1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**Лабораторная работа 2**

1. «Защита программ от нелегального использования»
2. по дисциплине «Практикум по информационной безопасности»

Выполнил:

студент гр. 13558/1 Никулкин В. А.

Проверил:

ассистент преподавателя Алексеев. И. В.

1. Санкт-Петербург
2. 2017
3. **Цель работы**
4. Приобретение навыков защиты приложения от нелегального использования, анализа исполняемых кодов в отсутствие исходных текстов и применения способов защиты программ от дизассемблирования и отладки.

**Ход работы**

1. Задания, связанные с программой, запрашивающей пароли.

Листинг программы, запрашивающей пароль:

#include <stdio.h>

#include "stdlib.h"

#include <locale.h>

#define N 1000

int enter(char \*password)

{

char symbol;

FILE \*in;

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

int i = 0;

char key[N];

while(1)

{

symbol = fgetc(in);

if(symbol == EOF)

break;

key[i] = symbol;

++i;

}

fclose(in);

key[i] = '\0';

printf("Введите пароль: ");

fgets(password, N-1, stdin);

i = 0;

while((key[i] != '\0') && (password[i] != '\0'))

{

if(password[i] != key[i])

{

printf("Bad password\n");

return 1;

}

++i;

}

printf("Welcome!\n");

return 0;

}

int main()

{

int i;

setlocale(LC\_ALL, "Rus");

char password[N];

i = enter(password);

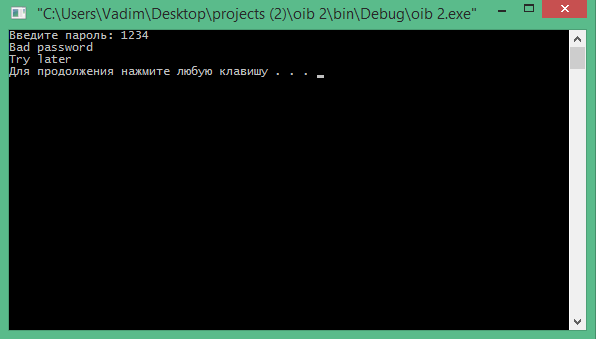
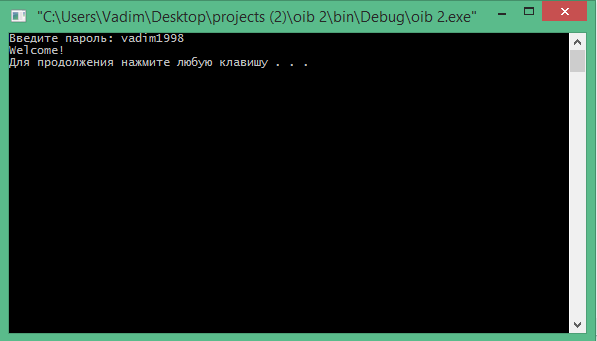
if(i == 1)

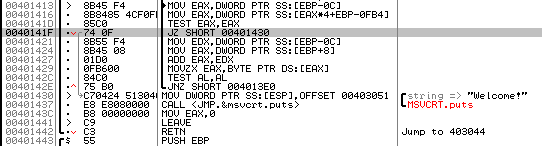
printf("Try later\n");

system("pause");

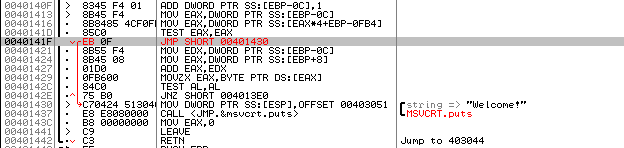
return 0;

}

1. При запуске данной программы у пользователя запрашивается пароль, в данном случае vadim1998. При вводе неправильного пароля пользователю будет отказано в доступе.
2. ****
3. При вводе верного пароля доступ будет разрешен.
4. ****
5. Далее, для того чтобы программа принимала любой пароль как верный, находится, с помощью дизассемблера OllyDbg, команда, отвечающая за переход к области памяти с адресом 00401430, если флаг ZF не равен нулю, то есть был введен верный пароль.

****

Команда JE является командой условного перехода, то есть должно быть соблюдено определенное условие (в данном случае введен правильный пароль). Чтобы переход выполнялся независимо от каких-либо условий, команду JE надо заменить JMP.



Теперь, независимо от введенного пароля, пользователь будет получать доступ.



После оптимизации кода (настройки компилятора были изменены, была включена функция оптимизации сгенерированного кода). В связи с оптимизацией кода по времени, скорость выполнения программы была увеличена. Это влечет за собой то, что в оптимизированном исполняемом файле с целью повышения быстродействия программы или сокращения объема исполняемого файла нарушается прямой (“логичный”) порядок вызова команд и использования регистров. Это несколько усложняет процесс анализа функции и передачи управления внутри кода программы.

Для того, чтобы пароль не хранился в открытом виде, была написана функция, осуществляющая простейшее шифрование с помощью функции XOR.

char \*shifr(char \*password)

{

int f = strlen(password);

char str[f];

int i;

int shifr = 4;

for(i = 0; i < strlen(password); ++i)

str[i] = password[i] ^ shifr;

for(i = 0; i < strlen(password); ++i)

password[i] = str[i];

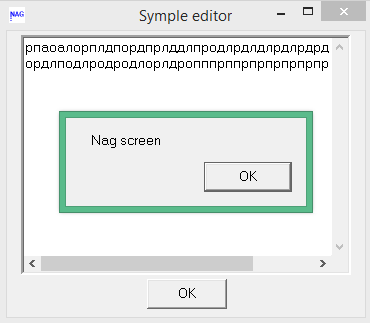
return(password);

}

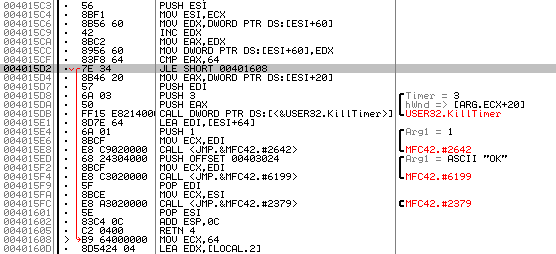
Данное изменение не повлияло на работу программы. Она продолжила правильно выполнять свои функции.

1. Задания, связанные с Nag.exe

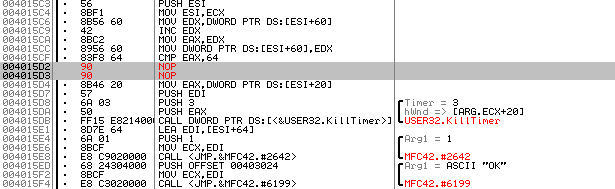
При работе с Nag.exe каждые 8 секунд появляется счетчик, который на некоторое время блокирует работу с программой (в начальный момент значение счётчика равно 99). И такое всплывающее окно появляется многократно.



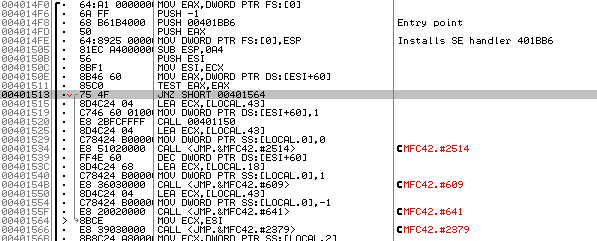
С помощью Ollydbg был найден участок кода программ, отвечающий за счетчик на кнопке



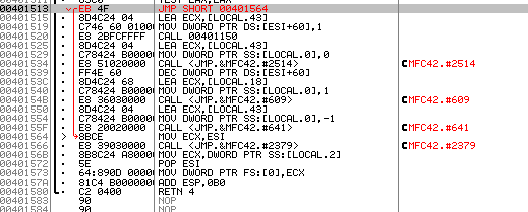
После внесения данных изменения кнопка ОК стала доступна сразу



Далее, для того, чтобы всплывающее окно больше не появлялось, нужно внести изменения в этот участок кода



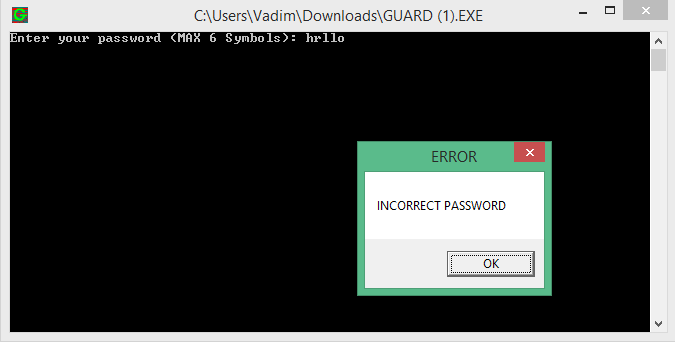
Для того, чтобы всплывающее окно больше не появлялось надо внести такое изменение в код.



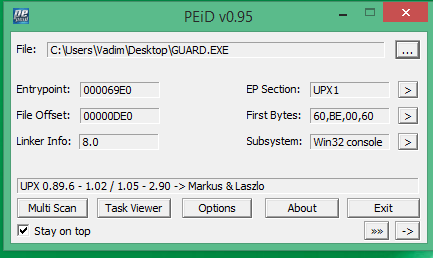
В результате, всплывающее окно больше не появляются.

1. Задания, связанные с Guard.exe

При вводе произвольного пароля в программе Guard.exe был получен отказ в доступе

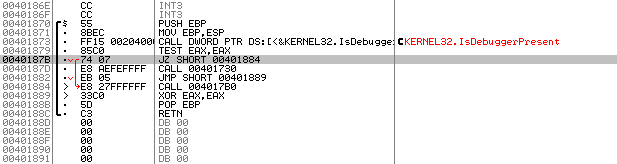


С помощью утилиты PEiD было установлено, что Guard.exe был запакован с помощью утилиты UPX

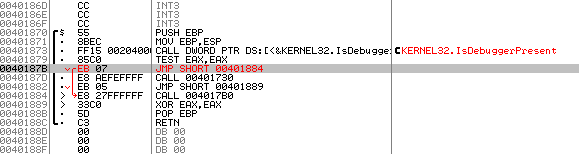


После выяснения того, с помощью чего был запакован Guard, он был распакован и проанализирован.

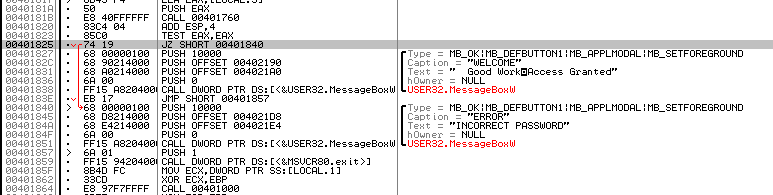
С помощью OllyDBG был найден участок кода, отвечающий за вывод сообщения о наличии отладчика



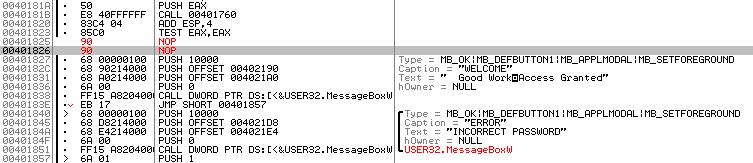
Были внесены данные изменения, после которых сообщение больше не появляется

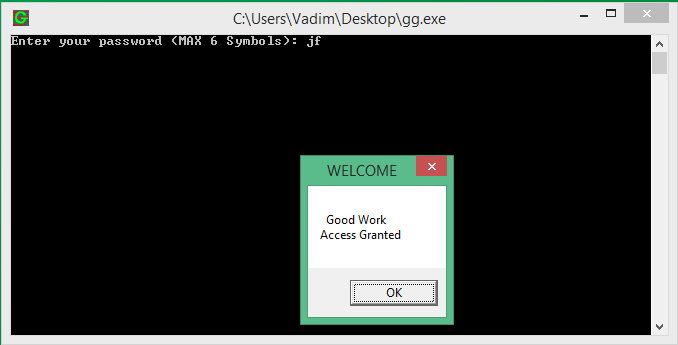


В коде были найдены команды, отвечающие за проверку пароля



После данной модификации происходит обход функции проверки пароля, и теперь доступ можно получить при любом введенном пароле





1. Листинг обфускатора:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void coping(FILE \*in, FILE \*out)

{

in = fopen("output.txt", "r");

out = fopen("test.txt", "w");

char c;

while(1)

{

c = fgetc(in);

if(c == EOF)

break;

fputc(c, out);

}

fclose(in);

fclose(out);

}

char \*renaming2(char \*str, char \*a, char \*b, int length)

{

char buffer[length];

int i, j, pos = 0;

for(i = 0; str[i]; i++)

{

for(j = 0; (str[i + j]) && (a[j]); j++)

if(str[i + j] != a[j])

break;

if(!a[j])

{

i += j - 1;

for(j = 0; b[j]; j++)

buffer[pos++] = b[j];

}

else

buffer[pos++] = str[i];

buffer[pos] = NULL;

}

strcpy(str,buffer);

}

void renaming(FILE \*text)

{

char \*buffer = 0;

long length;

text = fopen ("test.txt", "rb");

FILE \*shifr = fopen("output.txt", "w");

fseek (text, 0, SEEK\_END);

length = ftell (text);

fseek (text, 0, SEEK\_SET);

buffer = (char\*) malloc(sizeof(char) \* length);

fread (buffer, 1, length, text);

renaming2(buffer, "key", "\_\_\_", length);

renaming2(buffer, "enter", "\_\_\_\_\_", length);

renaming2(buffer, "symbol", "\_\_\_\_\_\_", length);

renaming2(buffer, "password", "\_\_\_\_\_\_\_\_", length);

renaming2(buffer, "Bad", "\_w2", length);

renaming2(buffer, "Welcome", "\_dgdg\_\_", length);

renaming2(buffer, "Try", "\_1\_", length);

renaming2(buffer, "later", "\_123\_", length);

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

fputc(buffer[i], shifr);

fclose(text);

fclose(shifr);

coping(shifr, text);

free(buffer);

}

void delete\_space(FILE \*text)

{

text = fopen("test.txt", "r");

FILE \*shifr = fopen("output.txt", "w");

char symbol;

while(1)

{

symbol = fgetc(text);

if(symbol == EOF)

break;

if(symbol == '#')

{

do{

do{

fputc(symbol, shifr);

symbol = fgetc(text);

}while(symbol != '\n');

fputc('\n', shifr);

symbol = fgetc(text);

}while(symbol == '#');

}

while((symbol == '\n') || (symbol == '\t'))

symbol = fgetc(text);

if(symbol == EOF)

break;

fputc(symbol, shifr);

}

fclose(text);

fclose(shifr);

coping(shifr, text);

}

void delete\_space2(FILE \*text)

{

text = fopen("test.txt", "r");

FILE \*shifr = fopen("output.txt", "w");

char symbol;

while(1)

{

symbol = fgetc(text);

if(symbol == EOF)

break;

if(symbol == '(')

{

fputc(symbol, shifr);

symbol = fgetc(text);

while(symbol == '(')

{

fputc(symbol, shifr);

symbol = fgetc(text);

}

while((symbol == ' ') || (symbol == '\t'))

symbol = fgetc(text);

}

if(symbol == ')')

{

fputc(symbol, shifr);

symbol = fgetc(text);

while(symbol == ')')

{

fputc(symbol, shifr);

symbol = fgetc(text);

}

while((symbol == ' ') || (symbol == '\t'))

symbol = fgetc(text);

}

if(symbol == '{')

{

fputc(symbol, shifr);

symbol = fgetc(text);

while(symbol == '{')

{

fputc(symbol, shifr);

symbol = fgetc(text);

}

while((symbol == ' ') || (symbol == '\t'))

symbol = fgetc(text);

}

if(symbol == '}')

{

fputc(symbol, shifr);

symbol = fgetc(text);

while(symbol == '}')

{

fputc(symbol, shifr);

symbol = fgetc(text);

}

while((symbol == ' ') || (symbol == '\t'))

symbol = fgetc(text);

}

if(symbol == EOF)

break;

fputc(symbol, shifr);

}

fclose(text);

fclose(shifr);

coping(shifr, text);

}

void delete\_space3(FILE \*text)

{

text = fopen("test.txt", "r");

FILE \*shifr = fopen("output.txt", "w");

char symbol;

while(1)

{

symbol = fgetc(text);

if(symbol == EOF)

break;

if(symbol == ';')

{

fputc(symbol, shifr);

symbol = fgetc(text);

while((symbol == ' ') || (symbol == '\t'))

symbol = fgetc(text);

}

if(symbol == EOF)

break;

fputc(symbol, shifr);

}

fclose(text);

fclose(shifr);

coping(shifr, text);

}

void degraph(FILE \*text)

{

text = fopen("test.txt", "r");

FILE \*shifr = fopen("output.txt", "w");

char symbol;

while(1)

{

symbol = fgetc(text);

if(symbol == EOF)

break;

if(symbol == '#')

{

fputc('%', shifr);

fprintf(shifr, ":");

symbol = fgetc(text);

}

if(symbol == '[')

{

fprintf(shifr, "<:");

symbol = fgetc(text);

}

if(symbol == ']')

{

fprintf(shifr, ":>");

symbol = fgetc(text);

}

if(symbol == '{')

{

fprintf(shifr, "<");

fputc('%', shifr);

symbol = fgetc(text);

}

if(symbol == '}')

{

fputc('%', shifr);

fprintf(shifr, ">");

symbol = fgetc(text);

}

if(symbol == EOF)

break;

fputc(symbol, shifr);

}

fclose(text);

fclose(shifr);

coping(shifr, text);

}

void delete\_comments(FILE \*in)

{

in = fopen("test.txt", "r+");

FILE \*out = fopen("output.txt", "w");

char symbol;

symbol = fgetc(in);

while (1)

{

if(symbol == EOF)

break;

if(symbol == '/')

{

symbol = fgetc(in);

if(symbol == '/')

{

do{

symbol = fgetc(in);

}while(symbol != '\n');

}

else if(symbol == '\*')

{

while(1)

{

if(symbol == '\*')

{

symbol = fgetc(in);

if(symbol == '/')

break;

}

symbol = fgetc(in);

}

symbol = fgetc(in);

}

}

fputc(symbol, out);

symbol = fgetc(in);

}

fclose(in);

fclose(out);

coping(out, in);

}

int main()

{

FILE \*test;

delete\_comments(test);

delete\_space(test);

delete\_space2(test);

delete\_space3(test);

degraph(test);

renaming(test);

return 0;

}

Для программы, которая считывает пароль, был написан обфускатор, который удаляет излишние пробельные символы (пробелы, вертикальную и горизонтальную табуляцию), комментарии, заменяет имена переменных и функций, заменяет символы {, }, [, ], # на эквивалентные диграфы, сохраняя первоначальную логику работы программы.

Контрольные вопросы:

1. Для защиты коммерческих программ от ввода некорректных учетных данных применяются такие методы защиты, как: запрос пароля у пользователя, если введенный пароль не верен, то в доступе ему будет отказано; использование шифрование пароля, хранящегося в исходном файле, чтобы он не хранился в открытом виде.
2. Код содержит команду условного перехода, который осуществляется в том случае, если установлен соответствующий флаг (то есть пароли совпадают или наоборот). Командами условного перехода являются, например, команды JZ и JE.
3. Основными методами защиты от дизассемблирования и отладки программ являются: использование упаковщика, для упаковки EXE – файла; применение оптимизирующего компилятора; применение шифрования.
4. При таком подходе программа, которую необходимо обфусцировать, преобразуется в байт-код для некоторой виртуальной машины, которая имеет интерфейс для её исполнения. Взломщик может дизассемблировать код исполнителя виртуальной машины, но это не поможет понять ему логику программы, так как программа хранится в байт-коде, а для него дизассемблиров не существует.
5. Методы усиления механизмов защиты программ от нелегального использования: для того, чтобы получить доступ обязательно нужно интернет соединение, так как пароль будет храниться на веб-сервере; использование программ с всплывающими окнами, причем может всплывать сразу несколько окон и период их существования будет разным. Также при этом будет сложно изменять участки кода с помощью дизассемблера, так как, логика программы становится менее понятной. Также можно ограничивать число запусков программы.

**Вывод**

Для защиты программ от незаконного использования, используются такие методы, как: запрос пароля, обфускация кода, использование программ с всплывающими окнами, использования упаковщиков. При этом эти методы защиты часто можно легко обойти с помощью дизассемблера и отладчика, поэтому для обеспечения высокого уровня безопасности следует использовать сразу несколько методов защиты.