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

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**Изучение принципов поиска уязвимостей в программном обеспечении без исходных кодов**

по дисциплине «Модели безопасности компьютерных систем»

1. Выполнила
2. студентка гр. 4851003/90801 Кулеева А.Г.

1. Руководитель
2. ст. преподаватель Овасапян Т.Д.
4. Санкт-Петербург
5. 2021

# Цель работы

Изучение типовых ошибок и принципов поиска уязвимостей в программном обеспечении без исходных кодов.

# Ход работы

Была получена уязвимая программа и её конфигурационный файл в соответствии с вариантом.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 — Программа, полученная по варианту

Для начала с помощью утилиты DIE (Detect It Easy) была получена информация об исполняемом файле (Рисунок 2).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 ― Информация об исполняемом файле

Таким образом удалось выяснить, тип файла и битность. Исследуем файл с помощью IDA Pro x32. Утилита открывает функцию main, в ней сразу же была найдена функция считывания конфига в одну строку, размер которой зависит от размера файла. Первые 47 символов являются хранителями информации о файле. При этом самый первый байт соответствует варианту, байты 0x08 0x09 0x0A 0x0B являются значением переменной dst\_len, байты 0x10 0x11 определяют значение переменной shell\_len (Рисунок 1). В ходе анализа функции parse\_string удалось обнаружить, что с помощью функции strstr ищется вхождение строки /start в конфиге. При нахождении осуществляется переход в vuln\_func\_sh (Рисунок 3). В ней происходит копирование строки после старта, при этом, если длина строки превышает значение 2500 символов, то выводится соответствующая ошибка. Таким образом, размер буфера 2500 символов, и он располагается по адресу -000009CC.

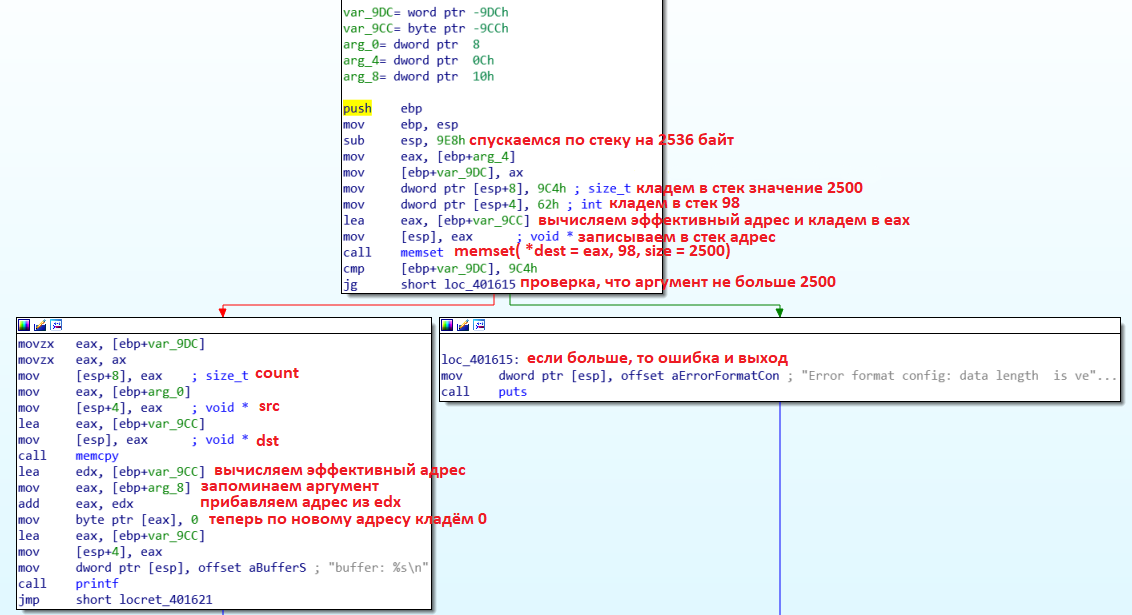


Рисунок 3 ― Уязвимая функция

Также с помощью IDA удалось восстановить код уязвимой функции на языке Си:

int \_\_cdecl vuln\_func\_sh(void \*a1, int a2, int a3)

{

char v4[2508]; // [esp+1Ch] [ebp-9CCh]

memset(v4, 98, 2500u);

if ( (signed \_\_int16)a2 > 2500 )

return puts("Error format config: data length is very long");

memcpy(v4, a1, (unsigned \_\_int16)a2);

v4[a3] = 0;

return printf("buffer: %s\n", v4);

}

Исходя из данного анализа, можно сформулировать, каким будет фаззер, способный «сломать программу». Поскольку нигде в коде нет проверки реальной длины строки и данные о ней и максимальной допустимой длине поступают из конфига, то достаточно просто изменить значения переменных shell\_len и dst\_len.

Был написан соответствующий фаззер на языке Си, эксплуатирующий уязвимость целочисленное переполнение. Его код представлен в Приложении 1. Результат работы программы представлен на Рисунке 3. Вызов программы vuln7.exe зависает в консоли.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 ― Демонстрация работы программы

Также ход работы фаззера логировался в файл. Пример части лога представлен на Рисунке 4.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 — Логирование процесса фазинга

Кроме того, был написан скрипт IDAPython, с помощью которого в листинге из IDA Pro можно найти уязвимые функции и функции ввода-вывода. Код скрипта представлен в Приложении 2, результат работы на Рисунке 5.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 5 — Скрипт поиска функций

ВЫВОД

В ходе выполнения данной лабораторной работы были изучены принципы поиска уязвимостей в программном обеспечении без исходных кодов. Был реализован собственный фаззер на языке Си, который эксплуатирует уязвимость целочисленного переполнения буфера. С помощью подмены байт на граничные значения удалось вызвать исключение в уязвимой программе. Для анализа программы были применены полученные в прошлом семестре навыки реверс инжиниринга.

Также были изучены некоторые функции WinAPI, с помощью которых удалось реализовать удобное логирование исключения и состояния регистров.

Кроме того, были приобретены навыки по реализации скриптов IDAPython. При изучении теории по данному пункту лабораторной работы были изучены некоторые особенности IDA API. Также в качестве примера были изучены некоторые скрипты, которые значительно упрощают работу в данной программе.

# Приложение 1

#include <stdio.h>

#include <process.h>

#include <stdbool.h>

#include <string.h>

#include <stdlib.h>

#include <windows.h>

#include <winnt.h>

#include <dbghelp.h>

#include <processthreadsapi.h>

#include <tlhelp32.h>

//#include <psapi.h>

#pragma comment(lib, "dbghelp.lib")

unsigned int menu() {

puts("Choose what you like to do:\n"

//"0: Add to file\n"

"1: Change 1 byte\n"

"2: Change 2 bytes\n"

"3: Change 3 bytes\n"

"4: Change 4 bytes\n");

return getc(stdin) - 48;

}

bool get\_info(unsigned int\* shell\_len, unsigned int\* dst\_len) {

FILE\* f;

unsigned int buf[12] = { 0 };

if (fopen\_s(&f, "config\_7", "rb")) {

puts("Error open config file.");

return false;

}

fseek(f, SEEK\_SET, SEEK\_CUR);

fread(buf, sizeof(unsigned int), 12, f);

//for (unsigned int i = 0; i < 12; i++)

//printf("%u ", buf[i]);

\*shell\_len = buf[1];

\*dst\_len = buf[2];

fclose(f);

return true;

}

void Get\_Registers\_State(CONTEXT\* hContext, const char\* error, HANDLE hProcess) {

FILE\* log;

if (fopen\_s(&log, "log", "a")) {

puts("Error open log file.");

exit(-1);

}

fprintf(log, "\nException: %s\n", error);

fprintf(log, "EAX: 0x%p ESP: 0x%p\n", (void\*)hContext->Eax, (void\*)hContext->Esp);

fprintf(log, "EBX: 0x%p EBP: 0x%p\n", (void\*)hContext->Ebx, (void\*)hContext->Ebp);

fprintf(log, "ECX: 0x%p EDI: 0x%p\n", (void\*)hContext->Ecx, (void\*)hContext->Edi);

fprintf(log, "EDX: 0x%p ESI: 0x%p\n", (void\*)hContext->Edx, (void\*)hContext->Esi);

fprintf(log, "EIP: 0x%p FLG: 0x%p\n", (void\*)hContext->Eip, (void\*)hContext->EFlags);

fclose(log);

}

void run\_program() {

HANDLE hThread;

PROCESS\_INFORMATION proc\_info;

STARTUPINFO startup\_info;

BOOL status;

CONTEXT hContext;

DEBUG\_EVENT debug\_event = { 0 };

RtlZeroMemory(&startup\_info, sizeof(startup\_info));

startup\_info.cb = sizeof(startup\_info);

status = CreateProcessA((LPCSTR)"vuln7.exe",

NULL, NULL, NULL, FALSE, DEBUG\_PROCESS, NULL, NULL, (LPSTARTUPINFOA)&startup\_info, &proc\_info);

if (status == false) {

puts("CreateProcess failed.");

exit(-1);

}

while (true) {

// ожидаем событие отладки

status = WaitForDebugEvent(&debug\_event, 500);

if (status == false) {

if (GetLastError() != ERROR\_SEM\_TIMEOUT)

puts("WaitForDebugEvent failed.");

break;

}

// смотрим код события

if (debug\_event.dwDebugEventCode != EXCEPTION\_DEBUG\_EVENT) {

// если это не исключение - продолжаем ожидать

ContinueDebugEvent(debug\_event.dwProcessId, debug\_event.dwThreadId, DBG\_CONTINUE);

continue;

//return 0;

}

hThread = OpenThread(THREAD\_ALL\_ACCESS, FALSE, debug\_event.dwThreadId);

if (hThread == NULL) {

puts("OpenThread failed.");

break;

}

hContext.ContextFlags = CONTEXT\_FULL;

// по хэндлу получаем его контекст

status = GetThreadContext(hThread, &hContext);

if (status == false) {

puts("GethThreadContext failed.");

CloseHandle(hThread);

break;

}

switch (debug\_event.u.Exception.ExceptionRecord.ExceptionCode) {

case EXCEPTION\_ACCESS\_VIOLATION:

Get\_Registers\_State(&hContext, "EXCEPTION\_ACCESS\_VIOLATION", proc\_info.hProcess);

break;

case EXCEPTION\_STACK\_OVERFLOW:

Get\_Registers\_State(&hContext, "EXCEPTION\_STACK\_OVERFLOW", proc\_info.hProcess);

break;

case EXCEPTION\_ARRAY\_BOUNDS\_EXCEEDED:

Get\_Registers\_State(&hContext, "EXCEPTION\_ARRAY\_BOUNDS\_EXCEEDED", proc\_info.hProcess);

break;

case EXCEPTION\_DATATYPE\_MISALIGNMENT:

Get\_Registers\_State(&hContext, "EXCEPTION\_DATATYPE\_MISALIGNMENT", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_DENORMAL\_OPERAND:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_DENORMAL\_OPERAND", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_DIVIDE\_BY\_ZERO", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_INEXACT\_RESULT:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_INEXACT\_RESULT", proc\_info.hProcess); \

break;

case EXCEPTION\_FLT\_INVALID\_OPERATION:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_INVALID\_OPERATION", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_OVERFLOW:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_OVERFLOW", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_STACK\_CHECK:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_STACK\_CHECK", proc\_info.hProcess);

break;

case EXCEPTION\_FLT\_UNDERFLOW:

Get\_Registers\_State(&hContext, "EXCEPTION\_FLT\_UNDERFLOW", proc\_info.hProcess);

break;

case EXCEPTION\_ILLEGAL\_INSTRUCTION:

Get\_Registers\_State(&hContext, "EXCEPTION\_ILLEGAL\_INSTRUCTION", proc\_info.hProcess);

break;

case EXCEPTION\_IN\_PAGE\_ERROR:

Get\_Registers\_State(&hContext, "EXCEPTION\_IN\_PAGE\_ERROR", proc\_info.hProcess);

break;

case EXCEPTION\_INT\_DIVIDE\_BY\_ZERO:

Get\_Registers\_State(&hContext, "EXCEPTION\_INT\_DIVIDE\_BY\_ZERO", proc\_info.hProcess);

break;

case EXCEPTION\_INT\_OVERFLOW:

Get\_Registers\_State(&hContext, "EXCEPTION\_INT\_OVERFLOW", proc\_info.hProcess);

break;

case EXCEPTION\_INVALID\_DISPOSITION:

Get\_Registers\_State(&hContext, "EXCEPTION\_INVALID\_DISPOSITION", proc\_info.hProcess);

break;

case EXCEPTION\_NONCONTINUABLE\_EXCEPTION:

Get\_Registers\_State(&hContext, "EXCEPTION\_NONCONTINUABLE\_EXCEPTION", proc\_info.hProcess);

break;

case EXCEPTION\_PRIV\_INSTRUCTION:

Get\_Registers\_State(&hContext, "EXCEPTION\_PRIV\_INSTRUCTION", proc\_info.hProcess);

break;

case EXCEPTION\_SINGLE\_STEP:

Get\_Registers\_State(&hContext, "EXCEPTION\_SINGLE\_STEP", proc\_info.hProcess);

break;

default:

//cout << "Unknown exception: " << dec << debug\_event.u.Exception.ExceptionRecord.ExceptionCode << endl;

ContinueDebugEvent(debug\_event.dwProcessId, debug\_event.dwThreadId, DBG\_CONTINUE);

}

}

CloseHandle(proc\_info.hProcess);

}

bool fuz(unsigned int task) {

FILE\* f;

FILE\* log;

signed \_\_int16 shell\_len = 0;

DWORD dst\_len = 0;

unsigned char bufer[48];

unsigned int amount;

unsigned int pos;

switch (task)

{

case 1: {//change 1 byte

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

if (fopen\_s(&log, "log", "w")) {

puts("Error open log file.");

return false;

}

unsigned char change\_1byte[4] = { '\x00', '\x80', '\x7f', '\xff'};

amount = 1;

for (unsigned int i = 0; i < 4; i++) {

fprintf(log, "\n\nChange %d byte for %02x\n", amount, (unsigned int)change\_1byte[i]);

fseek(f, 0, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_SET); //switched from write to read

fread(bufer, sizeof(char), 48, f); //read bufer

fprintf(log, "Original string: ");

for (unsigned int h = 0; h < 48; h++)

fprintf(log, "%02x ", (unsigned int)bufer[h]);

fseek(f, 0, SEEK\_SET); //switched from read to write

for (unsigned int j = 0; j < 48; j++) {

pos = ftell(f);

fwrite(&change\_1byte[i], sizeof(unsigned char), amount, f); //put new symbol

fprintf(log, "\nNew string: ");

for (unsigned int h = 0; h < 48; h++) {

if (h == j) {

fprintf(log, "%02x ", (unsigned int)change\_1byte[i]);

}

else fprintf(log, "%02x ", (unsigned int)bufer[h]);

}

fseek(f, sizeof(signed \_\_int16) \* 8, SEEK\_SET); //switched from write to read

fread(&shell\_len, sizeof(signed \_\_int16), 1, f);

fseek(f, sizeof(DWORD) \* 2, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_CUR); //switched from write to read

fread(&dst\_len, sizeof(DWORD), 1, f);

fprintf(log, "\nshell\_len = %d, dst\_len = %d", shell\_len, dst\_len);

fclose(f);

fclose(log);

run\_program("vuln7.exe"); //test

if (fopen\_s(&log, "log", "a")) {

puts("Error open log file.");

return false;

}

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

fseek(f, pos, SEEK\_CUR); //switched from read to write

fputc(bufer[j], f); //restore original string

}

}

fclose(f);

fclose(log);

return true;

}

case 2: {//change 2 bytes

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

if (fopen\_s(&log, "log", "w")) {

puts("Error open log file.");

return false;

}

unsigned char change\_2bytes[4][2] = { "\x00\x00", "\x80\x00", "\x7f\xff", "\xff\xff" }; //{"0x00000000", "0x80000000", "0x7FFFFFFF", "0xFFFFFFFF"};

amount = 2;

for (unsigned int i = 0; i < 4; i++) {

fprintf(log, "\n\nChange %d bytes for %02x %02x\n", amount, (unsigned int)change\_2bytes[i][0],

(unsigned int)change\_2bytes[i][1]);

fseek(f, 0, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_SET); //switched from write to read

fread(bufer, sizeof(char), 48, f); //read bufer

fprintf(log, "Original string: ");

for (unsigned int h = 0; h < 48; h++)

fprintf(log, "%02x ", (unsigned int)bufer[h]);

fseek(f, 0, SEEK\_SET); //switched from read to write

for (unsigned int j = 0; j < 48; j += amount) {

pos = ftell(f);

fwrite(&change\_2bytes[i], sizeof(unsigned char), amount, f); //put new symbol

fprintf(log, "\nNew string: ");

for (unsigned int h = 0; h < 48; h++) {

if (h == j) {

fprintf(log, "%02x ", (unsigned int)change\_2bytes[i][0]);

fprintf(log, "%02x ", (unsigned int)change\_2bytes[i][1]);

h++;

}

else fprintf(log, "%02x ", (unsigned int)bufer[h]);

}

fseek(f, sizeof(signed \_\_int16) \* 8, SEEK\_SET); //switched from write to read

fread(&shell\_len, sizeof(signed \_\_int16), 1, f);

fseek(f, sizeof(DWORD) \* 2, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_CUR); //switched from write to read

fread(&dst\_len, sizeof(DWORD), 1, f);

fprintf(log, "\nshell\_len = %d, dst\_len = %d", shell\_len, dst\_len);

fclose(f);

fclose(log);

run\_program("vuln7.exe"); //test

if (fopen\_s(&log, "log", "a")) {

puts("Error open log file.");

return false;

}

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

fseek(f, pos, SEEK\_CUR); //switched from read to write

fputc(bufer[j], f); //restore original string

fputc(bufer[j + 1], f);

}

}

fclose(f);

fclose(log);

return true;

}

case 3: {//change 3 bytes

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

if (fopen\_s(&log, "log", "w")) {

puts("Error open log file.");

return false;

}

unsigned char change\_3bytes[4][3] = { "\x00\x00\x00", "\x80\x00\x00", "\x7f\xff\xff", "\xff\xff\xff" }; //{"0x00000000", "0x80000000", "0x7FFFFFFF", "0xFFFFFFFF"};

amount = 3;

for (unsigned int i = 0; i < 4; i++) {

fprintf(log, "\n\nChange %d bytes for %02x %02x %02x\n", amount, (unsigned int)change\_3bytes[i][0],

(unsigned int)change\_3bytes[i][1], (unsigned int)change\_3bytes[i][2]);

fseek(f, 0, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_SET); //switched from write to read

fread(bufer, sizeof(char), 48, f); //read bufer

fprintf(log, "Original string: ");

for (unsigned int h = 0; h < 48; h++)

fprintf(log, "%02x ", (unsigned int)bufer[h]);

fseek(f, 0, SEEK\_SET); //switched from read to write

for (unsigned int j = 0; j < 48; j += amount) {

pos = ftell(f);

fwrite(&change\_3bytes[i], sizeof(unsigned char), amount, f); //put new symbol

fprintf(log, "\nNew string: ");

for (unsigned int h = 0; h < 48; h++) {

if (h == j) {

for (unsigned int k = 0; k < amount; k++)

fprintf(log, "%02x ", (unsigned int)change\_3bytes[i][k]);

h += 2;

}

else fprintf(log, "%02x ", (unsigned int)bufer[h]);

}

fseek(f, sizeof(signed \_\_int16) \* 8, SEEK\_SET); //switched from write to read

fread(&shell\_len, sizeof(signed \_\_int16), 1, f);

fseek(f, sizeof(DWORD) \* 2, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_CUR); //switched from write to read

fread(&dst\_len, sizeof(DWORD), 1, f);

fprintf(log, "\nshell\_len = %d, dst\_len = %d", shell\_len, dst\_len);

fclose(f);

fclose(log);

run\_program("vuln7.exe"); //test

if (fopen\_s(&log, "log", "a")) {

puts("Error open log file.");

return false;

}

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

fseek(f, pos, SEEK\_CUR); //switched from write to read

fseek(f, 0, SEEK\_CUR); //switched from read to write

fputc(bufer[j], f);

fputc(bufer[j + 1], f);

fputc(bufer[j + 2], f);

}

}

fclose(f);

fclose(log);

return true;

}

case 4: {//change 4 bytes

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

if (fopen\_s(&log, "log", "w")) {

puts("Error open log file.");

return false;

}

unsigned char change\_4bytes[4][4] = { "\x00\x00\x00\x00", "\x80\x00\x00\x00", "\x7f\xff\xff\xff", "\xff\xff\xff\xff" }; //{"0x00000000", "0x80000000", "0x7FFFFFFF", "0xFFFFFFFF"};

for (unsigned int i = 0; i < 4; i++) {

unsigned int amount = 4;

fprintf(log, "\n\nChange %d bytes for %02x %02x %02x %02x\n", amount, (unsigned int)change\_4bytes[i][0],

(unsigned int)change\_4bytes[i][1], (unsigned int)change\_4bytes[i][2], (unsigned int)change\_4bytes[i][3]);

fseek(f, 0, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_SET); //switched from write to read

fread(bufer, sizeof(char), 48, f); //read bufer

fprintf(log, "Original string: ");

for (unsigned int h = 0; h < 48; h++)

fprintf(log, "%02x ", (unsigned int)bufer[h]);

fseek(f, 0, SEEK\_SET); //switched from read to write

for (unsigned int j = 0; j < 48; j += amount) {

pos = ftell(f);

fwrite(&change\_4bytes[i], sizeof(unsigned char), amount, f); //put new symbol

fprintf(log, "\nNew string: ");

for (unsigned int h = 0; h < 48; h++) {

if (h == j) {

for (unsigned int k = 0; k < amount; k++)

fprintf(log, "%02x ", (unsigned int)change\_4bytes[i][k]);

h += 3;

}

else fprintf(log, "%02x ", (unsigned int)bufer[h]);

}

fseek(f, sizeof(signed \_\_int16) \* 8, SEEK\_SET); //switched from write to read

fread(&shell\_len, sizeof(signed \_\_int16), 1, f);

fseek(f, sizeof(DWORD) \* 2, SEEK\_SET); //switched from read to write

fseek(f, 0, SEEK\_CUR); //switched from write to read

fread(&dst\_len, sizeof(DWORD), 1, f);

fprintf(log, "\nshell\_len = %d, dst\_len = %d", shell\_len, dst\_len);

fclose(f);

fclose(log);

run\_program("vuln7.exe"); //test

if (fopen\_s(&log, "log", "a")) {

puts("Error open log file.");

return false;

}

if (fopen\_s(&f, "config\_7", "r+b")) {

puts("Error open config file.");

return false;

}

fseek(f, pos, SEEK\_CUR); //switched from write to read

fseek(f, 0, SEEK\_CUR); //switched from read to write

fputc(bufer[j], f);

fputc(bufer[j + 1], f);

fputc(bufer[j + 2], f);

fputc(bufer[j + 3], f);

}

}

fclose(f);

fclose(log);

}

default:

break;

}

return true;

}

int main() {

unsigned int shell\_len = 0, dst\_len = 0;

if (get\_info(&shell\_len, &dst\_len)) {

puts("Information was successfully read.");

unsigned int task = menu();

if (fuz(task)) {

puts("Fuzzing complete.");

return 0;

}

else return -1;

}

else return -1;

}

# Приложение 2

from idautils import \*

from idc import \*

ea = ScreenEA()

unsafe\_func = ["strcpy", "strcat", "strtok", "sprintf", "gets", "scanf", "sscanf", "snscanf", "strlen", "memcpy", "memmove"]

io\_func = ["fread", "fscanf", "read", "fgets", "fgetc", "fgetchar", "\_fgetchar", "\_fgetwchar", "fputc", "fputchar", "\_fputchar", "\_fputwchar", "fputs", "fprintf", "getchar", "getwchar", "gets\_s", "\_getws\_s", "printf", "\_printf\_l", "wprintf", "\_wprintf\_l", "putc", "putwc", "putchar", "putwchar", "scanf", "\_scanf\_l", "wscanf", "\_wscanf\_l", "write"]

for funcea in Functions(SegStart(ea), SegEnd(ea)):

name = GetFunctionName(funcea)

if name in unsafe\_func or name in io\_func:

print (str(name) + "found at %X" % funcea)