o"); return SUCCESS; } } }

1. **Выводы**

В ходе проделанной работы были приобретены навыки по защите приложений от нелегального использования, по анализу исполняемых кодов в отсутствии исходных текстов и по применению способов защиты программ от дизассемблирования и отладки. Были получены некоторые навыки работы с программами HIEW, IDA Pro, UPX и PEiD. Была написана простая программа на проверку пароля, после чего был написан обфускатор для программы на проверку пароля. Были изучены такие методы защиты, как запрос пароля, вывод «назойливого окна» и обфускация.

**ПРИЛОЖЕНИЕ А**

Листинг программы «Password.c»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Password.h"

#include <stdio.h>

#include <string.h>

int check(char\* enter\_password, char\* real\_password)

{

if (strcmp(enter\_password, real\_password) == 0)

return CORRECT\_PASSWORD;

else

return ERROR\_PASSWORD;

}

int main()

{

//открываем файл и берем пароль

int result = 0;

FILE\* password=NULL;

char real\_password[MAX\_SIZE\_OF\_PASSWORD]={'\0'};

char enter\_password[MAX\_SIZE\_OF\_PASSWORD] = { '\0' };

password = fopen("password.txt", "r");

if (password == NULL)

{

printf("Password file didn't open");

return ERROR\_OPEN\_FILE;

}

fgets(real\_password, MAX\_SIZE\_OF\_PASSWORD, password);

real\_password[strlen(real\_password)] = '\0';

fclose(password);

//просим пользователя ввести пароль и проверяем введенный пароль на правильность

while (result != CORRECT\_PASSWORD)

{

printf("Enter the password: ");

scanf("%s", enter\_password);

result = check(enter\_password, real\_password);

if (result != CORRECT\_PASSWORD)

printf("Incorrect password\n");

else

{

printf("Correct password\n");

return SUCCESS;

}

}

}

**ПРИЛОЖЕНИЕ B**

Листинг программы «Password.h»

#define MAX\_SIZE\_OF\_PASSWORD 255

#define ERROR\_OPEN\_FILE 254

#define ERROR\_PASSWORD 253

#define CORRECT\_PASSWORD 252

#define SUCCESS 251

**ПРИЛОЖЕНИЕ С**

Листинг программы «Without\_open\_Password.c»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include "Password.h"

#include <stdio.h>

#include <string.h>

int check(char\* enter\_password, char\* real\_password)

{

if (strcmp(enter\_password, real\_password) == 0)

return CORRECT\_PASSWORD;

else

return ERROR\_PASSWORD;

}

int coder(char\* enter\_password)

{

for (int i = 0;i < strlen(enter\_password);i++)

enter\_password[i] ^= CONST;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//открываем файл и берем пароль

int result = 0;

FILE\* password=NULL;

char real\_password[MAX\_SIZE\_OF\_PASSWORD]={'\0'};

char enter\_password[MAX\_SIZE\_OF\_PASSWORD] = { '\0' };

password = fopen("C:\\Users\\1610036\\source\\repos\\Защита программного обеспечения от нелегального использования\\Защита программного обеспечения от нелегального использования\\password.txt", "r");

if (password == NULL)

{

printf("Password file didn't open");

return ERROR\_OPEN\_FILE;

}

fgets(real\_password, MAX\_SIZE\_OF\_PASSWORD, password);

real\_password[strlen(real\_password)] = '\0';

fclose(password);

//просим пользователя ввести пароль и проверяем введенный пароль на правильность

while (result != CORRECT\_PASSWORD)

{

printf("Enter the password: ");

scanf("%s", enter\_password);

coder(enter\_password);

result = check(enter\_password, real\_password);

if (result != CORRECT\_PASSWORD)

printf("Incorrect password\n");

else

{

printf("Correct password\n");

return SUCCESS;

}

}

}

**ПРИЛОЖЕНИЕ D**

Листинг программы «Without\_open\_Password.h»

#define MAX\_SIZE\_OF\_PASSWORD 255

#define ERROR\_OPEN\_FILE 254

#define ERROR\_PASSWORD 253

#define CORRECT\_PASSWORD 252

#define SUCCESS 251

#define CONST 10

**ПРИЛОЖЕНИЕ E**

Листинг программы «Obfuscator.c»

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <Windows.h>

#include "Obfuscator.h"

#include <stdio.h>

char\* add\_garbage()

{

char garbage\_file[SIZE\_OF\_NAME] = "Мусор.txt";

FILE\* garbage = NULL;

//printf("Введите имя файла с мусором: ");

//scanf("%s", garbage\_file);

garbage=fopen(garbage\_file, "r");

if (garbage == NULL)

{

printf("Open error\n");

return NULL;

}

fseek(garbage, 0, SEEK\_END);//устанавливаем позицию в конце файла

int lsize = ftell(garbage);//получаем размер в байтах

char\* garbage\_code = (char\*)malloc((lsize + 1) \* sizeof(char));//выделяем память

if (garbage\_code == NULL)

{

fclose(garbage);

printf("Malloc error\n");

return NULL;

}

rewind(garbage);//ставим позицию в начало файла

int letters = fread(garbage\_code, sizeof(char), lsize, garbage);//считываем из файла в массив символов

garbage\_code[letters] = '\0';

fclose(garbage);

free(garbage\_code);

return garbage\_code;//возвращаем указатель на мусорный код

}

void delete\_new\_line(char\* code, int size)//удаление переноса строки

{

for (int i = 100; i < size; i++)

if (code[i] == '\n') code[i] = ' ';//ищем перенос на новую строку и меняем на пробел

}

void change\_text(char\* code, int size)//изменение текста, выводимого программой

{

int i = 0, j = 0;

for (i = 60; i < size;)//чтобы не изменять имя заголовочного файла при подключении

if (code[i] == '"')//ищем начало строки и меняем строку

{

if (code[i + 2] == '"' || code[i + 3] == '"')

{

if (code[i + 2] == '"') i += 3;

else i += 4;

continue;//спецификаторы для открытия файла

}

j = i + 1;

while (code[j] != '"')

{

if (code[j] == '%' || code[j - 1] == '%')//спецификаторы не меняем

{

j++;

continue;

}

code[j++]+=CHANGE\_TEXT;

if (j > size) return;

}

i = ++j;//если меняли строку, то устанавливаем i после строки

}

else i++;//если не строка, то проверяем следующий элемент

}

void delete\_tabs(char\* code, int size)//удаление табуляции

{

for (int i = 0; i < size; i++)

if (code[i] == '\t') code[i] = ' ';

}

void delete\_comments(char\* code, int size)//удаление комментариев

{

int i = 0, j = 0, k;//индексы

for (i = 0; i < size - 1; i++)//проходим весь код

{

//проверяем на первый формат комментариев

if (code[i] == '/' && code[i + 1] == '/')//ищем обозначение комментария

{

j = i;

while (code[j] != '\n') j++;//ищем конец комментария

int k = i;

while (j < size) code[k++] = code[j++];//сдвигаем текст на комментарий

while (k < size) code[k++] = '\0';//меняем значение перенесенной части кода

}

//проверяем на второй формат комментариев

if (code[i] == '/' && code[i + 1] == '\*')//ищем начало комментария

{

int j = i;

while (!(code[j] == '/' && code[j - 1] == '\*')) j++;//ищем конец комментария

j++;//увеличиваем индекс еще на одни, чтобы указывал на код, который после комментария

int k = i;

while (j < size) code[k++] = code[j++];//сдвигаем на комментария

while (k < size) code[k] = '\0';//меняем значение перенесенной части

}

}

}

void change\_name(char\* code, int size)

{

char name\_of\_something[SIZE\_OF\_NAME] = { '\0' };

int i = 0;

for (i = 0; i < size;)

{

int j = 0, k = 0, z = 0, n = 0, flag = 1;

memset(name\_of\_something, 0, sizeof(char) \* SIZE\_OF\_NAME);//обнуляем массив, где будет храниться имя переменной

if (code[i] == 'i' && code[i + 1] == 'n' && code[i + 2] == 't' && code[i + 3] == ' '&& !(code[i+4]=='m'&&code[i + 5] == 'a'&&code[i + 6] == 'i'&&code[i + 7] == 'n'))//если встречено имя типа int и это не функция main

{

j = i + 4;//поставили индекс на начало имени

//если функция, то будет открыта скобка после имени, если переменная, то она либо просто будет объявлена, либо будет объявлена с начальным значением, если массив, то будет с квадратными скобками, если указатель, то звездочка, если перечисление, то запятая

while (code[j] != '(' && code[j] != '=' && code[j] != ';' && code[j] != '[' && code[j] != '\*' && code[j] != ' ' && code[j] != ',' && code[j] != ')') j++;//нашли конец имени

for (k = i + 4;k < j;k++)

{

name\_of\_something[z++] = code[k];//сохраняем имя

code[k] += CHANGE\_NAME;//меняем имя

}

i = j + 1;//ставим индекс после имени

for (n=i;n < size;n++)//проходим весь код и проверяем встречается ли это имя еще

{

flag = 1;//показываем, что на данный момент все совпадает (проверено 0 символов)

for (j = 0, k = n;j < z;j++, k++)//проверяем встречается ли в тексте это имя еще

if (code[k] != name\_of\_something[j])//если имя не совпадает, то указываем на это

{

flag = 0;

break;

}

if (flag) for (k = n;k < n + z;k++)//если имя совпало, то меняем его также, как и до этого

code[k] += CHANGE\_NAME;

}

continue;//пропускаем оставшуюся часть кода и ищем следующее имя

}

else if (code[i] == 'c' && code[i + 1] == 'h' && code[i + 2] == 'a' && code[i + 3] == 'r' && code[i + 4] == '\*')//если встречено имя типа char

{

j = i + 6;//поставили индекс на начало имени

//если функция, то будет открыта скобка после имени, если переменная, то она либо просто будет объявлена, либо будет объявлена с начальным значением, если массив, то будет с квадратными скобками, если перечисление, то запятая

while (code[j] != '(' && code[j] != '=' && code[j] != ';' && code[j] != '[' && code[j] != '\*' && code[j] != ' ' && code[j] != ',' && code[j] != ')') j++;//нашли конец имени

for (k = i + 6;k < j;k++)

{

name\_of\_something[z++] = code[k];//сохраняем имя

code[k] += CHANGE\_NAME;//меняем имя

}

i = j + 1;//ставим индекс после имени

for (;n < size;n++)//проходим весь код и проверяем встречается ли это имя еще

{

flag = 1;//показываем, что на данный момент все совпадает (проверено 0 символов)

for (j = 0, k = n;j < z;j++, k++)//проверяем встречается ли в тексте это имя еще

if (code[k] != name\_of\_something[j])//если имя не совпадает, то указываем на это

{

flag = 0;

break;

}

if (flag) for (k = n;k < n + z;k++)//если имя совпало, то меняем его также, как и до этого

code[k] += CHANGE\_NAME;

}

continue;//пропускаем оставшуюся часть кода и ищем следующее имя

}

else if (code[i] == 'c' && code[i + 1] == 'h' && code[i + 2] == 'a' && code[i + 3] == 'r' && code[i + 4] == ' ')//если встречено имя типа char

{

j = i + 5;//поставили индекс на начало имени

//если функция, то будет открыта скобка после имени, если переменная, то она либо просто будет объявлена, либо будет объявлена с начальным значением, если массив, то будет с квадратными скобками, если перечисление, то запятая

while (code[j] != '(' && code[j] != '=' && code[j] != ';' && code[j] != '[' && code[j] != '\*' && code[j] != ' ' && code[j] != ',' && code[j] != ')') j++;//нашли конец имени

for (k = i + 5;k < j;k++)

{

name\_of\_something[z++] = code[k];//сохраняем имя

code[k] += CHANGE\_NAME;//меняем имя

}

i = j + 1;//ставим индекс после имени

for (;n < size;n++)//проходим весь код и проверяем встречается ли это имя еще

{

flag = 1;//показываем, что на данный момент все совпадает (проверено 0 символов)

for (j = 0, k = n;j < z;j++, k++)//проверяем встречается ли в тексте это имя еще

if (code[k] != name\_of\_something[j])//если имя не совпадает, то указываем на это

{

flag = 0;

break;

}

if (flag) for (k = n;k < n + z;k++)//если имя совпало, то меняем его также, как и до этого

code[k] += CHANGE\_NAME;

}

continue;//пропускаем оставшуюся часть кода и ищем следующее имя

}

else if (code[i] == 'F' && code[i + 1] == 'I' && code[i + 2] == 'L' && code[i + 3] == 'E' && code[i + 4] == '\*')

{

j = i + 6;//поставили индекс на начало имени

//если функция, то будет открыта скобка после имени, если переменная, то она либо просто будет объявлена, либо будет объявлена с начальным значением, если массив, то будет с квадратными скобками

while (code[j] != '(' && code[j] != '=' && code[j] != ';' && code[j] != '[' && code[j] != '\*' && code[j] != ' ' && code[j] != ')') j++;//нашли конец имени

for (k = i + 6;k < j;k++)

{

name\_of\_something[z++] = code[k];//сохраняем имя

code[k] += CHANGE\_NAME;//меняем имя

}

i = j + 1;//ставим индекс после имени

for (;n < size;n++)//проходим весь код и проверяем встречается ли это имя еще

{

flag = 1;//показываем, что на данный момент все совпадает (проверено 0 символов)

for (j = 0, k = n;j < z;j++, k++)//проверяем встречается ли в тексте это имя еще

if (code[k] != name\_of\_something[j])//если имя не совпадает, то указываем на это

{

flag = 0;

break;

}

if (flag) for (k = n;k < n + z;k++)//если имя совпало, то меняем его также, как и до этого

code[k] += CHANGE\_NAME;

}

continue;//пропускаем оставшуюся часть кода и ищем следующее имя

}

i++;//если никакого имени, то смещаем курсор

}

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int i = 0;

//char name[SIZE\_OF\_NAME];

char name[SIZE\_OF\_NAME]="Password.txt";

FILE\* code = NULL;

FILE\* obf\_code = NULL;

//printf("Введите имя файла с кодом: ");

//scanf("%s", name);

code = fopen(name, "r");

if (code == NULL)

{

printf("Open error\n");

return OPEN\_ERROR;

}

fseek(code, 0, SEEK\_END);//устанавливаем позицию в конце файла

int lsize = ftell(code);//получаем размер в байтах

char\* changed\_code = (char\*)malloc((lsize+1) \* sizeof(char));//выделяем память

if (changed\_code == NULL)

{

fclose(code);

printf("Malloc error\n");

return MALLOC\_ERROR;

}

rewind(code);//ставим позицию в начало файла

int letters=fread(changed\_code, sizeof(char), lsize, code);//считываем из файла в массив символов

changed\_code[letters] = '\0';

fclose(code);

change\_name(changed\_code, letters);

delete\_comments(changed\_code,letters);

delete\_new\_line(changed\_code, letters);

delete\_tabs(changed\_code, letters);

change\_text(changed\_code, letters);

char\* garbage=add\_garbage();//получаем мусор для вставки

if (garbage == NULL)

{

free(changed\_code);

printf("Garbage error\n");

return GARBAGE\_ERROR;

}

int ind = 67;

obf\_code = fopen("Obf\_code.txt", "w");

if (obf\_code == NULL)

{

free(changed\_code);

printf("Open error\n");

return OPEN\_ERROR;

}

for (i = 0;i < 67;i++) fprintf(obf\_code,"%c", garbage[i]);//вставление мусорных переменных как глобальных

fprintf(obf\_code, "%c", '\n');

i = 0;

while (changed\_code[i])

{

fprintf(obf\_code, "%c", changed\_code[i]);

if ((changed\_code[i] == '{'&& changed\_code[i-1]!='=')&&(changed\_code[i] == '{' && changed\_code[i - 2] != '='))//если это не значение чего-либо, то вставляем мусор

{

while (garbage[ind] != ';' && garbage[ind] != '\0')//до тех пор, пока не закончится мусор или пока мусор не будет вставлен во все возможные места

fprintf(obf\_code, "%c", garbage[ind++]);//вставляем мусорную строку

fprintf(obf\_code, "%c", garbage[ind++]);//ставим точку с запятой после мусора и ставим индекс на следующую мусорную строку

}

i++;

}

fclose(obf\_code);

fclose(changed\_code);

free(changed\_code);

return SUCCESS;

}

**ПРИЛОЖЕНИЕ F**

Листинг программы «Obfuscator.h»

#define SIZE\_OF\_NAME 0xff

#define MALLOC\_ERROR 0xfe

#define SUCCESS 0xfd

#define OPEN\_ERROR 0xfd

#define GARBAGE\_ERROR 0xfc

#define CHANGE\_TEXT 0x1

#define CHANGE\_NAME 0x1