МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ
НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
« ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

А.Н. Рысованый

РЕВЕРСНОЕ ПРОГРАММИРОВАНИЕ

Внедрение кода.
Среда программирования masm64

Учебное пособие
для студентов специальности 123 – «Компьютерная инженерия»
и специальности 125 – «Кибербезопасность»

Утверждено
редакционно-издательским
советом университета,
протокол №7 от 02.07.2021

Харьков
2021

ББК 32.973-018.1
УДК 004.42
Р54

Рецензенты:

А.В. Шостак, канд. техн. наук, доцент Национального аэрокосмического университета им. Н.Е. Жуковского "Харьковский авиационный институт";

Г.А. Кучук, доктор техн. наук, профессор НТУ ХПИ

Рысованый А.Н.

Р54 Реверсное программирование. Внедрение кода. Среда программирования masm64 : учебное пособие для студентов специальностей 123 «Компьютерная инженерия», 125 «Кибербезопасность»/ А.Н. Рысованый. – Харьков : «Слово», 2021. – 250 с.– На рус. яз.

Основное внимание уделено внедрению кода в exe-файлы как инструкций, так и API-функций в ОС Windows 10 при программировании на языке Ассемблер в среде masm64. Внедрение осуществляется при помощи отладчика x64Dbg и программ-анализаторов кода. Рассмотрены вопросы исследования заголовка PE-файла, а также различные варианты внедрения в зависимости от наличия свободного места в различных секциях: как кода, так и данных. Приведены материалы, особенности использования отладчика x64Dbg, программ LordPE, Resourse Hacker, CFF Explorer, Hiew 8.43, PE.Tools.v1.9.762.2018, FlexHex, упаковщиков UPX, ASPack, PECompact, протекторов ORiEN, VMProtect, ASProtect, Enigma а так же лабораторные работы с заданиями и примерами выполнения в среде masm64.

Приведена литература, которая используется при изучении рассматриваемого материала.

Предназначено для студентов специальности 123 – «Компьютерная инженерия», 125 «Кибербезопасность».

Ил. 165. Библиогр. 10 назв.

УДК 004.42
ББК 32.973-018.1
© А.Н. Рысованый, 2021

СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ 5
1. СОЗДАНИЕ ИКОНКИ 7
2. ИЗМЕНЕНИЕ ЯЗЫКА ИНТЕРФЕЙСА 14
3. ПЕРЕИМЕНОВАНИЕ ПРОГРАММНОГО СЕГМЕНТА 16

4. PE-ФОРМАТ 18
4.1. Общие сведения о PE-формате 18
4.2. DOS MZ-заголовок 20
4.3. PE-заголовок 22
4.3.1. Общие сведения о PE-заголовке 22
4.3.2. Директории данных DataDirectory 26
4.3.3. Структура IMAGE_SECTION_HEADER 35
4.4. Секции PE-файла 36
4.5. Пример исследования PE-файла 40
4.6. Исследование диалоговой программы 49
4.7. Лабораторная робота «Исследование PE-формата» 64

5. СТРУКТУРЫ PEB И TEB 70
5.1. Использование структуры PEB функцией IsDebuggerPresent 72
5.2. Использование глобальной системной переменной NtGlobalFlag из структуры PEB 73
5.3. Получение адресов системных функций 74

6. ВНЕДРЕНИЕ КОДА В EXE-ФАЙЛ В КОНЕЦ СЕКЦИИ КОДА С ПОМОЩЬЮ ОТЛАДЧИКА X64DBG БЕЗ ДОБАВЛЕНИЯ API- ФУНКЦИЙ 76
6.1. Алгоритм внедрения кода 76
6.2. Пример внедрения кода 77
6.3. Внедрение кода в диалоговую программу 93
6.4. Лабораторная работа «Внедрение кода в exe-файл в конец секции без добавления новых API-функций» 99

7. ВНЕДРЕНИЕ КОДА В EXE-ФАЙЛ В КОНЕЦ СЕКЦИИ КОДА С ДОБАВЛЕНИЕМ НОВЫХ API-ФУНКЦИЙ 105
7.1. Общие сведения 105
7.2. Внедрение кода с добавлением API-функций 105
7.3. Внедрение кода в диалоговую программу 115
7.4. Лабораторная работа «Внедрение кода в exe-файл в конец секции с добавлением новых API-функций» 123

8. ВНЕДРЕНИЕ КОДА В НОВУЮ СЕКЦИЮ КОДА 136
8.1. Основной алгоритм внедрения в новую секцию кода 136
8.2. Внедрение в новую секцию кода 136
8.3. Лабораторная работа «Добавление новой секции» 150

9. ВНЕДРЕНИЕ КОДА В РАЗЛИЧНЫЕ СЕКЦИИ ДАННЫХ 154
9.1. Общие сведения 154
9.2. Алгоритм внедрения кода в секции данных 156
9.3. Создание программ с несколькими диалоговыми окнами 165
9.3.1. Создание программы с тремя диалоговыми окнами, одно из которых – окно ввода пароля 165
9.3.2. Создание программы с последовательным вызовом диалоговых окон 176
9.3.3. Создание программы с параллельным вызовом диалоговых окон 183
9.3.4. Создание нестандартного окна, представленного в виде BMP-файла 192
9.4. Лабораторная работа «Внедрение кода в секции данных» 196

10. ВНЕДРЕНИЕ КОДА. ИЗМЕНЕНИЕ ТОЧКИ ВХОДА В ПРОГРАММУ 200
10.1. Алгоритм изменения точки входа в программу 200
10.2. Лабораторная работа «Изменение точки входа» 209

11. ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ 211
11.1. Общие сведения о динамических библиотеках 211
11.2. Типы динамических библиотек 212
11.3. Создание динамического файла 213
11.4. Точка входа DLL 217
11.5. Создание нового импорта в DLL 223

12. УПАКОВЩИКИ И ПРОТЕКТОРЫ 225
12.1. Общие сведения 225
12.2. Сжатие данных без потерь 226
12.3. Упаковщики 231
12.3.1. Упаковщик UPX 232
12.3.2. Упаковщик ASPack 236
12.3.3. Упаковщик PECompact 238
12.4. Программы-протекторы 238
12.4.1. Назначение программ-протекторов 238
12.4.2. Основные характеристики программ-протекторов 239
12.4.3. Протектор VMProtect 244
12.4.4. Протектор ASProtect 245
12.4.4. Протектор Enigma 247

СПИСОК ЛИТЕРАТУРЫ 249

ВСТУПЛЕНИЕ

Реинжиниринг программного обеспечения – процесс создания новой функциональности или устранения ошибок, путём изменения исполняемого файла. Процесс реинжиниринга описан Чиковски и Кроссом в их труде 1990 года как «The examination and alteration of a system to reconstitute it in a new form». Задача реверсивного (реверсного) программирования – получение исходного кода из исполняемого файла исследуемой программы.

Реинжиниринг часто называют обратным инжинирингом (англ. reverse engineering) – это исследование готового программного продукта с целью определения принципа его работы, обнаружения недокументированных возможностей (в том числе программные закладки), а также внесение изменений в исполняемый файл.

Реинжиниринг применяется в случае, если создатель оригинального программного продукта не предоставил информации о структуре и способе создания объекта, а также для изменения его функционирования. Почти всегда правообладатели могут заявить, что проведение обратной разработки или использование её результатов нарушает их авторское право.

Анализом кода занимаются не только антивирусные лаборатории, выпускающие свои продукты для защиты, но и группы экспертов, стремящихся быть в курсе атак, и даже вирусописатели, которые конкурируют между собой. А от вирусного аналитика требуется кропотливая работа, требующая нестандартных подходов [1-3, 9].

Для полноценного реинжиниринга исполняемого файла применяется язык Ассемблер, т.к. только этот язык является низкоуровневым машинным языком, который и понимает компьютер. И делать это могут только высоко квалифицированные специалисты.

Ассемблер (Assembly) – язык программирования, применение которого основывается на архитектуре компьютера и представляет собой символьную форму записи машинного кода.

Существенные преимущества знания ассемблера:

1. Глубокое понимание работы компьютера и операционной системы.
2. Максимальная гибкость при работе с аппаратными ресурсами.
3. Оптимизация программ по скорости выполнения.
4. Оптимизация программ по размеру кода.
5. Возможность дизассемблирования и отладки.

Внедрение можно осуществлять как непосредственно кодом, так и динамической библиотекой (если код имеет большой объем). Внедрение

непосредственно кода занимает много времени и применимо для статических программ, т.е. для программ, исполняемые файлы которых находятся на компьютере хакера.

Внедрение динамической библиотеки можно обнаружить при помощи обозревателей процессов, например программы Process Explorer. В этом случае самый простой прием – применение имени своего файла, очень похожим на системную утилиту, например ntdl1.dll - в конце имени стоит цифра 1.

При некорректном внедрении кода могут произойти проблемы и ошибки в приложениях. При внедрении в браузер Chrome он на 15% чаще сталкиваются с сбоями, поэтому Google работает над блокировкой внедрений. Microsoft вообще заблокировал возможность внедрения в свой браузер Edge.

Внедрение кода не изменяет базовое приложение на диске. Программа для внедрения ожидает запуска приложения и вводит дополнительный код в этот запущенный процесс, чтобы изменить его работу.

Windows включает в себя интерфейс прикладного программирования (API), функции которого используют для внедрения кода. Процесс может присоединиться к целевому процессу, выделить память, записать DLL или другой код в эту память, а затем дать целевому процессу команду выполнить код. Windows не препятствует такому взаимодействию процессов на вашем компьютере друг с другом.

Перед внедряемым кодом стоят основные задачи:

– разместить свое тело внутри исследуемого файла;

– перехватить управление до начала выполнения основной программы;

– определить адреса API-функций, необходимые для собственного функционирования.

Осуществить внедрение можно в различные места файла:

– в заголовок PE-файла;

– в текущую секцию кода;

– в новую секцию кода;

– в секции данных;

– в расширение секций (кода или данных).

Внедрить код можно в любую секцию, но в этом случае надо принудительно поставить свойства (атрибуты) секции. А такие секции должны определяться антивирусами и защитником Windows как неестественные и следовательно – враждебные. Хотя так бывает далеко не всегда.

1. СОЗДАНИЕ ИКОНКИ

Иконки являются важной частью любой системы дизайна и способствует улучшению внешнего вида интерфейса окна, а также улучшают его идентификацию. В современных ОС иконка может иметь достаточно большие размеры (512 на 512 пикселей) и, практически, может заменить фотографию на форме окна. Из-за большого количества пикселей иконки стало возможным на них размещать различную текстовую информацию, например, свою фамилию, название учебной группы и др. В учебных целях размещение фамилии на иконке частично свидетельствует об авторстве программы (проекта).

Существуют два способа создания иконки:

– создание иконки с ”нуля” – как с использованием специальных программ рисования иконки, так и с использованием графических редакторов:

– с применением готового рисунка (фотографии).

Рассмотрим создание иконки из готового рисунка. Это самый быстрый способ создания иконки и алгоритм ее создания может быть следующим:

1. В Гугл-е осуществить поиск фотографии с цифровой тематикой, например, со словами <двоичный, бинарный код, бинарная система, HTTP, WWW> и скопировать в буфер.

2. Открыть презентацию. Выставить фон с самым слабым серым цветом (чтобы фон иконки не отличался от стандартного фона окна). Применить этот фон ко всем слайдам.

Вставить рисунок.

Вырезать квадратную область, т.к. иконка всегда создается в виде квадрата. Вставить выделенную область на новый слайд.

Нажать на вставленное изображение дважды. Откроется панель <Работа с рисунками/Формат>.

Применить одну из графических форм рамок (с закругленными рамками, с поворотом и др.).

Вставить текст (например, название кафедры, свою фамилию).

Для вставки текста необходимо выполнить: <Вставка / WordArt / Выбрать цвет>.

Ввести текст.

Выбрать: <Текстовые эффекты / Преобразовать / Искривление>.

Сохранить презентацию.

3. Скопировать изображение экрана с созданным рисунком и надписями. Открыть графический редактор, вставить изображение из буфера. Откорректировать изображение и сохранить как рисунок.

4. Открыть сайт создания иконок, например https://online-converting.ru.

Выбрать преобразование в файл ico.

Выставить все отметки качества иконки (если не выбрать высокое качество иконки, то иконка будет маленькая и иметь размытый вид).

Загрузить рисунок на сайт.

Скачать иконку.

Примеры вариантов иконок приведены на рис. 1.1.

Рис. 1.1. Варианты иконок

Вместо файла иконки нельзя использовать фотографию, т.к. иконка внутри себя содержит несколько иконок различного разрешения.

Иконку больших размеров можно применить только в полноценных Windows-окнах. Поэтому рассмотрим проект создания и вывода иконки размером 256 х 256 пикселей, например, в диалоговой программе.

Проект создания Windiws-окна состоит из этапов:

1. Создание при помощи программы ресурсов, например ResEd, программы ресурсов;

2. Изменение пакетного файла с именем makeit.bat для создания программы с ресурсами;

3. Написание исходного кода программы пользователя.

Файл ресурсов с иконкой размерностью 256 х 256 создан при помощи программы ResEd и представлен в программе 1.1.

Программа 1.1. Файл ресурсов:

```
#define IDD_DLG1 1000
#define IDC_IMG1 1001
#define IDC_GRP1 1002
#define IDC_BTN1 1003
#define IDC_BTN2 1004
#define IDC_STC1 1005
#include "C:/masm64/include64/Resource.h"
IDD_DLG1 DIALOGEX 10,10,300,171
CAPTION "Диалоговое окно"
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_CAPTION|WS_SYSMENU| WS_MINIMIZEBOX|
DS_CENTER
    BEGIN
CONTROL "",IDC_IMG1,"Static", WS_CHILDWINDOW|WS_VISIBLE|
SS_CENTERIMAGE|SS_ICON,6,6,174,162
      CONTROL "Управление программой",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,195,39,96,84
      CONTROL "Вычисление",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,63,78,15
      CONTROL "Выход",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,96,78,15
      CONTROL "http://blogs.kpi.kharkov.ua/v2/asm",IDC_STC1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,174,156,123,9
    END

100 ICON DISCARDABLE "icon1.ico"
101 ICON DISCARDABLE "icon2.ico"
102 ICON DISCARDABLE "icon3.ico"
```

Изменения в файле makeit.bat состоят в том, что в нем должна быть строка кода

\masm64\bin64\rc.exe %appname%.rc

Пример применения иконки размерностью 256 х 256 представлен в программе 1.2.

Программа 1.2. Диалоговая программа с иконкой размерностью 256 х 256 для вычисления уравнения 6cb/3; переменные заданы массивом:

```
        include \masm64\include64\masm64rt.inc ; библиотеки
.data?
         hInstance dq ?
         hIcon dq ?
.data ;
        mas   dq 2,1,3,6,3,16,2,8
                  ; a,b,c,d,3,16,2,8
        buff1 dq 8 dup(0),0; буфер
str1 db "a = 2, b = 1, c = 3, d = 6",10,10,"Уравнение: 6cb/3;" ,10,"Результат:
%d",10,10,  "Автор: http://blogs.kpi.kharkov.ua/v2/asm",0

                        comment * ----------------------------
                        Применения иконки размерностью 256 х 256
                        ---------------------------------------------*
.code ;
entry_point proc
     mov hInstance,rv(GetModuleHandle,0) ; дескриптор окна
 mov hIcon,rv(LoadImage,hInstance,100, IMAGE_ICON, 256,256,
LR_DEFAULTCOLOR)
         invoke DialogBoxParam,hInstance,1000,0,ADDR main,hIcon
         invoke ExitProcess,0
         ret
   entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
         .switch uMsg
          .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; уст. маленькую
иконку в строке названия окна
invoke SendMessage,rv(GetDlgItem,hWin,1001),\
      STM_SETIMAGE,IMAGE_ICON,lParam ; уст. иконку в клиентской обл.

         .case WM_COMMAND
          .switch wParam
```

```
          .case 1003        ; кнопка Вычисление       6cb/3
           mov rax,mas[8] ; b
           mul mas[16]  ; c
           mul mas[24]  ; d=6
           xor rdx,rdx    ; инициализация rdx (xor - обнуление)
           div mas[32]  ; 3
           mov r10,rax  ; результат
        invoke wsprintf,addr buff1,addr str1,r10
        invoke MsgboxI,0,addr buff1,"Вычисление",MB_OK,102

          .case 1004         ; кнопка Выход
invoke MsgboxI,0,"Вы подтверждаете выход из программы?", "Выход",
MB_YESNO,101
           cmp rax,6    ; YES=6
           je exit1     ; Переход, если равно
         .endsw

        .case WM_CLOSE
exit1: invoke EndDialog,hWin,0 ;
        .endsw
        xor rax, rax
        ret
       main endp
       end
```

Результат выполнения программы представлен на рис. 1.2.

В программе используются три иконки.

Первая иконка – иконка большого размера имеет идентификатор 100 и используется в начале программы при создании окна в строке кода
mov hIcon,rv(**LoadImage**,hInstance,**100**, IMAGE_ICON, 256,256, LR_DEFAULTCOLOR)

Эта же иконка, но маленького размера, устанавливается в строке заголовка окна в строке кода

invoke SendMessage,hWin,WM_SETICON,1,lParam

Вторая иконка с идентификатором 102 (рис. 1.3) используется при вызове макроса в строке кода

invoke MsgboxI,0,addr buff1,"Вычисление",MB_OK,**102,**

который основывается на использовании функции MessageBoxIndirect.

Рис. 1.2. Диалоговое окно программы

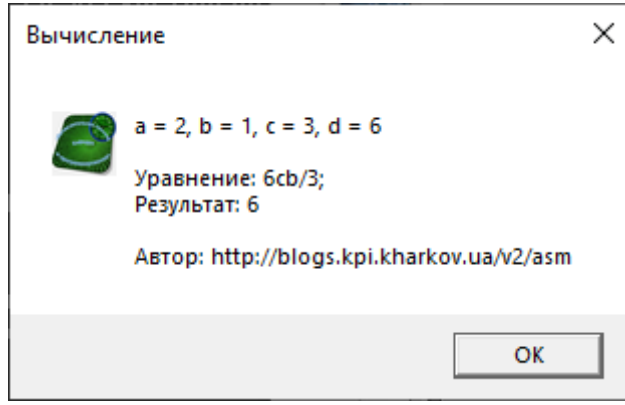


Рис. 1.3. Упрощенное окно с названием Вычисление

Третья иконка с идентификатором 101 (рис. 1.4) используется в строке кода

invoke Msgboxl,0,"Вы подтверждаете выход из программы?", "**Выход**", MB_YESNO,**101**

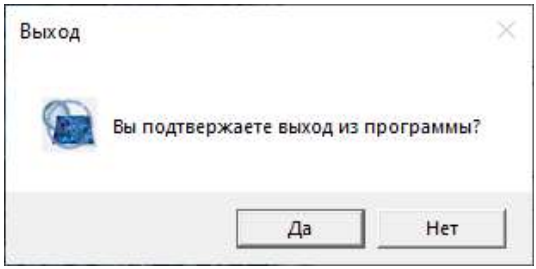


Рис. 1.4. Упрощенное окно с названием Выход

При необходимости отображения иконок больших размеров можно воспользоваться созданием полноценных диалоговых окон.

2. ИЗМЕНЕНИЕ ЯЗЫКА ИНТЕРФЕЙСА

Для изменения языка интерфейса программы можно использовать различные программы работы с ресурсами: Resourse Hacker, eXeScope, PE Explorer, QTranslate, Multilizer и др.

Программа Multilizer – мощная программа для перевода программ на русский язык. Имеет несколько импортеров, благодаря которым можно организовать автоперевод.

Существуют специально разработанная программы для перевода. Например, программа LikeRusXP. В ней есть встроенные переводчики и собственные глоссарии, которые можно дополнять.

В качестве примера изменения языка интерфейса воспользуемся программой Resourse Hacker. Загружаем (действия: File / Open) в Resourse Hacker программу, язык интерфейса которой будем менять. Например программу LordPE. Пример интерфейса программы приведен на рис. 2.1.

Рис. 2.1. Пример интерфейса программы

Последовательно открывая соответствующие папки ищем те окна, в которых планируем изменить интерфейс и в ручном режиме осуществляем перевод. Результат проведенных действий перевода можно посмотреть на рис. 2.2.

После внесения изменений нажимаем иконку треугольной формы и сохраняем файл.

Таким образом, результат выполненных действий представлен на рис. 2.3.

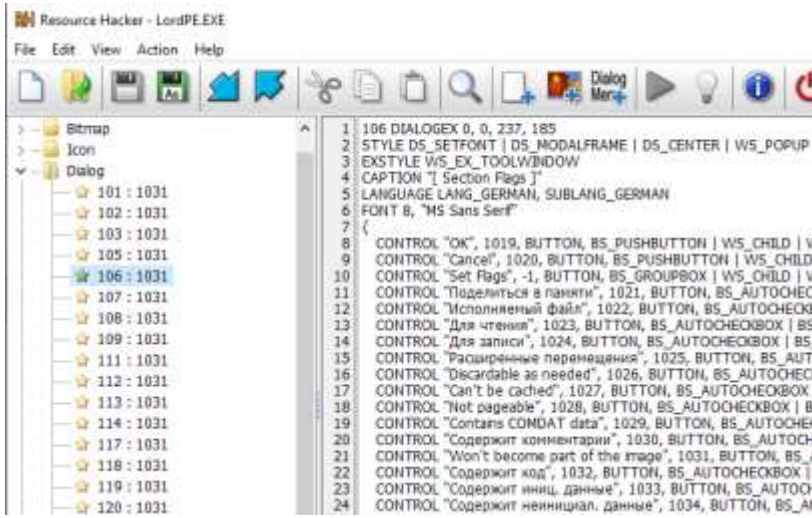


Рис. 2.2. Результат проведенных действий перевода

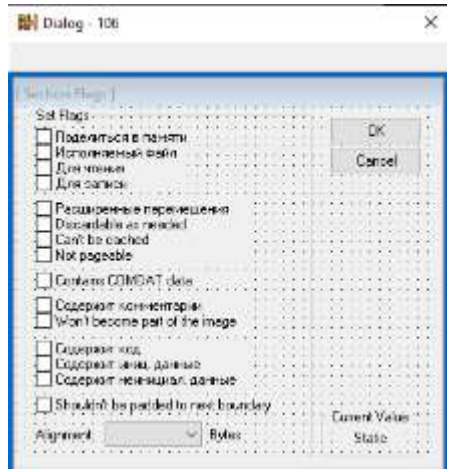


Рис. 2.3. Окно программы с измененным интерфейсом

Аналогичным образом можно поменять любые окна или всю программу.

3. ПЕРЕИМЕНОВАНИЕ ПРОГРАММНОГО СЕГМЕНТА

При написании программы на Ассемблере почти всегда используется секция данных (.data) и обязательно – секция, где размещается текст программы (.code). Секция программы code по умолчанию является обязательной для ассемблерной программы. Но в окнах отладчика x64Dbg секция code переименовывается на имя text.

Однако, программа может быть оформлена в виде секции с собственным именем и с определенными свойствами доступа. Применение секции с оригинальным именем придает программе только внешнее изящество, которое можно увидеть только при анализе этого exe-файла специальными программами. Кроме того, применение такой секции косвенно может свидетельствовать о способностях автора в оформлении своего кода.

Рассмотрим программу 3.1, которая выполняется в среде masm64 [4-8].

Программа 3.1.

```
include \masm64\include64\masm64rt.inc ; библиотеки
        comment * ----------------------------
Вывод смс в упрощенное окно функцией MessageBoxTimeout
        длительностью 5 секунд
        -------------------------------------------*
.data
    st1 db "Внимание! ",0
    titl db "Просто MessageBoxTimeout",0
mySg1 segment READ WRITE EXECUTE alias("mySeg")
;.code
    entry_point proc
invoke MessageBoxTimeout,0,addr st1,addr titl, MB_ICONINFORMATION,0,5000
    invoke ExitProcess,0
    entry_point endp
mySg1 ends
    end
```

Программа с собственным именем секции текста, естественно, не имеет строки с именем code. Вместо этой строки применена конструкция, оформленная ввиде сегмента:

```
mySg1 segment READ WRITE EXECUTE alias("mySeg")
…
```

```
mySg1 ends
```

Теперь секция текста программы имеет собственное имя секции mySeg (рис. 3.1).

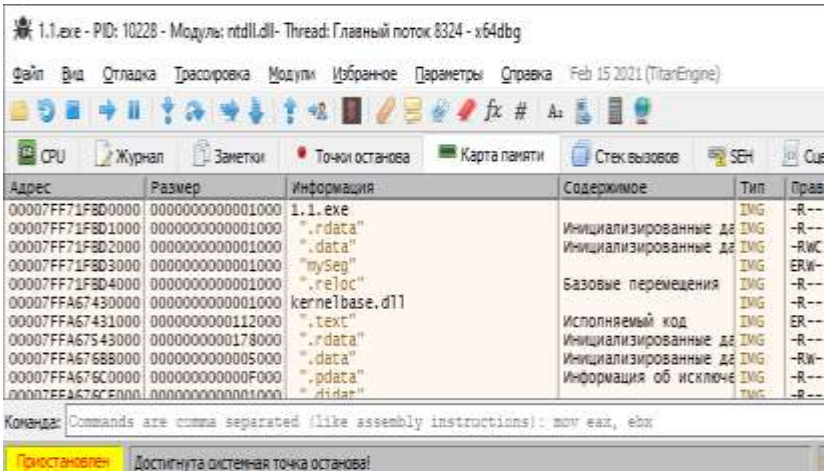


Рис. 3.1. Секции программы

Посмотреть характеристики секций можно при выборе в отладчике x64Dbg кнопки меню Карта памяти.

4. PE-ФОРМАТ

4.1. Общие сведения о PE-формате

Работа приложения начинается с того, что операционная система создает процесс.

Процесс предполагает создание множества внутренних системных структур для обеспечения работы программы и предоставления ей различных ресурсов: памяти, процессорного времени, доступа к установленному в системе оборудованию и т.д. [4, 8].

Каждый процесс получает в свое распоряжение виртуальное адресное пространство памяти (для x64 это 8192 Гб (8 Тб)). А это значит, что он может обращаться по любому из адресов памяти.

В адресном пространстве процесса резервируются области для динамически выделяемой памяти ("кучи") и стека. Затем образ программы загружается из файла на диск по базовому адресу загрузки. Образ программы состоит из одной или нескольких секций. Для каждой секции выделяется несколько страниц памяти, имеющих одинаковые атрибуты.

Исполняемый файл с расширением exe остался в наследство от DOS (executable) файлов. В Windows системах исполняемый файл называется Portable Executable (PE, «переносимый исполняемый») – формат исполняемых файлов, объектного кода и динамических библиотек 32- и 64-разрядных версий ОС. **PE-формат создается при компиляции программы** и представляет собой **структуры данных**, которые содержат всю информацию, необходимую PE-загрузчику для отображения файла в память. **Под PE-формат выделяется одна 4К-страница виртуальной памяти размером 0x1000 байт.**

В документации Microsoft приводилась схема PE-файла (рис. 4.1). В более упрощенном виде компоновку PE-файла можно представить в виде **заголовка и секций.**

Все PE-файлы начинаются с **ДОС MZ-заголовка**, который располагается в самом начале файла. В ДОС заголовке есть DOS stub заглушка – это модифицированный DOS-заголовок, который представляет собой небольшую программу DOS, выводящую простое текстовое сообщение наподобие "This program cannot be run in DOS mode". Это сделано для того, чтобы при ошибочной попытке запуска программы Windows под DOS она могла сообщить об ошибке. Модификация заголовка DOS заключается в том, что по смещению 3Ch от начала файла расположено 32-разрядное смещение **PE-заголовка**, в котором записаны буквы PE (что является признаком PE-формата. Программисты называют это сигнатурой – что глубоко не научно).

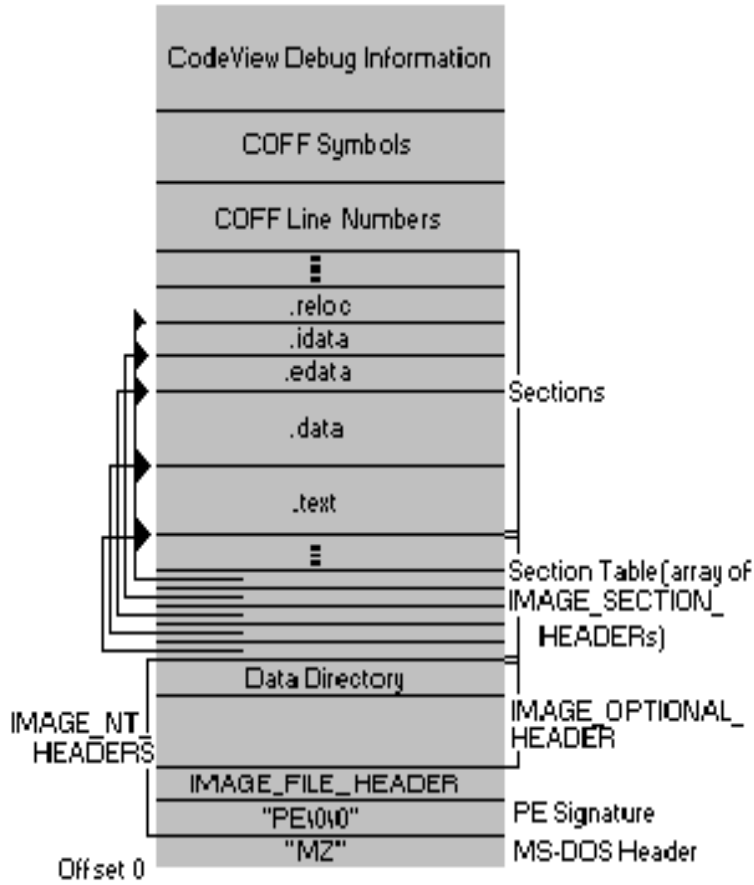


Рис. 4.1. Схема PE-файла от Microsoft

Т.о., формат PE-файла состоит из частей:

– структура IMAGE_DOS_HEADER – заголовок DOS;

– заглушки DOS;

– структура IMAGE_NT_HEADER – заголовок NT, одним из параметров которой есть структура IMAGE_OPTIONAL_HEADER;

– структура IMAGE_SECTION_HEADER, которая содержит столько же последовательных структур IMAGE_SECTION_HEADER, сколько существует разделов;

– разделы.

Рассмотрим основные шаги, выполняющиеся при загрузке PE-файла в память:

– когда PE-файл запускается, PE-загрузчик проверяет DOS MZ-заголовок, для того, чтобы определить смещение PE-заголовка. Если оно найдено, то загрузчик переходит к PE-заголовку. Первыми байтами являются символы «MZ», которые называются сигнатурой (сокращение от имени Марк Збиновски, который являлся ведущим разработчиком MS-DOS и архитектором формата PE);

– PE-загрузчик проверяет, является ли PE-заголовок целым и не испорченным. Если это так, то он переходит к концу PE-заголовка;

– за ним незамедлительно следует таблица секций. PE-загрузчик считывает информацию о секциях и загружает эти секции в память. Также он устанавливает каждой секции атрибуты, указанные в таблице секций;

– после того, как PE-файл загружен в память PE-загрузчик обрабатывает логические части PE-файла, например таблицу импорта.

ОС разрешает обращение к ячейкам памяти заголовка не на прямую, а через большое количество различных структур.

4.2. DOS MZ-заголовок

DOS MZ-заголовок начинается с букв "MZ" и буквы "MZ" – это сигнатура, которая описана в структуре **IMAGE_DOS_HEADER** и располагающейся в самом начале PE-файла.

В файле winnt.h располагается структура IMAGE_DOS_HEADER:

```
typedef struct _IMAGE_DOS_HEADER {      // DOS .EXE header
0x00 WORD   e_magic; // Сигнатура заголовка “MZ”
0x02 WORD   e_cblp;    // количество байт на последней странице файла
0x04 WORD   e_cp;      // количество секций в файле
0x06 WORD   e_crlc;    // Relocations
0x08 WORD   e_cparhdr;   // размер заголовка в параграфах
0x0a WORD   e_minalloc;  // минимальные дополнительные параграфы
0x0c WORD   e_maxalloc; // максимальные дополнительные параграфы
0x0e WORD   e_ss;   // начальное  относительное значение регистра SS
0x10 WORD   e_sp;      // значение регистра SP
0x12 WORD   e_csum;     // контрольная сумма
0x14 WORD   e_ip;          // начальное значение регистра IP
0x16 WORD   e_cs;  // начальное относительное значение регистра CS
0x18 WORD   e_lfarlc;   // адрес в файле на таблицу переадресации
0x1a WORD   e_ovno;  // количество оверлеев – части программы,
которые могут динам. накладываться на обл. памяти
0x1c WORD   e_res[4];         // Reserved words
```

0x24 WORD e_oemid; // OEM идентификатор
0x26 WORD e_oeminfo; // OEM информация; e_oemid specific
0x28 WORD e_res2[10]; // Reserved words
0x3c DD LONG **e_lfanew**; // **адрес в файле нового exe заголовка PE**
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;

В этой структуре **IMAGE_DOS_HEADER** (http://cs.usu.edu.ru/docs/pe/header.html) наиболее важны два поля: **e_magic** – это поле размером два байта (dw) и содержит сигнатуру 'MZ'; **e_lfanew** – указатель (размер dd) на начало PE-заголовка. Это поле указывает на первый байт PE-заголовка (структура **IMAGE_NT_HEADERS**), т.е. на сигнатуру "PE\0\0".

PE-формат имеет много плохо документированных полей. И одной программой-анализатором не всегда удобно просматривать поля структур заголовка. В программе-анализаторе CFF Explorer (иногда ее называют «швейцарским ножом») более полно раскрываются значения полей структур PE-формата и их содержимое (рис. 4.2).

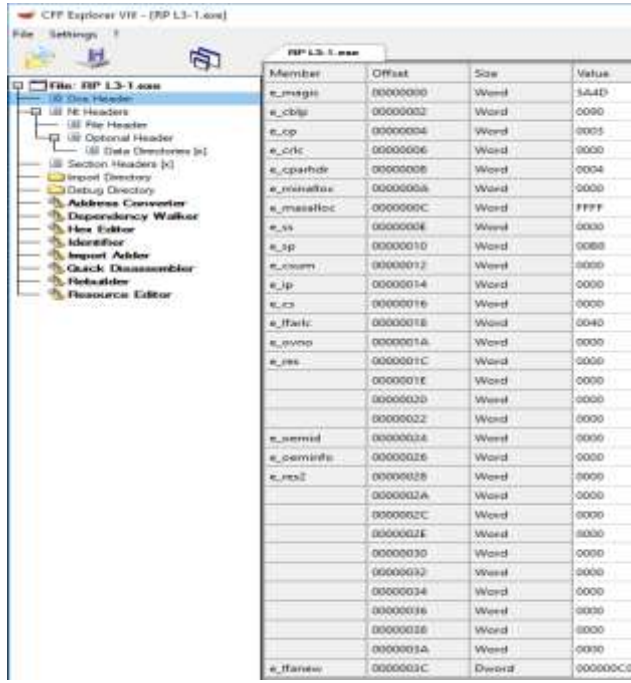


Рис. 4.2. Окно программы CFF Explorer с содержимым структуры

4.3. PE-заголовок

4.3.1. Общие сведения о PE-заголовке

Заголовок PE (структура IMAGE_NT_HEADERS) всегда выравнен на границу 8 байт и предназначен для загрузки файла в ОС Windows.

PE-заголовок – это общее название структуры под названием **IMAGE_NT_HEADERS**. Если программа запускается операционной системой, которая знает о PE-формате, то PE-загрузчик может найти смещение PE-заголовка в заголовке DOS-MZ.

Поля PE-заголовка приведены в табл. 4.1.

Таблица 4.1

Смещение	**Размер, байт**	**Поле**	**Типовое значение**
0	**4**	**Сигнатура 'PE'**	**50h 45h 00 00**
4	**2**	**Тип процессора**	**14Ch**
6	**2**	**Число секций в образе программы**	**-**
8	4	Время/дата создания файла	-
0Ch	4	Указатель на таблицу символов	0
10h	4	Количество отладочных символов	0
14h	**2**	**Размер дополнительного заголовка**	**E0h**
16h	**2**	**Тип файла**	**10Fh**
18h	**2**	**"Магическое" значение**	**10Bh**
1Ah	1	Старшая версия компоновщика	-
1Bh	1	Младшая версия компоновщика	-
1Ch	4	Размер кода	-
20h	4	Размер инициализированных данных	-
24h	4	Размер неинициализированных данных	-
28h	**4**	**Смещение точки входа**	**-**
2Ch	4	Смещение секции кода в памяти	-
30h	4	Смещение секции данных в памяти	-
34h	**4**	**Адрес загрузки образа в память**	**400000h**
38h	**4**	**Выравнивание секций в памяти**	**1000h**
3Ch	**4**	**Выравнивание в файле**	**200h**
40h	**2**	**Старшая версия Windows**	**4**
42h	2	Младшая версия Windows	0
44h	2	Старшая версия образа	-
46h	2	Младшая версия образа	-
48h	**2**	**Старшая версия подсистемы**	**4**
4Ah	2	Младшая версия подсистемы	0
4Ch	4	Зарезервировано	0

50h	**4**	**Размер загруженного файла в памяти**	**-**
54h	**4**	**Размер всех заголовков в файле**	**-**
58h	4	Контрольная сумма	0
5Ch	**2**	**Подсистема**	**2 или 3**
5Eh	2	Флаги dll	0
60h	4	Зарезервированный размер стека	100000h
64h	4	Выделенный размер стека	1000h
68h	4	Зарезервированный размер кучи	100000h
6Ch	4	Выделенный размер кучи	1000h
70h	4	Устарело	0
74h	**4**	**Число элементов в каталоге смещений**	**10h**

В документации на PE-файл используются три типа адресов: **базовый** Base, **виртуальный** VA и **относительный** RVA (Relative-Virtual-Address).

Виртуальный адрес получается из комбинации: VA = Base+RVA.

RVA – это смещение данного элемента по отношению к адресу, с которого начинается отображение PE-файла в памяти.

Базовый адрес можно получить при помощи блока кода:

```
invoke  GetModuleHandle,"kernel32.dll",0
mov r14,rax
```

Синтексис структуры **IMAGE_NT_HEADERS**:

```
IMAGE_NT_HEADERS STRUCT
  Signature       dd ?
  FileHeader      IMAGE_FILE_HEADER   <>
  OptionalHeader  IMAGE_OPTIONAL_HEADER32   <>
IMAGE_NT_HEADERS ENDS
```

Рассмотрим основные параметры структуры.

Signature – это 4-байтовая сигнатура "PE\0\0" (IMAGE_NT_SIGNATURE) с содержимым 50h, 45h, 00h, 00h. Фактически, это текст "PE", за которым следуют два нуля. Это поле, которое всегда и проверяют.

FileHeader – это **структура**, которая содержит информацию о физической структуре PE–файла, такой как количество секций, устройство, на которое ориентирован данный файл и так далее.

OptionalHeader – это **структура**, которая содержит информацию о логической структуре PE-файла. Необязательный заголовок имеет два формата: для PE32 и для PE32+. Соответствующие структуры называются IMAGE_OPTIONAL_HEADER32 и IMAGE_OPTIONAL_HEADER64.

Если значение сигнатуры в IMAGE_NT_HEADERS равно "PE", за которым следуют два нуля, тогда файла является PE. Фактически, специально для подобных сравнений Microsoft определила константу под название IMAGE_NT_SIGNATURE, которую можно использовать:

```
IMAGE_DOS_SIGNATURE equ  5A4Dh     ; MZ
IMAGE_OS2_SIGNATURE equ  454Eh     ; NE
IMAGE_OS2_SIGNATURE_LE equ 454Ch ; LE
IMAGE_VXD_SIGNATURE equ 454Ch      ; LE
IMAGE_NT_SIGNATURE equ 4550h       ; PE
```

В файле winnt.h так же располагается структура **_IMAGE_NT_HEADERS64**:

```
typedef struct _IMAGE_NT_HEADERS64 {
   DWORD Signature;
   IMAGE_FILE_HEADER FileHeader;
   IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;
```

Одним из значений структуры _IMAGE_NT_HEADERS64 есть структура IMAGE_FILE_HEADER, которая имеет синтаксис:

```
typedef struct _IMAGE_FILE_HEADER {
  WORD  Machine;
  WORD  NumberOfSections;
  DWORD TimeDateStamp;
  DWORD PointerToSymbolTable;
  DWORD NumberOfSymbols;
  WORD  SizeOfOptionalHeader;
  WORD  Characteristics;
} IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER
```

Сокращенные значения полей структуры **IMAGE_FILE_HEADER** приведены в табл. 4.2.

Таблица 4.2.

Смещение (hex)	Размер	Название	Описание
00	2	Machine	Обозначение процессора
02	2	NumberOfSections	Количество секций в файле
04	4	TimeDateStamp	Дата и время создания файла
08	4	PointerToSymbolTable	Смещение до таблицы символов или 0
0C	4	NumberOfSymbols	Количество элементов в таблице символов
10	2	SizeOfOptionalHeader	Размер необязательного заголовка
12	2	Characteristics	Атрибуты файла

Опциональный заголовок PE-файла

После этих структур располагается **опциональный заголовок PE-файла (**IMAGE_OPTIONAL_HEADER)

```
typedef struct _IMAGE_OPTIONAL_HEADER {
  WORD            Magic;
  BYTE            MajorLinkerVersion;
  BYTE            MinorLinkerVersion;
  DWORD           SizeOfCode;
  DWORD           SizeOfInitializedData;
  DWORD           izeOfUninitializedData;
  DWORD           AddressOfEntryPoint;
  DWORD           BaseOfCode;
  DWORD           BaseOfData;
  DWORD           ImageBase;
  DWORD           SectionAlignment;
  DWORD           FileAlignment;
  WORD             MajorOperatingSystemVersion;
  WORD             MinorOperatingSystemVersion;
  WORD             MajorImageVersion;
  WORD             MinorImageVersion;
  WORD             MajorSubsystemVersion;
  WORD             MinorSubsystemVersion;
  DWORD             Win32VersionValue;
  DWORD             SizeOfImage;
  DWORD             SizeOfHeaders;
  DWORD             CheckSum;
  WORD             Subsystem;
  WORD             DllCharacteristics;
  DWORD             SizeOfStackReserve;
  DWORD             SizeOfStackCommit;
  DWORD             SizeOfHeapReserve;
  DWORD             SizeOfHeapCommit;
  DWORD             LoaderFlags;
  DWORD             NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY
DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;
```

Самые важные поля опционального заголовка PE-файла следующие:

– Magic – для 32-разрядных PE-файлов это поле должно содержит значение 0x10B, а для 64-разрядных – 0x20B.

– AddressOfEntryPoint – адрес точки входа относительно базового адреса загрузки файла (ImageBase).

– ImageBase – базовый адрес загрузки PE-файла. В памяти по этому адресу после загрузки располагается структура IMAGE_DOS_HEADER. Если у файла имеется таблица перемещаемых элементов, то этот адрес может варьироваться, а ImageBase будет содержать лишь рекомендуемый адрес загрузки.

– FileAlignment и SectionAlignment – файловое и виртуальное выравнивание секций с условием:

1. SectionAlignment >= 0x1000;
2. FileAlignment >= 0x200;
3. SectionAlignment >= FileAlignment.

– SizeOfImage – поле, которое содержит размер в байтах загруженного образа PE-файла, который должен быть равен виртуальному адресу последней секции плюс ее виртуальный выровненный размер.

– SizeOfHeaders – размер всех заголовков. Значение поля не должно превышать относительного виртуального адреса первой секции.

– CheckSum – контрольная сумма файла, которая проверяется загрузчиком только для самых важных системных файлов.

– Subsystem – подсистема файла.

– SizeOfStackReserve и SizeOfStackCommit, SizeOfHeapReserve и SizeOfHeapCommit – размер соответственно стека и кучи, которые должны быть зарезервированы и выделены для PE-файла; 0 – значение по умолчанию. Если SizeOfStackCommit > SizeOfStackReserve или SizeOfHeapCommit > SizeOfHeapReserve, то файл загружен не будет.

– NumberOfRvaAndSizes – количество элементов в таблице DATA_DIRECTORY, расположенной в самом конце опционального заголовка. Может варьироваться от 0 до 16

– **DataDirectory** – указатель на первую структуру IMAGE_DATA_DIRECTORY в каталоге данных.

4.3.2. Директории данных *DataDirectory*

Последний член опционального заголовка – это директория данных. Применение этой структуры имеет основополагающее значение при организации процесса ручного внедрения, например, динамической библиотеки в exe-файл.

Директория данных – это массив описателей, каждый из которых имеет такой вид:

```
struct IMAGE_DATA_DIRECTORY {
  DWORD VirtualAddress;
  DWORD Size;
};
```

Поля структуры:

VirtualAddress – это относительный виртуальный адрес (RVA) структуры данных;

Size – размер таблицы в байтах.

Этот массив часто используют для импортирования функций, когда загрузчик заполняет соответствующую таблице функций таблицу адресов всех необходимых библиотек. По сути, это вызов функции LoadLibrary для каждой требуемой библиотеки и затем вызовы GetProcAddress для каждой импортируемой функции.

Таблица импорта

Информация об импортируемых функциях, располагаетсяв трех структурах:

1. Каталоге импорта (Import Directory Table);
2. Таблице просмотра импорта (Import LookUp Table);
3. Таблице адресов импорта (Import Address Table, сокращенно IAT).

Import Address Table – таблица адресов импорта и состоит из массива 20-байтовых элементов типа IMAGE_IMPORT_DESCRIPTOR. Каждый элемент IMAGE_IMPORT_DESCRIPTOR соответствует одной из DLL.

Адрес таблицы импорта содержится в поле VirtualAddress второго члена директории данных. Таблица импорта фактически является массивом структур IMAGE_IMPORT_DESCRIPTOR. Каждая струтура содержит информацию о DLL, откуда PE импортирует функции. Например, если PE имортирует функции из 10 разных DLL, этот массив будет состоять из 10 элементов. Конец массива отмечается элементом, все поля которого содержат нули.

Синтаксис структуры:

```
OriginalFirstThunk  DD   ? ; RVA на структуру _IMAGE_IMPORT_BY_NAME.
                                                    ; Может не использоваться
TimeDateStamp     DD  0
ForwarderChaine   DD  0
ModName  DD  ? ; RVA строки символов ASCII, оканчивающейся 0
```

```
FirstThunk  DD  ? ; RVA Import Address Table
    IMAGE_IMPORT_DESCRIPTOR ends
```

Сам массив заканчивается структурой IMAGE_IMPORT_DESCRIPTOR, все поля которой равны нулю. Из всех полей этой структуры интерес представляют два:

Name – указывает на строку с именем библиотеки;

FirstThunk – указывает на массив структур IMAGE_THUNK_DATA32.

Остальные поля могут быть пустыми.

При запуске программы загрузчик ОС получает адреса используемых в программе функций и записывает их в массив адресов имен функций, на который ссылаются поля FirstThunk, и в котором до этого были RVA, ссылающиеся на имена соответствующих функций. После того, как этот массив заполнен адресами функций, он называется таблицей адресов импорта (Import Address Table, сокращенно IAT).

Каталог импорта и таблица просмотра импорта создаются при компиляции программы, а таблица адресов импорта, на основании данных в каталоге импорта и таблице просмотра импорта, заполняется загрузчиком перед запуском программы и до заполнения адресами функций содержит RVA имен этих функций в таблице просмотра импорта.

Когда PE-файл загружается в память, и загрузчик ОС заполняет IAT (где до этого были RVA имен функций) адресами соответствующих функций, то если потребуется найти имена функций, то их можно будет найти с помощью полей OriginalFirstThunk, которые указывают на массив, в котором хранятся RVA этих имен функций. Т.е. OriginalFirstThunk, в отличие от FirstThunk, указывает на массив адресов имен функций, который остается нетронутым (т.е. оригинальным).

Рассмотрим пример **переноса** таблицы IMAGE_IMPORT_DESCRIPTOR в свободные области памяти.

Для того чтобы в последствии внедрить DLL необходимо добавить в таблицу импорта соответствующую запись и при этом не испортить размещенные за этой записью другие поля PE-заголовка.

Таблица (структура) Import Directory Table является массивом подчиненных структур типа IMAGE_IMPORT_DESCRIPTOR, каждая из которых содержит RVA-указатель на имя загружаемой динамической библиотеки и ссылки на другие поля.

Т.о., чтобы внедрить DLL-код в таблицу **Import Directory Table** необходимо добавить в нее впереди еще один экземпляр структуры

IMAGE_IMPORT_DESCRIPTOR (5 двойных слов). А так как количество элементов в массиве нигде не указано, то последний элемент массива IMAGE_IMPORT_DESCRIPTOR должен иметь поля, равные NULL (5 двойных слов). После таблицы Import Directory Table начинается таблица **Import Address Table (IAT)** первой динамической библиотеки. Но место для внедрения отсутствует. Поэтому надо перенести эти две таблицы на свободное место.

Таблица Import Address Table – таблица адресов импорта и состоит из массива 20-байтовых элементов типа IMAGE_IMPORT _DESCRIPTOR.

Для осуществления внедрения необходимы две программы: программа, куда будет осуществляться внедрение DLL и непосредственно динамическая библиотека (программа с расширением dll).

Для внедрения используем самую простую программу (программа 4.1) с именем 1.asm, которая выводит сообщение при помощи API-функции MessageBox.

Программа 4.1. Файл с именем 1.asm:

```
    include \masm64\include64\masm64rt.inc
  .data
    titl db "Просто MessageBox",0
inf1 db "Основная программа, в которую будет осуществляться
внедрение DLL.",0
.code
entry_point proc
    invoke MessageBox,0,addr inf1,addr titl,MB_ICONINFORMATION
    invoke ExitProcess,0
    entry_point endp
    end
```

Рассмотрим **процесс переноса** таблицы Import Directory Table при помощи программы **Hiew 8.43**. Программа Hiew – это программа для просмотра и редактирования файлов в шестнадцатеричном виде. Программа Hiew работает с образом файла **на диске**. Поэтому адреса структур и параметров не совпадают с адресами **в памяти**, с которыми работают отладчик и другие программы-анализаторы.

Многие поля в PE-файлах заполнены RVA (Relative Virtual Address – относительный виртуальный адрес). RVA – это смещение данного элемента по отношению к адресу, с которого начинается отображение PE-файла в памяти.

Таблица импорта представляет из себя сооружение иерархического типа. Вершиной таблицы является структура Import Directory Table, фактически являющаяся массивом подчиненных структур типа IMAGE_IMPORT_DESCRIPTOR, каждая из которых содержит RVA-указатель на имя загружаемой динамической библиотеки.

Т.о. кроме создания новой записи в таблице Import Directory Table необходимо также создать таблицу импортируемых адресов Import Address Table (IAT), импортирующую имя (DWORD Name функции, а также RVA таблицы IAT.

В запущенном редакторе Hiew открываем exe-файл (1.exe), в который будет осуществляться внедрение (а пока только перенос самой таблицы). Перемещения в редакторе осуществляется при помощи стрелок на клавиатуре.

После загрузки exe-файла для просмотра 16-ричного значения выбираем кнопку <F4> (Mode), а затем <Hex> и подтверждаем нажатием кнопки <Enter>.

Для просмотра параметров заголовка выбираем <F8> (Header), а затем <F10> (Dir – отобразить список папок и файлов). Список каталогов, который отобразился представлен на рис. 4.3.

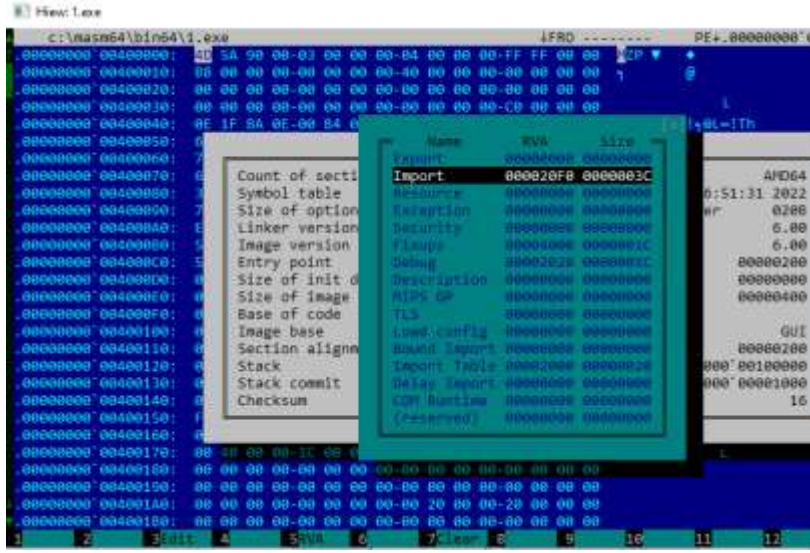


Рис. 4.3. Список папок и файлов заголовка в программе Hiew 8.43

Выбираем строку Import и нажимаем Enter.

Теперь необходимо **скопировать** оригинальную таблицу Import Directory Table размерностью 3Ch на новое место; впереди оставить место для внедряемой dll размером 14h (новый элемент структуры типа IMAGE_IMPORT_DESCRIPTOR), для того чтобы этот элемент структуры проинициализировался первым.

Для копирования таблицы Import Directory Table необходимо:

– выделить 3Ch байт. Для этого, подвести стрелками перемещения курсор на адрес начала таблицы – адрес 20F0h;

– нажать на клавишу <*>;

– переместить курсор вправо на 3Ch байт;

– еще раз нажать на клавишу <*>;

– переместить курсор в начало выделенного блока;

– нажать <Ctrl-F2> (Get Block), ввести имя файла, например old, затем Enter, затем в строку offset ввести размер этой таблицы, равный 3Ch (рис. 4.4); нажать Enter и в новом окне с именем Select table еще раз нажать Enter.

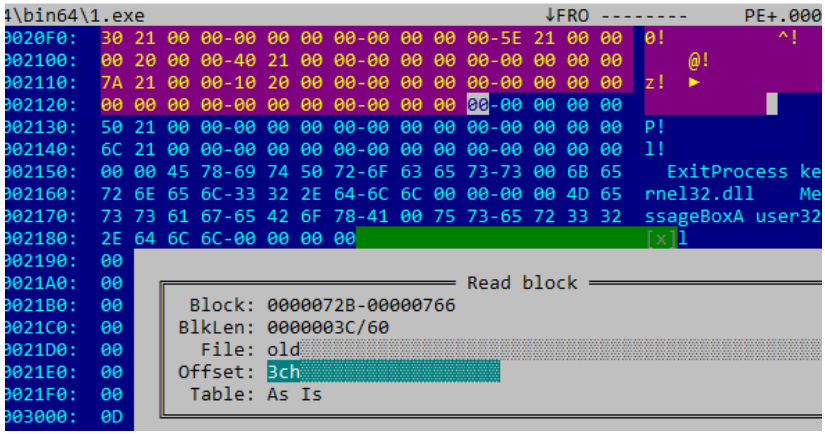


Рис. 4.4. Выделение таблицы Import Directory Table

Но перед сохранением таблицы на новом месте впереди необходимо зарезервировать 14h=20d байтов для нового элемента структуры типа IMAGE_IMPORT_DESCRIPTOR. Но удобнее

оставить две строки байтов (не 14h, а 20h). В этом случае таблица будет размещаться с начала строки адресов, т.е. с адреса 30D0h.

Для **сохранения** выделенного блока на свободном месте предыдущие действия **повторяются**:

– перемещаем стрелочками курсор на свободные ячейки;

– в свободной части памяти оставляем две строки байтов для нового элемента структуры типа IMAGE_IMPORT_DESCRIPTOR;

– нажимаем <Ctrl-F2> (Get Block), вводим имя файла (имя old), затем Enter, затем в строку offset вводим размер этой таблицы, равный 3Ch (рис. 4.5); нажимаем Enter и в новом окне с именем Select table еще раз нажимаем Enter.

Рис. 4.5. Результат копирования таблицы Import Directory Table на свободные ячейки

После этих действий таблица Import Directory Table скопирована, начиная с адреса 30D0h и размером 3Ch. Обратим внимание, что следующая свободная ячейка имеет адрес 310Ch.

Теперь **меняем адрес вызова** таблицы Import Directory Table. Для этого выполняем действия:

– для просмотра параметров заголовка выбираем <F8> (Header), а затем <F10>;

– подводим курсор к строке Import;

– нажимаем <F3>. Появляется окно с двумя строками (рис. 4.6);

– нажимаем еще раз <F3> и в окне со значением RVA и вводим все 8 цифр нового адреса – 000030D0h (рис. 4.7);

– нажимаем Enter. Значение RVA должно измениться;

– нажимаем <F9> для обновления значения;

– закрываем 16-ричный редактор Hiew для освобождения программы с именем 1.exe;

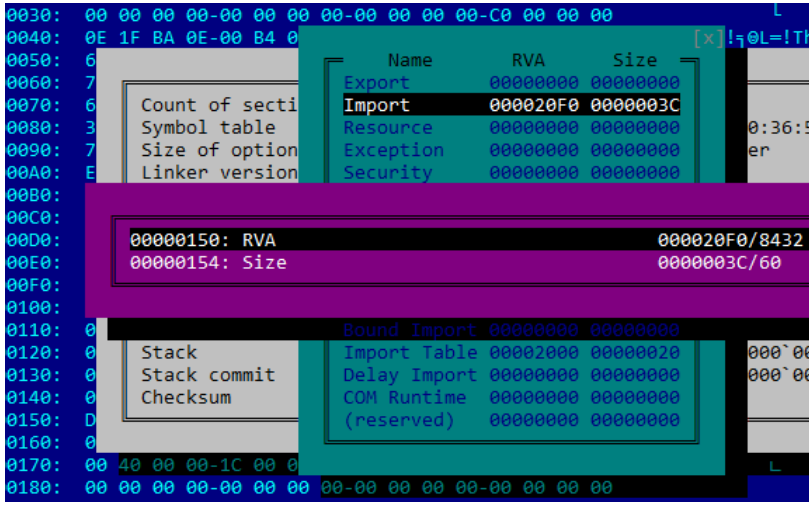


Рис. 4.6. Окно со значениями таблицы Import Directory Table

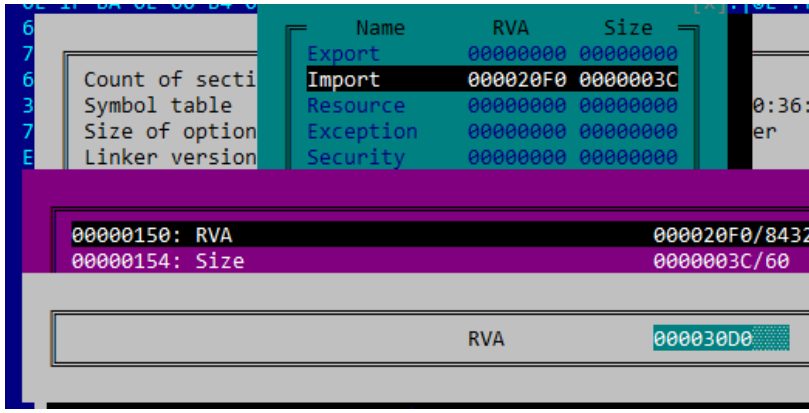


Рис. 4.7. Окно с изменением RVA таблицы Import Directory Table

– запускаем программу с именем 1.exe для проверки внесенных изменений. Отсутствием ошибок при переносе таблицы Import Directory Table является вызов функции MessageBox в программе с именем 1.exe.

Перенос таблицы Import Directory Table является **только первым** и самым простым этапом внедрения dll. Остальные самые ответственные этапы рассмотрены ниже в общем виде:

1 этап. В конце таблицы Import Directory Table в соответствии со структурой IMAGE_IMPORT_BY_NAME STRUCT необходимо запомнить имена внедряемой функции и внедряемого dll-файла.

Имя импортируемой функции в DLL указывается в структуре:

```
IMAGE_IMPORT_BY_NAME  STRUCT
  Hint  DW   ? ; индекс строки с именем импортируемой функции в DLL
  Name_ DB  ? DUP(?),0       ; название импортированной функции
     Pad  DB ($ and 1) DUP(0)  ; выравнивание до четной границы 0
IMAGE_IMPORT_BY_NAME ends
```

Для этого после таблицы Import Directory Table через 2 нулевых байта пишем имя функции, а следом за ним через пропуск одного байта – имя внедряемой динамической библиотеки.

2 этап. Необходимо заполнить поля нового элемента структуры размерностью 14h той dll, которую внедряем.

Синтаксис структуры:

```
OriginalFirstThunk  DD   ? ; RVA на структуру _IMAGE_IMPORT_BY_NAME.
                                                    ; Может не использоваться
TimeDateStamp     DD  0
ForwarderChaine   DD  0
ModName  DD  ? ; RVA строки символов ASCII, оканчивающейся 0
FirstThunk  DD  ? ; RVA Import Address Table
IMAGE_IMPORT_DESCRIPTOR ends
```

Для этого, перед таблицей Import Directory Table в оставленных 14h байтах заполняем новый элемент структуры Import Directory Table.

Первое двойное слово содержит OriginalFirstThunk и остается равным нулю.

Второе двойное слово содержит поле TimeDataStamp. В него заносится RVA-адрес импортируемой функции с учетом обратного порядка байт. Нажимаем <F3> и вписываем числа.

Третье двойное слово ForwarderChain остается равным нулю.

Четвертое двойное слово содержит поле Name. В него записывается RVA-адрес имени внедряемой динамической библиотеки. Нажимаем <F3> и вписываем числа.

Пятое двойное слово содержит поле FirstThunk, содержащее RVA-таблицы IAT, размещенной поверх поля TimeDateStamp. Нажимаем <F3> и вписываем числа.

Сохраняем внесенные изменения (нажимаем <F9>).

3 этап. С учетом заполнения нового элемента структуры типа IMAGE_IMPORT_DESCRIPTOR необходимо внести **изменения в адрес** нахождения этой таблицы и увеличение **размера этой таблицы** на 14h байтов.

Детальный пример внедрения dll рассмотрен в статье «Троянизация приложений – технологии XXI века», автора Крис Касперски.

4.3.3. Структура IMAGE_SECTION_HEADER

Последним элементом в PE-заголовке есть структура **IMAGE_SECTION_HEADER**, которая содержит различную информацию о секциях:

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE  Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
  } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
  WORD  NumberOfRelocations;
  WORD  NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

Назначение элементов этой структуры:

Name –поле, размером в 8 байт, содержит имя секции, в ASCII кодировке;

VirtualSize –четырёхбайтовое поле содержит размер (в байтах) секции (той самой комнаты) в виртуальной памяти;

VirtualAddress – четырёхбайтовое поле содержит относительный адрес секции в виртуальной памяти;

SizeOfRawData – четырёхбайтовое поле содержит размер секции в файле;

PointerToRawData – указатель на данные, которые содержатся в этом четырёхбайтовом поле;

Characteristics – четырёхбайтовое поле содержит атрибуты секции. Например, права чтения, записи и исполнения (Read Write Execute) (RWE).

4.4. Секции PE-файла

Содержимое PE-файла разделено на блоки, называемые **секциями**. Секция – это блок данных с общими атрибутами, такими как код/данные, чтение/запись и т.д.

Непосредственно за PE-заголовком следует таблица секций, представляющая собой массив структур.

Каждая структура содержит информацию о каждой секции в PE-файле:

– RVA – виртуальный адрес (смещение от начала всего блока памяти, выделенного загрузчиком для размещения DLL) той области памяти, которую загрузчик должен отвести для данной секции;

– VirtualSize – размер отводимой для секции области памяти (виртуальный размер секции);

– RawDataOffset – смещение (от начала файла) данных, которые следует поместить в область памяти, отводимую загрузчиком для данной секции при загрузке DLL;

– RawDataSize – размер переносимых в секцию данных;

– Protection – флаги защиты памяти, которые следует установить для данной секции после ее размещения в памяти.

Отличие в структурах PE32 и PE64 форматов заключается в том, что все виртуальные адреса (VA) – это 64-разрядные числа (8-байтовые). Все относительные адреса (RVA) так и остались 32-разрядными.

Часто встречающиеся секции

Секция **.text с**одержит программный код общего назначения.

Секция **.data** содержит инициализированные данные.

Секция **.bss** содержит неинициализированные данные.

Секция **.rsrc** содержит ресурсы модуля.

Секция **.rdata** содержит каталог отладки.

Секция **.idata** содержит информацию о функциях и данных, которые импортируются из других DLL.

Секция **.reloc** содержит таблицу базовых поправок (base relocation). Базовая поправка – это настройка по отношению к инструкции или значению инициализированной переменной. EXE- или DLL-файлы

используют поправку, если загрузчик не может загрузить файл по адресу, который предполагался компоновщиком. Если загрузчику удается загрузить отображение по указанному компоновщиком базовому адресу, загрузчик игнорирует поправочную информацию в этой секции.

Секция **.edata** представляет перечень функций и данных, которые PE-файл экспортирует для использования другими модулями.

Секция **.idata** содержит дескрипторы импорта.

Атрибуты секций

Каждая секция характеризуется признаками (свойства, флагами или характеристиками). Полный перечень флагов, имеющие название IMAGE_SCN_..._... приведен в файле winnt.h. Вот некоторые из них:

0x00000020 –секция содержит программный код;

x00000040 – секция содержит инициализированные данные;

0x00000080 – секция содержит неинициализированные данные;

0x00000200 – секция содержит комментарии или какой-нибудь другой вид информации. Типичное использование такой секции — это секция .drectve, создаваемая компилятором и содержащая команды для компоновщика;

0x00000800 – содержимое данной секции не должно быть помещено в конечный EXE-файл. Такая секция используется компилятором/ассемблером для передачи информации компоновщику;

0x02000000 – секцию можно отбросить, так как она не используется программой, после того как последняя загружена. Чаще всего встречается отбрасываемая секция — это секция базовых поправок (.reloc);

0x10000000 – секция является совместно используемой. При использовании с DLL данные в такой секции используются совместно всеми процессами, использующими эту DLL. По умолчанию секции данных не являются совместно используемыми, т.е. каждый процесс, использующий DLL, имеет свою собственную отдельную копию такой секции данных. Совместно используемая секция дает указание менеджеру памяти устанавливать отображение страниц для этой секции так, что все процессы, использующие DLL, ссылаются на одну и ту же физическую страницу в памяти. Чтобы сделать секцию совместно используемой, необходим атрибут SHARED во время компоновки;

0x20000000 – секция является исполняемой;

0x40000000 – секция предназначена для чтения;

0x80000000 – секция предназначена для записи.

Флаги могут быть использованы одновременно, если применять к ним побитовою операцию «или». Например, если нужно чтобы в секцию

можно было записывать, читать из нее, а также указывает, что секция содержит код, то итоговой значение поля Characteristics будет выглядеть следующим образом:

80000000h + 40000000h + 00000020h = A0000020h

Для того, чтобы посмотреть и отредактировать атрибуты секции необходимо выполнить последовательность действий:

1.Открыть полученный exe-файл в программе анализаторе PE-файла с названием LordPE, основное окно которого приведено на рис. 4.8.

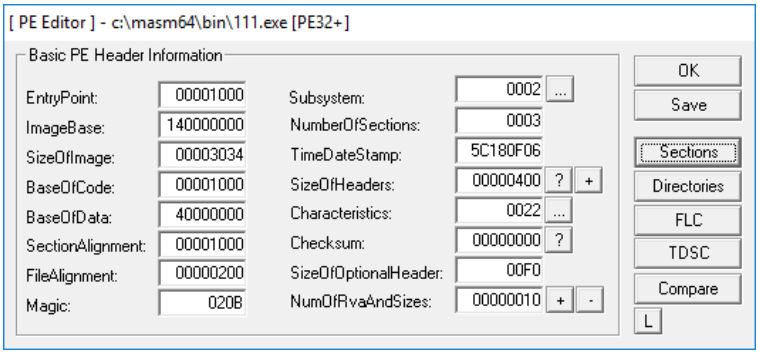


Рис. 4.8. Основное окно программы-анализатора LordPE

2. Выбрать пункт меню Section.

Секции программы и их характеристики представлены на рис. 4.9.

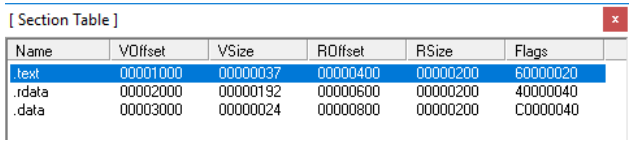


Рис. 4.9. Секции программы в анализаторе LordPE

3. Выделить секцию, нажать правую клавишу мышки и выбрать пункт edit section header (рис. 4.10).

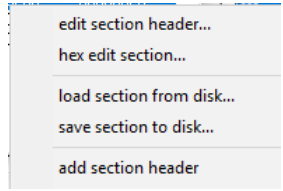


Рис. 4.10. Меню анализа секции в анализаторе LordPE

4. Выбрать пункт Flags (рис. 4.11).

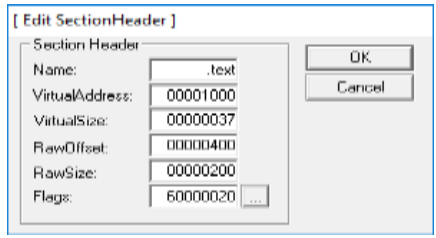


Рис. 4.11. Основные свойства секции в анализаторе LordPE

5. Отредактировать флаги секции (рис. 4.12).

Рис. 4.12. Окно редактирования флагов секции
в анализаторе LordPE

4.5. Пример исследования PE-файла

При исследовании заголовка PE-файла будет использована программа-анализатор с именем LordPE. Для этого рассмотрим программу 4.2, которая выводит сообщение в упрощенное окно в течении времени приблизительно равном 5 с.

Программа 4.2.

```
    include \masm64\include64\masm64rt.inc ; библиотеки
.data
    st1 db "Внимание! Окошко самоуничтожится через 5 с ",0
    titl db "Просто MessageBoxTimeout",0
.code
entry_point proc
invoke MessageBoxTimeout,0,addr st1,addr titl, MB_ICONINFORMATION,
0,5000
    invoke ExitProcess,0
entry_point endp
    end
```

Рассмотрим содержимое таблицы IAT (Import Address Table – таблица адресов импорта). В этой таблице находится информация, необходимая загрузчику для определения адреса импортируемых функций. Для этой цели будем использовать программу **Hiew 8.43**. Программа Hiew – это программа для просмотра и редактирования файлов в шестнадцатеричном виде. Программа Hiew работает с образом файла **на диске**. Поэтому адреса структур и параметров не совпадают с адресами **в памяти**, с которыми работают отладчик и другие программы-анализаторы. Перемещения по элементам меню программы происходит при помощи стрелок на клавиатуре.

Секция IAT (Import Address Table – таблица адресов импорта) состоит из массива 20-байтовых элементов типа IMAGE_IMPORT_DESCRIPTOR.

Для анализа таблицы IAT выполним действия:

– загружаем exe-файл в программу hiew32.exe;

– выбираем <F4< (mode) / Select mode→<Hex>;

– выбираем <F8> (Header) / <F10> (Dir);

– выбираем строку Import, обращаем внимание на адрес начала этой таблицы и значение ее размера, который равен 3Ch (рис. 4.13).

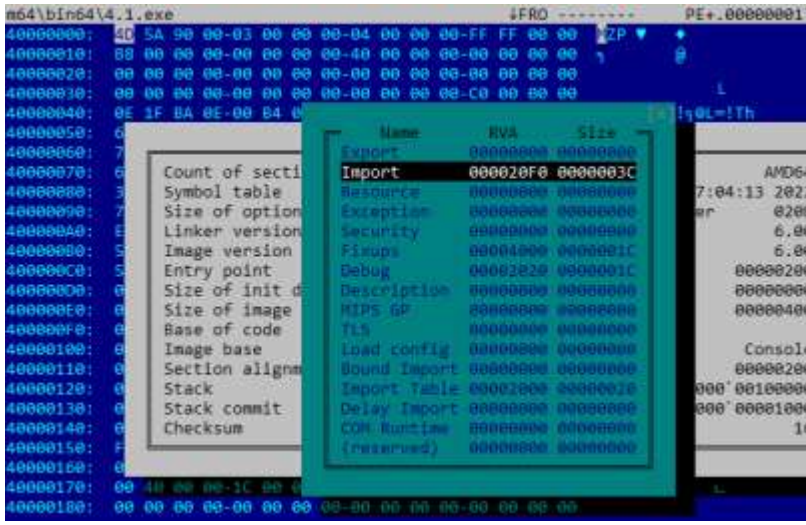


Рис. 4.13. Отображение параметров PE-заголовка в программе Hiew 8.43

При выборе последовательности <Esc> / <F7> (Import) отобразятся функции, которые использованы в программе и названия библиотек, где они расположены (рис. 4.14).

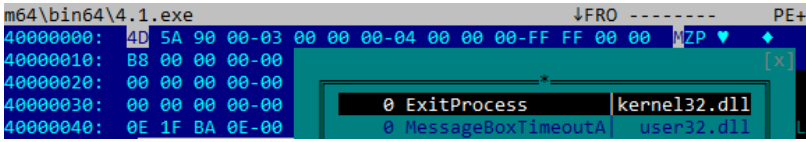


Рис. 4.14. Отображение имен функций и библиотек

Если подвести курсор на функцию MessageBoxTimeoutA и нажать на <Enter> то будет осуществлен переход к месту в программе, где происходит вызов этой функции (рис. 4.15).

А при выборе кнопки <F3> (Edit) в параметрах отображаются адреса ячеек памяти, где они располагаются (рис. 4.16).

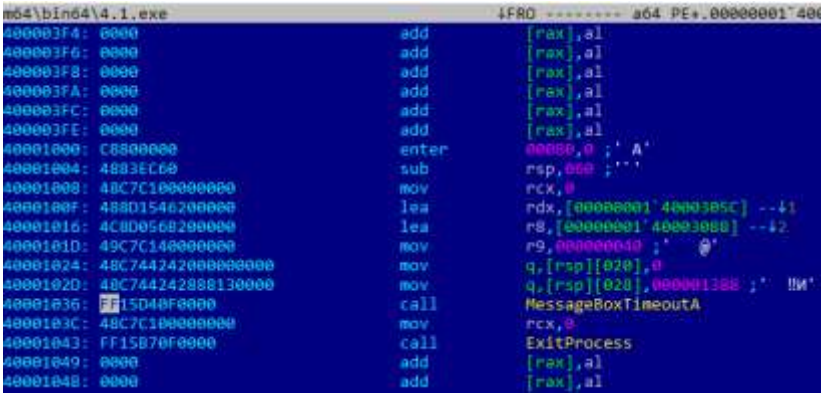


Рис. 4.15. Блок кода с отображением в параметрах номеров ячеек программы Hiew

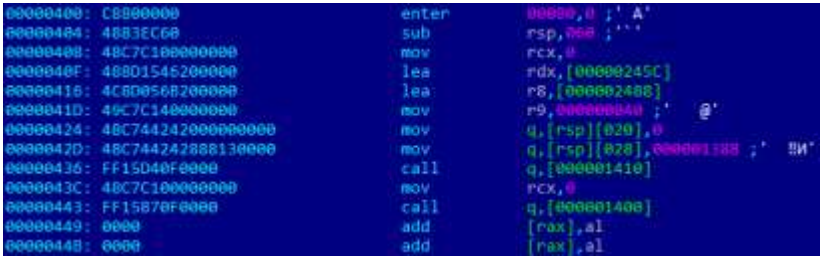


Рис. 4.16. Блок кода с отображением в параметрах ячеек памяти

С помощью утилиты (программы-анализатора PE-файлов) **LordPE** можно редактировать exe-файлы, анализировать их, оптимизировать, разбивать их на части и собирать обратно. Можно также дампить процесс частично или полностью, а также оптимизировать как дамп, так и сам файл после компилятора. Так же возможно редактировать информацию в заголовке файла (флаг SubSystem, корректировать контрольную сумму, редактировать характеристики и т.д.), а также просматривать и редактировать секции исполняемого файла.

Откроем полученный exe-файл в программе анализаторе PE-файла с названием **LordPE**.

Рассмотрим основные поля программы LordPE.

EntryPoint – AddressOfEntryPoint – адрес точки входа относительно

базового адреса загрузки файла.

ImageBase – базовый адрес загрузки PE-файла.

SizeOfImage – представляет общий размер всех частей отображения, находящихся под контролем загрузчика. Эта величина равна размеру области памяти, начиная с базового адреса отображения и заканчивая адресом конца последней секции. Адрес конца секции выровнен на ближайшую верхнюю границу секции.

BaseOfCode/BaseOfData – относительные базовые адреса кодовой секции и секции данных.

SizeOfImage – это поле содержит размер в байтах загруженного образа PE-файла.

FileAlignment и SectionAlignment – файловое и виртуальное выравнивание секций.

Magic – слово, определяющее состояние отображенного файла.

Subsystem – тип подсистемы, которую данный исполняемый файл использует для своего пользовательского интерфейса. Файл WINNT.H определяет следующие значения:

NATIVE == 1 – подсистема не требуется (например, для драйвера устройства);

WINDOWS_GUI = 2 – запускается в подсистеме Windows GUI;

WINDOWS_CUI = 3 – запускается в подсистеме Windows character (терминальное приложение);

OS2_CUI = 5 – запускается в подсистеме OS/2 (только приложения OS/2 1.x);

POSIX_CUI = 7 – запускается в подсистеме Posix.

NumberOfSections – количество секций в EXE- или OBJ-файле.

TimeDateStamp – временная отметка.

SizeOfHeaders – суммарный размер всех заголовков, сообщающий загрузчику, сколько байт читать от начала файла.

Characteristics – атрибуты доступа к секции и особенности ее загрузки.

CheckSum – контрольная сумма файла.

SizeOfOptionalHeader – размер опционального заголовка, который располагается за структурой IMAGE_FILE_HEADER.

NumOfRVAandSizes –размер массива VA/Size, который следует ниже.

В программе LordPE можно рассмотреть содержимое PE-заголовка, если выбрать кнопку меню FLC – калькулятор

местоположения файла. При вводе значения 0 в поле RVA (относительный виртуальный адрес) и нажатия кнопки DO отображается содержимое первых байтов (рис. 4.17).

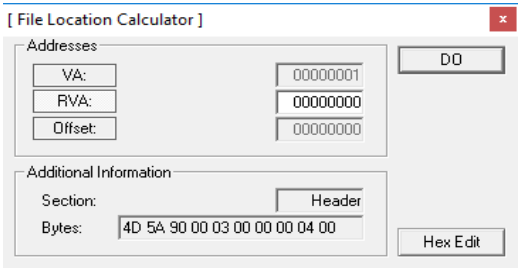


Рис. 4.17. Окно калькулятора FLC анализатора LordPE

При выборе кнопки Hex Edit отображается содержимое всех ячеек файла исследуемой программы (рис. 4.18).

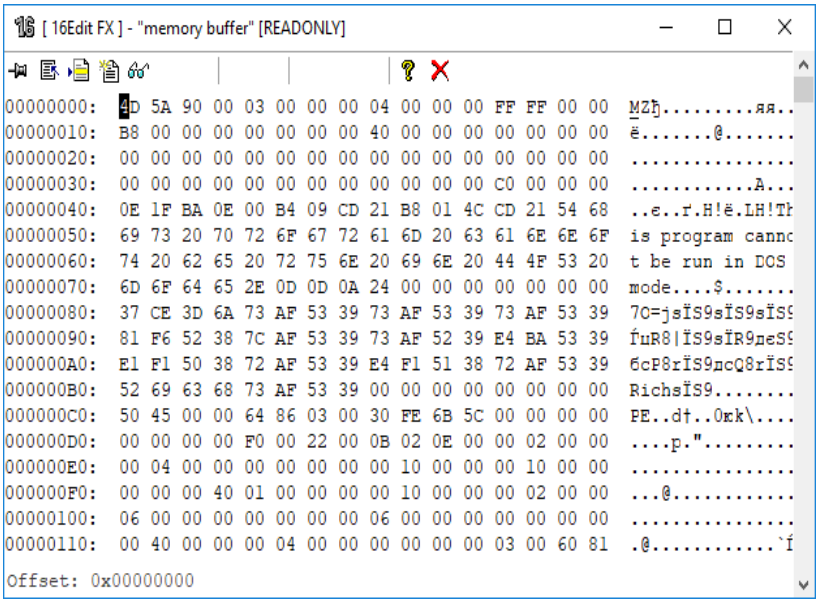


Рис. 4.18. Окно дампа памяти программы в анализаторе LordPE

С нулевого байта располагается структура IMAGE_DOS_HEADER, первые два байта которой содержат сигнатуру заголовка “MZ”, 4-й байт указывает на количество секций в файле (3 секции).

По адресу 0x3c находится адрес нового exe-заголовка PE (в данном случае там находится 32-разрядное значение 00 00 00 C0. А уже с адреса 0x3c находятся значения байтов, которые относятся к структуре IMAGE_NT_HEADERS и содержат сигнатуру "PE\0\0".

Определение свободного места в секции

Общий объем секции кода можно выяснить при открытии исследуемого exe-файла в программах-анализаторах PE-формата. Одним из ярких представителей таких анализаторов является программа CFF Explorer. Как правило секция кода равна 200h = 512 байтов.

Свободный объем секции, куда можно осуществить внедрение, можно определить различными способами, например:

– с использованием отладчика;

– с использованием программы-анализатора PE-формата.

Для определения свободного объема памяти ***при помощи отладчика*** необходимо выполнить последовательность действий:

– открыть exe-файл в отладчике x64Dbg;

– к адресу расположения первой строки кода прибавить размер секции, который определен программой-анализатором PE-формата (как правило 200h);

– от полученного максимального адреса секции вычесть адрес расположения первой свободной строки кода, которая располагается после последней команды программы.

Для определения свободного объема памяти ***при помощи программы-анализатора***, например программы LordPE, необходимо выполнить последовательность действий:

– в программе нажать на клавишу меню PE Editor и выбрать исследуемый exe-файл;

– в появившемся окне нажать клавишу меню Sections;

– выбрать секцию, где расположен текст программы (это секция text), нажать правую клавишу мышки и выбрать пункт меню hex edit section (рис. 4.19);

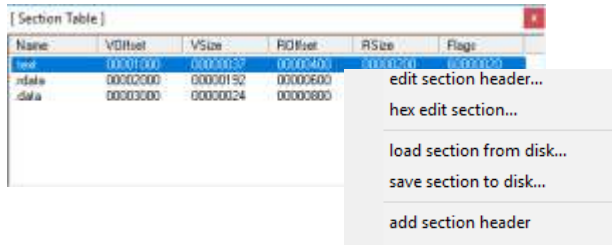


Рис. 4.19. Окно выбора меню анализатора LordPE

– в появившемся окне memory buffer выбранная секция выделится темным цветом (рис. 4.20).

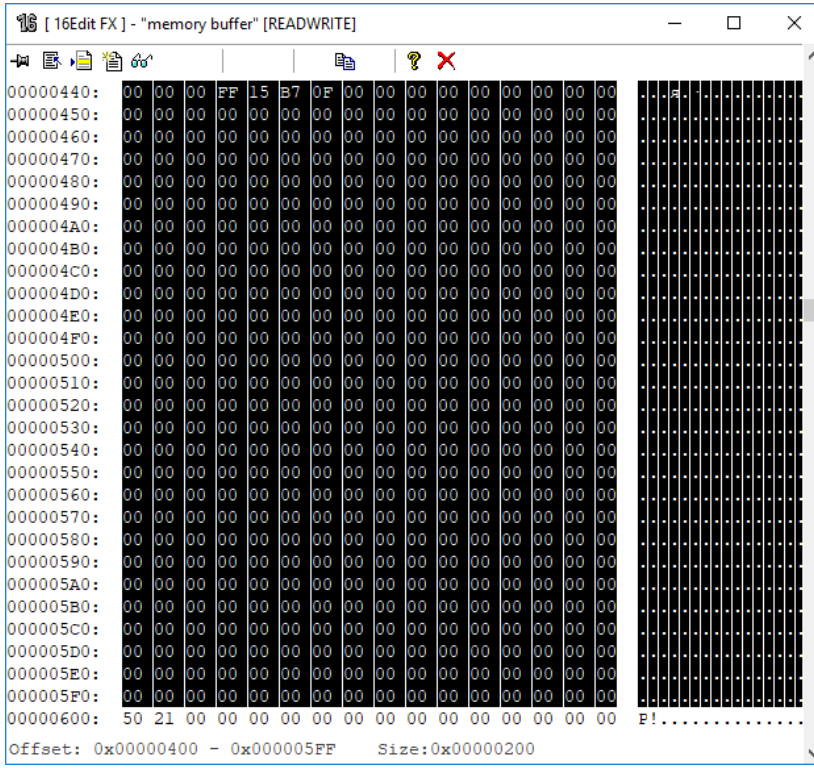


Рис. 4.20. Окно hex редактора секции кода

В конце секции свободные ячейки будут заполнены нулями. Их необходимо просто посчитать.

Для подсчета количества байтов свободных ячеек памяти необходимо из адреса начала следующей секции вычесть первую свободную ячейку после окончания кода: 600h – 447h = 1B9h = 441 байт.

Секция кода имеет объем 200h = 512 байтов и из них – 441 свободных. Следовательно, количество свободных ячеек намного больше, чем занятых, что позволяет внедрить большой блок нового кода.

Рассмотрим пример решения уравнения 32c/7b + √a для целых чисел, которые заданы массивом (программа 4.3).

Программа 4.3. Решение уравнения 32c/7b + √a для целых чисел:

```
        include \masm64\include64\masm64rt.inc ; библиотеки
.data ;
        mas1 dd 4,1,1,32,7 ; a, b, c, 32, 7
.code ;
entry_point proc      ; 32c/7b + Va
        vmovd xmm1,mas1[0]    ; a
        vmovd xmm2,mas1[4]    ; b
        vmovd xmm3,mas1[8]    ; c
        vmovd xmm4,mas1[12]   ; 32
        vmovd xmm5,mas1[16]   ; 7

        vmovd r10d,xmm1          ; a
vcvtsi2ss xmm0,xmm0,r10d ; преобр. 32-разр. целого в вещественное
        vsqrtss xmm0,xmm0,xmm0   ; корень вещественного числа
        cvttss2si r10d,XMM0      ; преобраз.  в 32-разрядное целое число
        vmovd xmm0,r10d          ; сохранение sqrt (a) в хмм0
        vpmulld xmm6,xmm4,xmm3   ; 32c
        vpmulld xmm7,xmm2,xmm5   ;  7b
        vmovd eax,xmm6                  ;  подг. к делению
        vmovd ebx,xmm7                  ;  подг. к делению
        xor edx,edx             ;  подг. к делению
        div ebx                 ; 32c/7b
        vmovd xmm8,eax     ;  сохранение результата деления
        vaddss xmm9,xmm6,xmm8   ; 32c/7b + Va
        movss res1,xmm9 ;

.data
        titl1 db "Решение уравнения. AVX команды.",0
        buf1 dq 0,0; буфер
```

```
        res1 dd 0,0
        fmt1 db "masm64.",10,10,"Уравнение 32c/7b + Va",10,
        "Результат = %d",10,10,
        "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
.code
        invoke wsprintf,addr buf1,ADDR fmt1,res1
        invoke MessageBox,0,addr buf1,addr titl1,MB_ICONINFORMATION
        invoke ExitProcess,0
entry_point endp
        end
```

Для решения этого уравнения задаем числа массивом целых чисел:

```
mas1 dd 4,1,1,32,7 ; a, b, c, 32, 7
```

с порядком расположения в соответствии с приведенным комментарием.

Каждое из целых чисел располагаем в соответствующих регистрах xmm:

```
        vmovd xmm1,mas1[0]    ; a
        vmovd xmm2,mas1[4]    ; b
        vmovd xmm3,mas1[8]    ; c
        vmovd xmm4,mas1[12]   ; 32
        vmovd xmm5,mas1[16]   ; 7
```

Для вычисления корня целого числа необходимо преобразовать его в вещественное:

```
        vmovd r10d,xmm1         ; a
vcvtsi2ss xmm0,xmm0,r10d ; преобр. 32-разр. целого в вещественное
        vsqrtss xmm0,xmm0,xmm0   ; корень вещественного числа
```

После получения корня вещественного числа необходимо результат конвертировать (из ss в si) в целое число:

```
        cvttss2si r10d,XMM0    ; преобраз.  в 32-разрядное целое число
```

Т.к. все числа уже представлены целыми, то применяется и целочисленное деление:

```
        vmovd eax,xmm6    ;  подг. к делению
        vmovd ebx,xmm7    ;  подг. к делению
        xor edx,edx           ; подг. к делению
        div ebx                 ; 32c/7b
```

Для удобства написания программы применено упрощенное сегментирование – это, когда по мере необходимости применяются несколько сегментов data и code (для удобства программирования данные располагаются не далеко от кода).

Откроем полученный exe-файл в программе анализаторе PE-файла с названием LordPE.

Нажимаем кнопку FLC, проставляем в RVA ноль и подтверждаем кнопкой DO. Выбираем кнопку Hex Edit и в появившемся окне анализируем заголовок PE-файла (рис. 4.21).

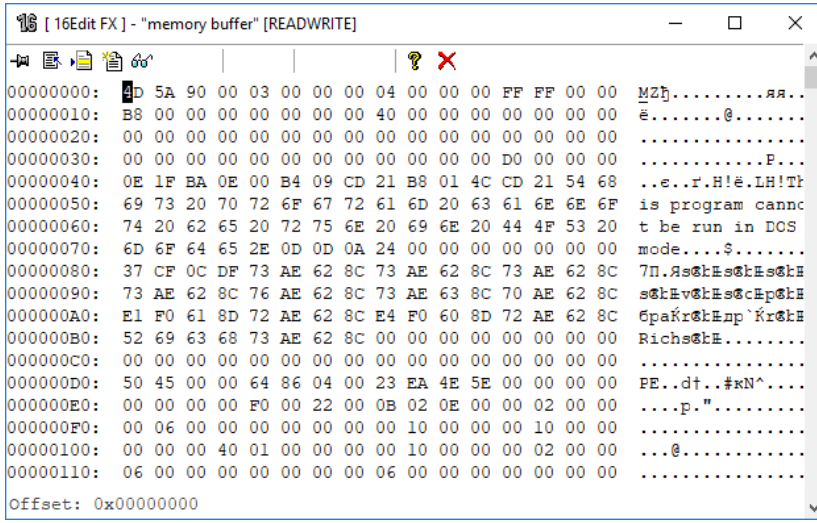


Рис. 4.21. Окно дампа памяти программы в анализаторе LordPE

В нулевой ячейке находится сигнатура MZ, что соответствует DOC заголовку. В ячейке 0x3C находится адрес, по которому начинается PE заголовок.

4.6. Исследование диалоговой программы

Исследование заголовка диалоговой программы ничем не отличается от исследования простых программ, которые не имеют шаблонов окон. Актуальностью этого материала является то, что рассматривается процесс создания и последующего исследования полноценной Windows-программы.

Рассмотрим диалоговую программу для целых и вещественных чисел, заданных массивами решения уравнения ab + c/d с использованием команд AVX.

Разобъем решение этой задачи на этапы:
1). Решение уравнения для целых чисел;
2). Решение уравнения для вещественных чисел;
3) Создание шаблона диалоговой программы;
4) Соединение отдельных программ в единую программу.
Для каждого из этапов применим отладку.

Этап 1. Решение уравнения ab + c/d для целых чисел.

Программа 4.4. Решение уравнения ab + c/d для целых чисел:

```
      include \masm64\include64\masm64rt.inc ; библиотеки
.data ; ab + c/d
      res1 dd 0,0
      titl1 db "Решение уравнения. AVX команды.",0
      buf1 dq 0,0; буфер
      fmt1 db "masm64. Числа целые",10,10,"Уравнение ab + c/d",10,
      "Результат = %d",10,10,
      "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

      mas1 dd 4,1,8,2 ; a, b, c, d   - целые
.code ;
entry_point proc      ; ab + c/d
      vmovd xmm1,mas1[0]  ; a
      vmovd xmm2,mas1[4]  ; b
      vmovd xmm3,mas1[8]  ; c
      vmovd xmm4,mas1[12]; d

      vpmulld xmm5,xmm2,xmm1; ab

      vmovd eax,xmm3       ; подг. к делению
      vmovd ebx,xmm4       ; подг. к делению
      xor edx,edx          ;  подг. к делению
      div ebx              ; c/d
      vmovd xmm6,eax; сохранение результата деления

      vaddss xmm7,xmm6,xmm5;
      movss res1,xmm7 ;

      invoke wsprintf,addr buf1,ADDR fmt1,res1
      invoke MessageBox,0,addr buf1,addr titl1,MB_ICONINFORMATION
      invoke ExitProcess,0
entry_point endp
      end
```

Для решения этого уравнения задаем числа массивом целых чисел:

```
mas1 dd 4,1,8,2 ; a, b, c, d   - целые
```

с порядком расположения в соответствии с приведенным комментарием.

Каждое из целых чисел располагаем в соответствующих регистрах xmm:

```
vmovd xmm1,mas1[0]  ; a
vmovd xmm2,mas1[4]  ; b
vmovd xmm3,mas1[8]  ; c
vmovd xmm4,mas1[12]; d
```

Умножение целых чисел выполняется строкой кода

```
vpmulld xmm5,xmm2,xmm1; ab
```

Деление целых чисел выполняется в целочисленных регистрах. Поэтому, сначала выполняется подготовка к делению, что предполагает занесение чисел в эти целочисленные регистры:

```
vmovd eax,xmm3;  подг. к делению
vmovd ebx,xmm4;  подг. к делению
xor edx,edx.....;  подг. к делению
div ebx .....; c/d
vmovd xmm6,eax;  сохранение результата деления
```

Результат выполнения программы представлен на рис. 4.22.

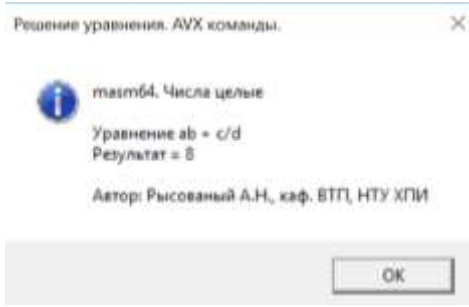


Рис. 4.22. Результат выполнения программы

Этап 2. Решение уравнения для вещественных чисел.

Программа 4.4. Решение уравнения ab + c/d для вещественных чисел:

```
        include \masm64\include64\masm64rt.inc ; библиотеки
.data ; ab + c/d
        titl2 db "masm64. Решение уравнения. AVX команды.",0
        buf2 dq 0,0; буфер
fmt2 db "masm64. Числа вещественные: 4.,1.,8.,2.",10,10,"Уравнение ab
+ c/d",10,
     "Результат = %d",10,10,
     "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

        mas2 DQ 4.,1.,8.,2. ; a, b, c, d   - вещественные

.code ;
entry_point proc      ; ab + c/d
        vmovsd xmm1,mas2[0]    ; a
        vmovsd xmm2,mas2[8]    ; b
        vmovsd xmm3,mas2[16]  ; c
        vmovsd xmm4,mas2[24]  ; d

        vmulsd xmm5,xmm2,xmm1; ab
        vdivsd xmm6,xmm3,xmm4; c/d
        vaddsd xmm7,xmm6,xmm5;
        vcvttsd2si r15d,xmm7; преобразование вещественного в целое
        movsxd r15,r15d ; расширение до 64 разрядов
   invoke wsprintf,addr buf2,ADDR fmt2,r15
   invoke MessageBox,0,addr buf2,addr titl2,MB_ICONINFORMATION
   invoke ExitProcess,0
entry_point endp
        end
```

Результат выполнения программы представлено на рис. 4.23.

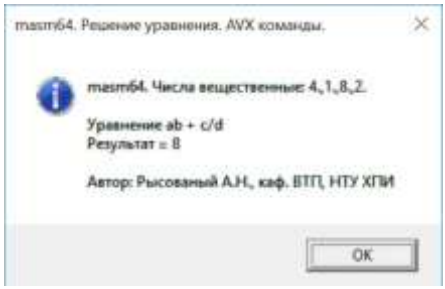


Рис. 4.23. Результат выполнения программы

Этап 3. Создание шаблона диалоговой программы.

Для создания ресурсов (например, вставка иконки, кнопки, JPG-файла и прочее) будем использовать программу ResEd. При создании проекта будут использованы файлы:

1. Prog_user.rc – файл ресурсов пользователя;
2. Prog_user.ico – иконка пользователя;
3. makeit.bat – управляющий файл для использования **ресурсов**;
4. Prog_user.asm – исходный код программы пользователя.

Рассмотрим последовательность создания диалогового окна с предъявленными требованиями:

1. В программе ResEd создаем новый файл ресурсов (File – New Project и вводим имя). В правой верхней панели появляется значок папки и имя файла (рис. 4.24).

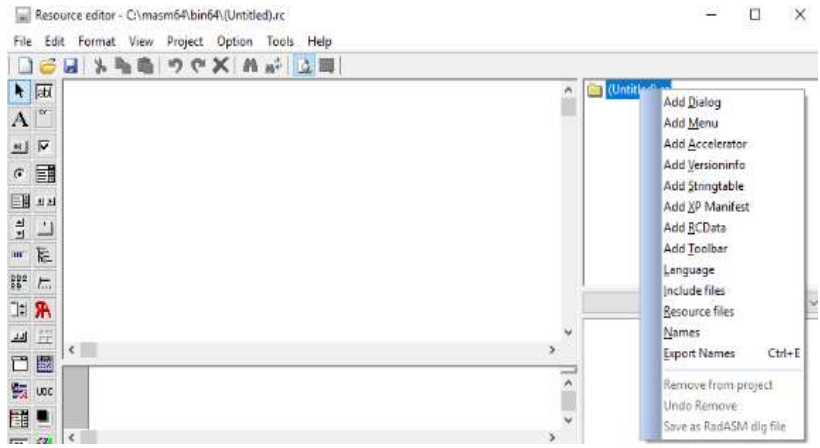


Рис. 4.24. Создание окна в программе ResEd

2. Нажимаем на нее правой кнопкой мыши и выбираем «Add Dialog».

3. Выставляем характеристики окна:

- выставляем мышкой необходимый размер окна;
- в поле свойств окна Caption меняем название диалогового окна.

4. Для размещения картинки (в нашем случае иконки) выбираем кнопку и перетаскиваем ее мышкой на диалоговое окно.

5. Для размещения кнопки выбираем в поле инструментов кнопку с изображением ОК и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим название кнопки.

6. Для написания текста на окне выбираем в поле инструментов кнопку с изображением буквы «А» и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим текст.

7. Для создания рамки (если необходимо) выбираем поле с изображением рамки. В поле Caption вводим название рамки.

8. Добавляем файл ресурсов.

Добавление файла констант. Для этого, нажимаем правой кнопкой мыши по значку папки, выбираем «Include file», затем «Add». Выбираем в окне программы три точки и вводим путь, например C:\masm64\include64\RESOURCE.H.

9. Сохраняем программу.

10. Открываем созданный rc-файл и добавляем строку с ресурсом курсора:

10 ICON DISCARDABLE "smail5.ico”

В этой строке число **10** обозначает цифровой идентификатор иконки (ресурс). Он может быть любым, но и в rc- и asm-файлах он должен быть одним и тем же.

В результате выполненных действий окно программы должно быть таким, как приведено на рис. 4.25.

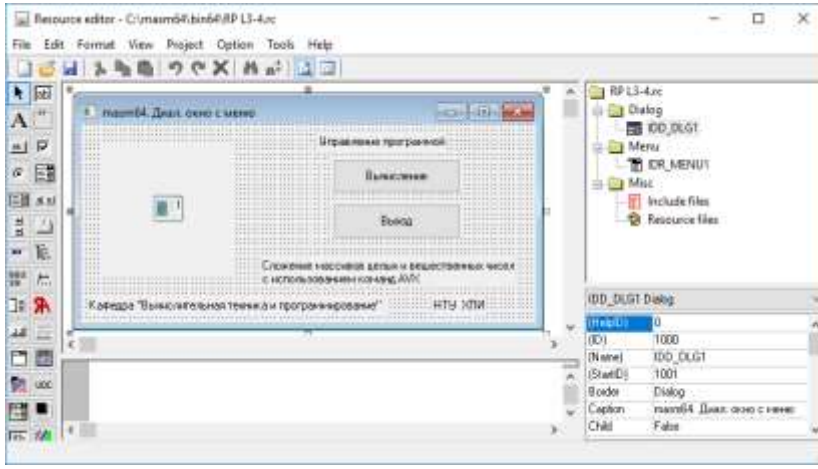


Рис. 4.25. Результат создания окна в программе ResEd

После сохранения внесенных изменений файл ресурсов будет таким, как приведено в программе 4.6.

Программа 4.6. Файл ресурсов rc:

```
#define IDM_1   10001
#define IDD_DLG1 1000
#define IDC_IMG1 1001
#define IDC_BTN1 1002
#define IDC_BTN2 1003
#define IDC_GRP1 1004
#define IDC_STC1 1005
#define IDC_STC2 1006
#define IDC_STC3 1007
#define IDR_MENU1 10000
#define IDM_2 10002
#define IDM_3 10003

#include "C:/masm64/include64/Resource.h"

10 ICON DISCARDABLE "Winking.ico"
11 ICON DISCARDABLE "user.ico"

IDD_DLG1 DIALOGEX 10,10,309,120
CAPTION "masm64. Диал. окно с меню"
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
BEGIN
  CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,15,12,90
,78
  CONTROL "Вычисление",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,171,21,87,21
  CONTROL "Выход",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,171,48,87,21
  CONTROL "Управление программой",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,156,6,111,72
  CONTROL "Кафедра ""Вычислительная техника и
программирование""", IDC_STC1,"Static",
WS_CHILDWINDOW|WS_VISIBLE,6,105,204,9
  CONTROL "НТУ  ХПИ",IDC_STC2,"Static",
WS_CHILDWINDOW|WS_VISIBLE,240,105,36,9
  CONTROL "Сложение массивов целых и вещественных чисел с
использованием команд AVX",IDC_STC3,"Static",
WS_CHILDWINDOW|WS_VISIBLE,123,81,171,18
```

```
END

IDR_MENU1 MENU
BEGIN
  POPUP "Программа"
  BEGIN
    MENUITEM "Решение",IDM_2
    MENUITEM "Выход",IDM_3
  END
END
```

Перед написанием ассемблерной программы и ее компиляции необходимо поменять управляющий (пакетный) файл (имя – makeit.bat; среда программирования от СтиваХатчессона), который является последовательностью инструкций операционной системы. В этом файле должна быть строка для программы управления ресурсами – \masm64\bin64**\rc.exe** %appname%.rc.

Программа 4.7. Файл makeit.bat:

```
:: вывод выполняющихся строк на экран
@echo off
:: установка переменной
set appname=%1

\masm64\bin64\ml64.exe /c %appname%.asm
\masm64\bin64\rc.exe  %appname%.rc
\masm64\bin64\link.exe  /SUBSYSTEM:WINDOWS  /ENTRY:entry_point
/nologo /LARGEADDRESSAWARE %appname%.obj %appname%.res

:: вывод списка файлов
dir %appname%.*

:: удаление файла *.obj
    del %appname%.obj
    del %appname%.res
    pause
```

Этап 4. Соединение отдельных программ в единую программу.

При соединении отдельных программ в общую необходимо эти программы оформить ввиде процедур. При этом, данные в секции data надо располагать раньше, чем эти данные будут вызваны командами из секции code.

Программа 4.8. Шаблон диалоговой программы:

```
      include \masm64\include64\masm64rt.inc
.data?
        hInstance dq ?
        hIcon     dq ?
.code
      entry_point proc
       mov hInstance,rv(GetModuleHandle,0)
 mov hIcon,rv(LoadImage,hInstance,10,IMAGE_ICON,128,128,
LR_DEFAULTCOLOR)
       invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
      invoke ExitProcess,0
         ret
      entry_point endp
      main proc
hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
      .switch uMsg
      .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; установить значок
для диалога
invoke SendMessage,rv(GetDlgItem,hWin,102),\ ; установить значок в
клиентской области
                  STM_SETIMAGE,IMAGE_ICON,lParam

      .case WM_COMMAND
        .switch wParam
          .case 1002
          invoke MsgboxI,0,"каф. ВТП","Вычисление",MB_OK,11
          .case 1003
invoke MsgboxI,0,"Вы подтверждаете выход из программы","Выход", MB_OK,11
        .endsw
      .case WM_CLOSE
 exit1: invoke EndDialog,hWin,0 ;
       .endsw
         xor rax, rax
         ret
      main endp
      end
```

В шаблоне диалоговой программы отсутствуют части кода, которые отвечают за математическую обработку уравнения. На начальном этапе для упрощения при нажатии на кнопки выводятся только сообщения. После отладки этого шаблона можно приступить ко вставке процедур в шаблон.

Программа 4.9. Диалоговая программа:

```
        include \masm64\include64\masm64rt.inc
.data?
          hInstance dq ?
          hIcon     dq ?
.data
        res1 dd 0,0
        titl1 db "Уравнение. Числа целые. AVX команды.",0;
        buf1 dq 0,0; буфер
        fmt1 db "masm64. Числа целые",10,10,"Уравнение ab + c/d",10,
        "Результат = %d",10,10,
        "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0
        mas1 dd 4,1,8,2 ; a, b, c, d   - целые
.data ; ab + c/d
        titl2 db "masm64. Решение уравнения. AVX команды.",0
        buf2 dq 0,0; буфер
fmt2 db "masm64. Числа вещественные: 4.,1.,8.,2.",10,10,"Уравнение ab
+ c/d",10,
"Результат = %d",10,10,
"Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

        mas2 DQ 4.,1.,8.,2. ; a, b, c, d
.code
proc1 proc;процедура ab + c/d
        vmovd xmm1,mas1[0] ;a
        vmovd xmm2,mas1[4] ;b
        vmovd xmm3,mas1[8] ;c
        vmovd xmm4,mas1[12];d
        vpmulld xmm5,xmm2,xmm1 ;ab
        vmovd eax,xmm3  ;  подг. к делению
        vmovd ebx,xmm4  ;  подг. к делению
        xor edx,edx     ;  подг. к делению
        div ebx;

        vmovd xmm6,eax;  сохранение результата деления
        vaddss xmm7,xmm6,xmm5;
        movss res1,xmm7;

invoke wsprintf,addr buf1,ADDR fmt1,res1
invoke MessageBox,0,ADDR buf1,ADDR titl1,MB_ICONINFORMATION
        ret
proc1 endp;

proc2 proc; процедура  ; ab + c/d
        vmovsd xmm1,mas2[0]    ; a
```

```
        vmovsd xmm2,mas2[8]    ; b
        vmovsd xmm3,mas2[16]   ; c
        vmovsd xmm4,mas2[24]   ; d
        vmulsd xmm5,xmm2,xmm1; ab
        vdivsd xmm6,xmm3,xmm4; c/d
        vaddsd xmm7,xmm6,xmm5;
        vcvttsd2si r15d,xmm7 ;
        movsxd r15,r15d
        invoke wsprintf,addr buf2,ADDR fmt2,r15
        invoke MessageBox,0,addr buf2,addr titl2,MB_ICONINFORMATION
        ret
proc2 endp;

entry_point proc
        mov hInstance,rv(GetModuleHandle,0)
 mov hIcon, rv(LoadImage, hInstance,10,IMAGE_ICON, 128,128,
LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,1000,0,ADDR main,hIcon
        invoke ExitProcess,0
           ret
        entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
        .switch uMsg
        .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; значок для диалога
invoke SendMessage,rv(GetDlgItem,hWin,1001),\    ; значок в клиентской
области
              STM_SETIMAGE,IMAGE_ICON,lParam

        .case WM_COMMAND
          .switch wParam

            .case 1002
        invoke proc1
        invoke proc2
            invoke MsgboxI,0,"каф. ВТП","Вычисления окончены",MB_OK,11

            .case 1003
invoke MsgboxI,0,"Вы подтверждаете выход из программы", "Выход",
MB_OK,11
            jmp exit1
          .endsw
        .case WM_CLOSE
 exit1: invoke EndDialog,hWin,0 ;
```

```
 .endsw
    xor rax, rax
    ret
main endp
end
```

Результат выполнения программы представлен на рис. 4.26.

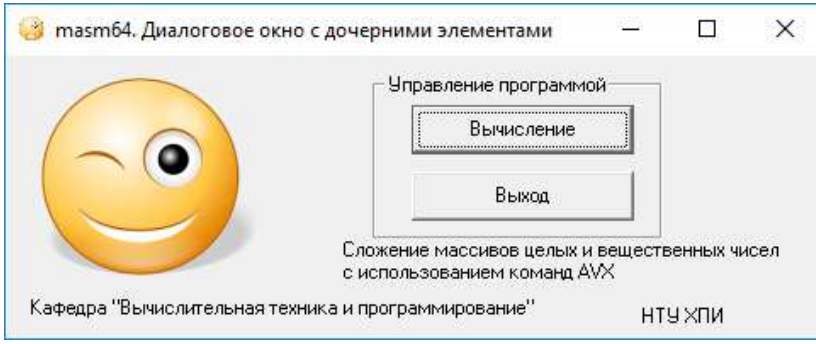


Рис. 4.26. Результат выполнения программы

Откроем полученный exe-файл в программе анализаторе PE-файла с названием LordPE (рис. 4.27).

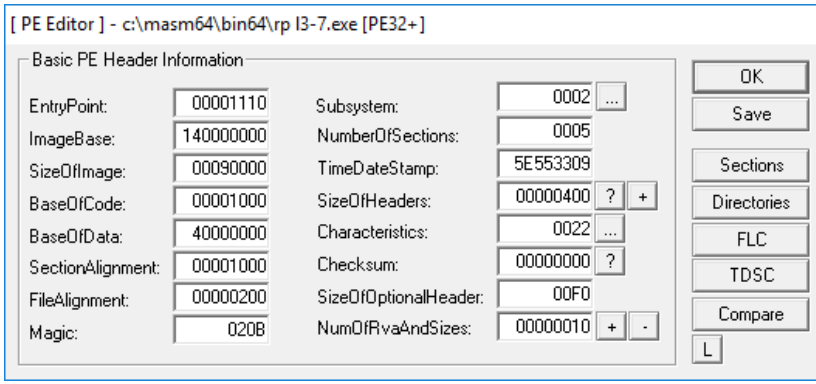


Рис. 4.27. Основное окно программы-анализатора LordPE

Программа анализатор PE-файла названием LordPE позволяет в ручном редактирование исполняемых файлов (Portable Executable). С помощью этой утилиты можно редактировать exe-файлы, анализировать их, оптимизировать, разбивать их на части и собирать обратно. Можно также дампить процесс частично или полностью, а также оптимизировать как дамп, так и сам файл после компилятора. Возможно редактирование информации в заголовке файла (флаг SubSystem, корректировка контрольной суммы, редактирование характеристик и т.д.), а также просмотр и редактирование многих других секций исполняемого файла.

В программа анализаторе PE-файла с названием LordPE необходимо выполнить основные действия (рассмотренные ранее):

– определить название и количество секций программы;
– проанализировать флаги секций;
– проанализировать сигнатуры PE-заголовка;
– определить свободный объем секции кода.

В связи с тем, что одна программа не всегда позволяет определить необходимые системные характеристики exe-файла и в некоторых программах присутствуют неточности, то всегда надо перепроверять полученные данные другими программами-анализаторами. Одной из них есть программа PE.Tools.v1.9.762.2018.

Для просмотра системных характеристик открываем исследуемую программу в PE.Tools. После открытия файла выбираем последовательность действий: Tools / PE Editor → Выбор файла / File Header. Появляется окно (рис. 4.28).

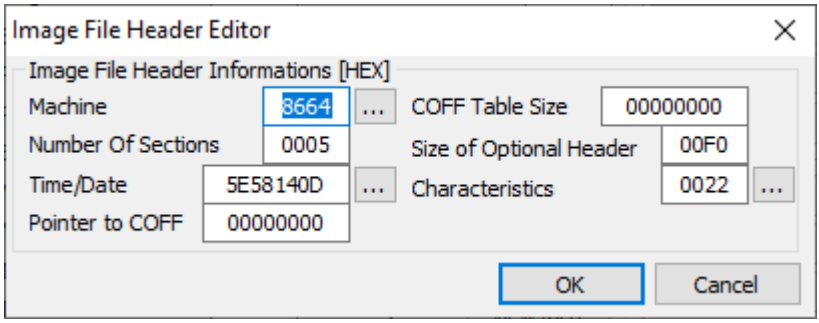


Рис. 4.28. Окно чтения файлового заголовка в программе PE.Tools.v1.9.762.2018

При выборе <…> напротив характеристики Machine выводится название архитектуры микропроцессора (рис. 4.29).

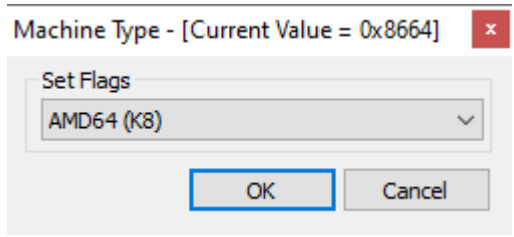


Рис. 4.29. Окно вывода названия архитектуры микропроцессора в программе PE.Tools.v1.9.762.2018

Слово AMD64 обозначает, что рассматриваемая программа написана в платформе x64.

При выборе <…> напротив характеристики Time/Date выводится время и дата создания exe-файла (рис. 4.30).

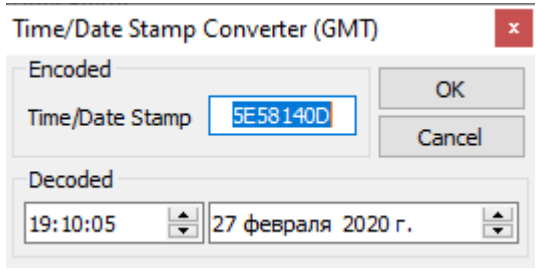


Рис. 4.30. Окно вывода информации о времени и дате создания файла в программе PE.Tools.v1.9.762.2018

При выборе кнопки <Directories> выводится информация о PE-директориях.

Например, числовые идентификаторы используемых ресурсов выводятся при выборе информации о Resource Directory (рис. 4.31).

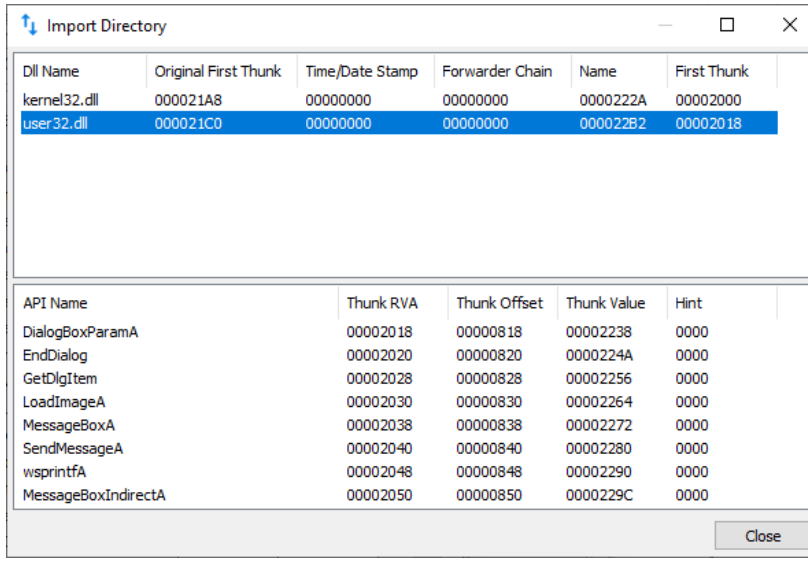


Рис. 4.31. Окно информации о Import Directory в программе PE.Tools.v1.9.762.2018

При двойном клике на название dll-библиотеки выводится информация об этой библиотеке (рис. 4.32).

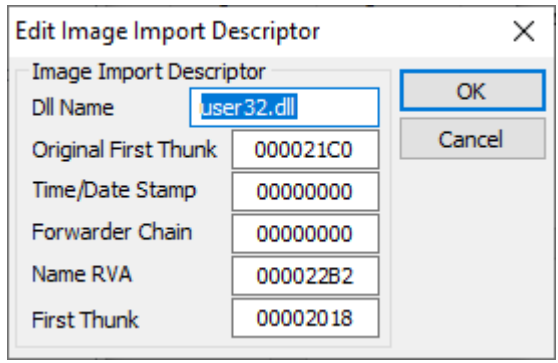


Рис. 4.32. Окно информации о dll-библиотеке в программе PE.Tools.v1.9.762.2018

4.7. Лабораторная работа «Исследование PE-файла»

Цель занятия: приобрести практические навыки анализа PE-файла для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с комментариями;

– результат выполнения программы (скриншоты упрощенного окна (MessageBoxIndirect) и отладчика);

– особенности выполнения, где описать общий алгоритм выполнения вашей программы.

Требования к интерфейсу пользователя

Создать диалоговое окно с ресурсами:

– иконкой (иконками) на диалоговом окне;

– кнопками;

– текстом на окне;

– статическими рамками и др.

Постановка задачи

1. Запрограммировать уравнение для целых и для вещественных чисел при помощи команд AVX и в разных процедурах. Числа заданы массивами. Осуществить проверку возможности микропроцессора поддерживать команды AVX и AVX2;

2. В конце программы вывести время выполнения уравнения и результаты через функцию MessageBoxIndirect (макроса MsgboxI), а также номер варианта, название лабораторной работы, шифр группы, фамилию исполнителя. Программу и результаты привести в отчете.

3. Представить:

— программу вычисления уравнения для целых чисел без использования диалогового окна;

— программу вычисления уравнения для вещественных чисел без использования диалогового окна;

— общую диалоговую программу вычисления уравнения для двух массивов.

4. В отчете представить скриншоты основных полей PE-заголовка и знать их назначение.

В программах анализаторах PE-файла выполнить основные действия (рассмотренные ранее):

– определить название и количество секций программы;
– проанализировать флаги секций;
– проанализировать сигнатуры PE-заголовка;
– определить свободный объем секции кода;
– дату и время создания файла;
– название архитектуры микропроцессора;
– перечень используемых функций.

Задание. Числа *a* є{*a*1,*a*2,*a*3,*a*4} заданы массивом и имеют размерность Real8. Вычислить уравнение:

1. $af\sqrt{(ce)} - bd$;
2. $bd/c + a\sqrt{f} - e$;
3. $c\sqrt{e} + a/b + fd$;
4. $def\sqrt{b} + a/c$;
5. $af/e + b\sqrt{cd}$;
6. $a/b/c + f\sqrt{d}$;
7. $c\sqrt{(de)} - af\sqrt{e}$;
8. $af/\sqrt{b} + \sqrt{e}$;
9. $ab\sqrt{cd} + e/f$;
10. $ad/b\sqrt{c} + e/f$;
11. $a\sqrt{b} - df/e$;
12. $af/b + d\sqrt{c}$;
13. $abc/d - e\sqrt{f}$;
14. $a/b/c + f\sqrt{(de)}$;
15. $af + bc - d\sqrt{e}$;
16. $ab + c/d - f\sqrt{e}$;
17. $af/b + cd\sqrt{e}$;
18. $ed/c - af\sqrt{b}$;
19. $a\sqrt{f} + \sqrt{b} + cdf/e$;
20. $c\sqrt{(de)} + df/a$;
21. $abf/cd + \sqrt{e}$;
22. $af/cd - \sqrt{e} + b$;
23. $ce + f\sqrt{a} + bd$;
24. $a/b/c + f\sqrt{(de)}$;
25. $af\sqrt{b} + c/d - e$,

где *a, b, c, d ,e, f* – числа.

Программа 4.10. Числа *a* є{*a*1,*a*2,*a*3,*a*4} заданы массивом и имеют размерность Real8. Вычислить уравнение *af sqrt(d) + e*:

```
        title Рысованый А.Н.
; masm64. Числа a є{a1,a2,a3,a4} заданы массивом
; и имеют размерность Real8. Вычислить уравнение af sqrt(d) + e

include \masm64\include64\masm64rt.inc

.data
    mas1 real8 1.,2.,3.,4.      ; массив чисел A
    len1 equ ($-mas1)/type mas1
    mas2 real8 16.,2.,1.        ; d., e., f.

tit1 db "masm64. AVX. Результат вычисления уравнения af sqrt(d) + e.",0
      res dq len1 DUP(0),0  ;
      buf1 dd len1 DUP(0),0  ; буфер вывода сообщения
```

```
  ifmt db "masm64.  Вариант №1.",10,
  "ЛР1. Исследование PE-файла",10, "Массив ai = 1., 2., 3., 4.",10,
  9,"Числа: d, e, f  := 16., 2., 1.",10,
  "Результаты вычисления: %d ,%d ,%d ,%d ",10,10,
  "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
  9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code                    ; уравнение af sqrt(d) + e
entry_point proc

.data
    titl db "Проверка МП на поддержку команд AVX",0 ; назв. окна
    szInf db "Команды AVX ПОДДЕРЖИВАЮТСЯ!!!",0  ;
    inf db "Команды AVX микропроцессором НЕ поддерживаются",0
.code
                        ; проверка на поддержку AVX команд

      mov EAX,1 ; при использования 64-разрядной ОС
      cpuid   ; по eax производится идентификация микропроцессора
      and  ecx,10000000h ; ecx:= ecx v 1000 0000h (28 разряд)
      jnz  exit1        ; перейти на exit1, если AVX не поддерживаются
    invoke MessageBox,0,addr inf,addr titl,MB_OK
      jmp exit12
exit1:
    invoke MessageBox,0,addr szInf,addr titl,MB_ICONINFORMATION

.data
      tit2 db "Проверка МП на поддержку команд AVX2",0 ; назв. окна
      szInf2 db "Команды AVX2 ПОДДЕРЖИВАЮТСЯ!!!",0  ;
      inf2 db "Команды AVX2 микропроцессором НЕ поддерживаются",0

.code
      mov eax,7
      mov ecx,0
      cpuid ; по содержимому rax производится идентификация МП
      and  rbx,20h ;   (5 разряд)
      jnz  @f       ; перейти, если не нуль
    invoke MessageBox,0,addr inf2,addr tit2,MB_OK
      jmp m1 ; на выполнение основной части
@@:
    invoke MessageBox,0,addr szInf2,addr tit2,MB_ICONINFORMATION
m1:
                                ; основная часть
      mov rcx,len1
      lea rdx,res
```

```
        lea rbx,mas1
        vmovsd xmm1,mas2[0]  ; xmm1 - d - переслать real8
        vmovsd xmm2,mas2[8]  ; xmm2 - e
        vmovsd xmm3,mas2[16] ; xmm3 – f

        vsqrtsd xmm4,xmm4,xmm1     ; sqrt(d)

          vcvttsd2si eax,xmm4 ; для отладки
        vmulsd xmm5,xmm3,xmm4      ; f x sqrt(d)
          vcvttsd2si eax,xmm5 ; для отладки

@@:
        vmovsd xmm0,qword ptr[rbx] ; xmm0 - a
          vcvttsd2si eax,xmm0 ; для отладки
        vmulsd xmm6,xmm0,xmm5  ; af sqrt(d)
        vaddsd xmm7,xmm6,xmm2  ; af sqrt(d) + e
          vcvttsd2si eax,xmm7            ; для отладки
        movsxd r15,eax ;
        mov [rdx],eax ; сохранение результата
        add rbx,8
        add rdx,8
        dec rcx
        jnz @b ; ссылка на предыдущую метку @@ (наверх)

mov r10,res[0]
mov r11,res[8]
mov r12,res[16]
mov r13,res[24]

  invoke wsprintf,addr buf1,addr ifmt,r10,r11,r12,r13
  invoke MessageBox,0,addr buf1,addr tit1,MB_ICONINFORMATION
exit12:  invoke ExitProcess,0
entry_point endp
end
```

Представлена программа в варианте без переименования имени сегмента кода. В начале программы осуществляется проверка микропроцессора на возможность выполнения команд AVX. Проверка на возможность выполнения команд AVX2 также осуществляется, но алгоритм проверки построен так, что основная часть программы выполняется даже, если команды AVX2 не поддерживаются.

Результат выполнения программы приведен на рис. 4.28.

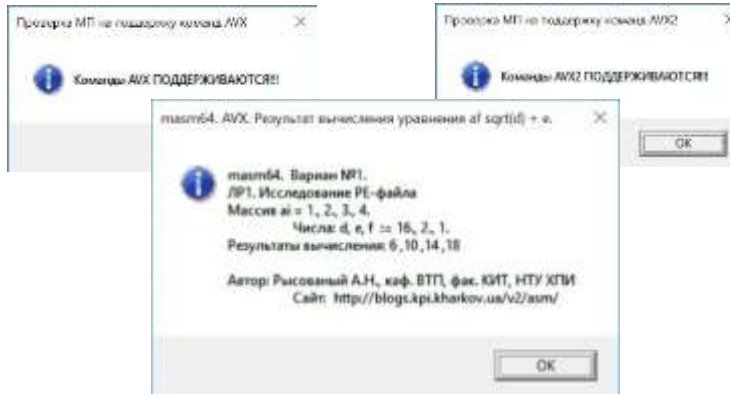


Рис. 4.28. Окна сообщений программы

Для отладки программы в отладчике применяется команда vcvttsd2si. Эту команду можно применять в тем местах программы, которые вызывают сомнения в правильности промежуточных результатов. В окончательном варианте программы эти строчки желательно удалять.

Программа 4.11. Числа целые и заданы массивом. Рассчитать уравнение не менее чем для 3-х значений любой из переменных. Вычислить уравнение $3^3 -$ *sqrt* b (acd):

```
        include \masm64\include64\masm64rt.inc ; библиотеки
.data ;
        mas1 dd 1,4,2,3 ; a, c, d,3 - целые
        masb dd 25,9,16
        titl1 db "Обработка чисел. AVX команды.",0
        buf1 dq 3 dup(0),0; буфер
        res1 dd 3 dup(0),0
        fmt1 db "masm64.",10,10,"Уравнение 3^3 – sqrt b (acd)",10,
        "Результат = ",3 dup(" %d "),10,10,
        "Автор: Рысованый А.Н., каф. ВТП, НТУ ХПИ",0

.code ;
entry_point proc        ; 3^3 – sqrt b (acd)
        lea rbx,masb    ; адр. masb
        lea rdx,res1    ; адр. res1
        mov rcx,3
        vmovd xmm1,mas1[0]    ; a
```

```
        vmovd xmm2,mas1[4]    ; c
        vmovd xmm3,mas1[8]    ; d
        vmovd xmm4,mas1[12]   ; 3
        vpmulld xmm5,xmm4,xmm4 ; 3 x 3
        vpmulld xmm6,xmm5,xmm5 ; 3^3
          ;movss res1,xmm6 ; для проверки результата
          ;mov r10d,res1 ;
        vpmulld xmm7,xmm1,xmm2 ; ac
        vpmulld xmm8,xmm7,xmm3 ; acd

@@:
        vmovd xmm0,dword ptr[rbx];
        vmovd r10d,xmm0 ;
        vcvtsi2ss xmm0,xmm0,r10d;res2
        vsqrtss xmm0,xmm0,xmm0 ;
        cvttss2si r10d,XMM0 ;
        vmovd xmm0,r10d
        vpmulld xmm9,xmm0,xmm8 ;
        vsubss xmm10,xmm6,xmm9
        movss res1,xmm10 ;
        mov r10d,res1 ;
            mov [rdx],r10d
            add rbx,4
            add rdx,4
            dec rcx
            jnz @b

        movsxd r10,res1
        movsxd r11,res1[4]
        movsxd r12,res1[8]

invoke wsprintf,addr buf1,ADDR fmt1,r10,r11,r12
invoke MessageBox,0,addr buf1,addr titl1,MB_OK
invoke ExitProcess,0
entry_point endp
        end
```

Для оформления отчета по лабораторной работе необходимо использовать материал, который рассмотрен ранее.

5. СТРУКТУРЫ PEB И TEB

Для того, чтобы приложения в ОС не соревновались за системные ресурсы имеется локальная память потока (Thread Local Storage). Решить проблему соревнования за ресурсы можно как при помощи механизма синхронизации (семафоры, эвенты и пр.), так и при помощи блока окружения TEB (Thread Environment Block).

При подготовке процесса к запуску, система сначала строит для него контекст (структуры KPROCESS и EPROCESS, фреймы физической памяти и т.д.). Делается это для того, чтобы находящиеся в пространстве пользователя системные механизмы не обращались к ядру ОС. Поэтому система создаёт для процесса структуру под названием PEB (Process Environment Block), куда записывает всю подготовительную информацию.

У любого процесса имеется как-минимум один поток, для которого система создаёт его личное окружение TEB (Thread Environment Block) размером в одну 4К-байтную страницу. Но программист может создавать для своего приложения ещё несколько дополнительных потоков. А для каждого потока создается свой TEB.

Структура PEB (Process Environment Block) – структура процесса ОС Windows, которая заполняется загрузчиком на этапе создания этого процесса. Структура PEB содержит информацию об окружении, загруженных модулях (LDR_DATA), базовой информации по текущему модулю и другие критичные данные, необходимые для функционирования процесса. Информация о версии Windows также записывается в PEB.

Многие системные API-функции ОС Windows, получающие информацию о модулях (библиотеках) в процессе, вызывают API-функцию ReadProcessMemory для считывания информации из PEB нужного процесса.

Для хранения информации о потоках в текущем процессе используется **структура TEB**. Каждый поток имеет свою структуру TEB. Подсистема операционной системы Windows, позволяющая запускать 32-битные приложения Wow64, **имеет два Process Environment Blocks и два Thread Environment Blocks.**

На текущий TEB всегда указывает сегментный регистр процессора FS на x32 системах и **GS на x64**.

Структура TEB создается функцией MmCreateTeb.

Структура PEB создается функцией MmCreatePeb.

Для получения структуры TEB конкретного потока можно воспользоваться функцией NtQueryInformationThread.

Структура TEB имеет синтаксис:

```
typedef struct _TEB {
  PVOID Reserved1[12];
  PPEB  ProcessEnvironmentBlock ; блок среды процесса
  PVOID Reserved2[399];
  BYTE  Reserved3[1952];
  PVOID TlsSlots[64]              ; слоты TLS
  BYTE  Reserved4[8];
  PVOID Reserved5[26];
  PVOID ReservedForOle;
  PVOID Reserved6[4];
  PVOID TlsExpansionSlots ; слоты расширения TLS
} TEB, *PTEB;
```

Доступ к значениям полей TlsSlots и TlsExpansionSlots можно осуществить при помощи функции TlsGetValue. А доступ к значению поля ReservedForOle – при помощи функции CoGetContextToken.

Для получения адреса 64-битной структуры TEB 64 используется значение

```
mov rax, qword ptr gs: [0x30h]
```

Структура PEB имеет синтаксис:

```
typedef struct _PEB {
   BYTE   Reserved1[2];
   BYTE   BeingDebugged;
   BYTE   Reserved2[21];
   PPEB_LDR_DATA LoaderData;
   PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
   BYTE   Reserved3[520];
   PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
   BYTE   Reserved4[136];
   ULONG SessionId;
} PEB;
```

Параметр BeingDebugged структуры PEB – указывает, отлаживается ли в данный момент указанный процесс. Однако структура PEB - это внутренняя структура операционной системы и в будущих версиях может измениться. Поэтому, для отслеживания отладки рекомендуется использовать функцию CheckRemoteDebuggerPresent.

Параметр Ldr – указатель на структуру PEB_LDR_DATA, предоставляющую информацию о загруженных модулях.

Параметр ProcessParameters – указатель на структуру RTL_USER_PROCESS_PARAMETERS, предоставляющую информацию о параметрах запуска процесса.

Параметр PostProcessInitRoutine – функция обратного вызова, которая вызывается после инициализации DLL, но до вызова основного исполняемого кода.

Для получения **адреса структуры PEB** 64-разрядных приложений можно использовать строку кода:

```
mov  rax,qword ptr gs:[60h]
```

Внутри структуры TEB отводятся поля для локальной памяти TLS. Локальная память процессов TLS бывает статической (выделяется на этапе компиляции программы) и динамической, прописанной в TEB.

5.1. Использование структуры PEB функцией IsDebuggerPresent

В качестве примера использования полей структуры PEB можно рассмотреть функцию проверки отладки **IsDebuggerPresent**() и ее параметра BeingDebugged. Этот параметр определяется как смещение byte ptr [rax+2] и применяется для определения отладки (программа 5.1).

Программа 5.1.

```
; Исследование API-функции IsDebuggerPresent"
include \masm64\include64\masm64rt.inc ;
.data
titl1 db "Исследование трассировки",0
szTest db "Программу НЕ трассируют.",10,
"Исследование функции IsDebuggerPresent",0
szTest2 db "Программу трассируют.",10,
"Исследование функции IsDebuggerPresent",0
.code
entry_point proc
  ; invoke IsDebuggerPresent
; функция IsDebuggerPresent заменяется на две строки:
mov  rax,qword ptr gs:[60h ]; получение адреса структуры PEB x64
movzx rax,byte ptr [rax+2] ; поле BeingDebugged в структуре PEB x64
cmp rax,0     ; сравнение
  jnz debug1 ;  перейти, если не ноль
invoke MessageBoxTimeout,0,addr szTest,addr titl1,
MB_ICONINFORMATION,0,5000
jmp @f   ; безусловный переход в конец кода @@ (finish)
```

```
debug1:
      invoke MessageBox,0,addr szTest2,addr titl1,MB_OK
@@:
invoke ExitProcess,0
      entry_point endp
      end
```

Если исполняемый файл программы 5.1 выполняется не под отладчиком, то выводится сообщение "Программу НЕ трассируют." При трассировке программы в отладчике выводится сообщение "Программу трассируют". Особенностью такого применения смещений является то, что при отладке отсутствует название функции. А такой прием тоже затрудняет процесс отладки.

5.2. Использование глобальной системной переменной NtGlobalFlag из структуры PEB

В Windows NT существует набор флагов, которые хранятся в глобальной переменной NtGlobalFlag в структуре PEB и является общей для всей системы.

Чтобы проверить, был ли запущен процесс с отладчиком, необходимо проверить значение поля NtGlobalFlag в структуре PEB. Это поле расположено по смещениям 0x068 (для x32) и **0xBCh** (для x64), относительно начала структуры PEB (программа 5.2).

Программа 5.2.

```
      include \masm64\include64\masm64rt.inc ;
.data
      titl1 db "Исследование трассировки",0
      szTest db "Программу НЕ трассируют.",10,
      "Исследование функции IsDebuggerPresent",0
szTest2 db "Программу трассируют.",10,
"Исследование структуры PEB глобальной системной переменной
NtGlobalFlag ",0
.code
entry_point proc
    mov   rax,qword ptr gs:[60h]
    movzx eax,byte ptr [rax+0BCh]
      cmp rax,0       ; сравнение
      jnz debug1      ;  перейти, если не ноль
invoke MessageBoxTimeout,0,addr szTest,addr titl1,
MB_ICONINFORMATION, 0,5000
        jmp @f    ; безусловный переход в конец кода @@ (finish)
```

```
debug1:
        invoke MessageBox,0,addr szTest2,addr titl1,MB_OK
@@:
        invoke ExitProcess,0
        entry_point endp
        end
```

5.3. Получение адресов системных функций

Для получения адресов системных функций можно воспользоваться как пунктом «Журнал» в отладчике x64Dbg при отладке x64-кода, так и при помощи программы 5.3.

Программа 5.3.

```
        include \masm64\include64\masm64rt.inc
.data
  titl1 db  'Исследование адресов загрузки библиотек',0
  text  db  'База в памяти системных библиотек',10
        db  '---------------------------------',10
  buff  db   256 dup(0)
  frmt db "Ntdll.dll...",9,": %016I64xh",10 ;// спецификаторы для wsprintf()
       db "Kernel32.dll...",9,": %016I64xh",10
       db "KernelBase.dll...",9,": %016I64xh",10
       db "user32.dll... ",9,": %016I64xh",10
       db "gdi32.dll... ",9,": %016I64xh",0
.code
              comment * ----------------------------
Ссылки:  ; https://codeby.net/threads/asm-dlja-x86-4-0-sistemnye-
mexanizmy-windows.68186/
     ; https://quares.ru/?id=175295
     ; https://codeby.net/threads/asm-dlja-x86-4-4-struktura-zagolovka-pe-
fajla.68382/
     ; https://xakep.ru/2018/01/17/antidebug/
              ---------------------------------------------*
entry_point proc

invoke  GetModuleHandle,"ntdll.dll",0
        mov r15,rax
invoke  GetModuleHandle,"kernel32.dll",0
        mov r14,rax
invoke  GetModuleHandle,"KernelBase.dll",0
        mov r13,rax
invoke  GetModuleHandle,"user32.dll",0
        mov r12,rax
invoke  GetModuleHandle,"gdi32.dll",0
```

```
        mov r11,rax

invoke  wsprintf,addr buff,addr frmt,r15,r14,r13,r12,r11
 invoke  MessageBox,0,addr text,addr titl1,0
         invoke  ExitProcess,0
        entry_point endp
        end
```

Результат работы программы 5.3 представлен на рис. 5.1, а в отладчике – на рис. 5.2.

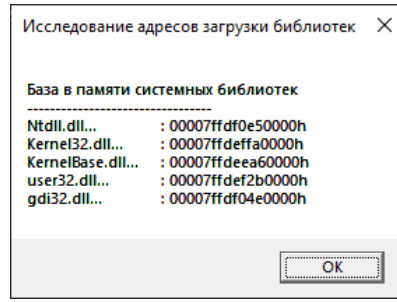


Рис. 5.1. Вывод адресов системных функций

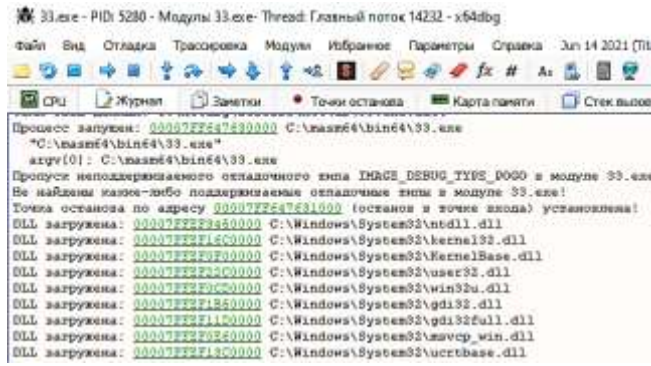


Рис. 5.2. Адреса системных функций в отладчике

Системы класса Win32/64 загружают библиотеки всегда в определённом порядке – сначала Ntdll.dll, потом Kernel32.dll и только затем все остальные.

6. ВНЕДРЕНИЕ КОДА В EXE-ФАЙЛ В КОНЕЦ СЕКЦИИ КОДА С ПОМОЩЬЮ ОТЛАДЧИКА X64DBG БЕЗ ДОБАВЛЕНИЯ API-ФУНКЦИЙ

6.1. Алгоритм внедрения кода

Технология внедрения кода в exe-файл применяется для изменения исходного функционирования программы при отсутствии ее исходного текста.

При низкой подготовке хакера внедрение кода вызывает проблемы и ошибки в приложениях. Эту технологию часто используют и вредоносные приложения. Фактически, внедрение кода – это грязный взлом.

Но в учебных целях технологию внедрения знать надо хотя бы по тому, что необходимо представлять этот механизм и противостоять ему.

Сначала рассмотрим механизм внедрения кода без использования новых API-функций (как самый простой вариант внедрения). А если внедрять API-функции, то уже необходимо представлять все параметры внедряемых функций и способы их передачи. А это в учебных целях несколько усложняет процесс внедрения.

Для внесения изменений в exe-файл необходимо найти в программе свободное место. В простейшем случае, когда код внедрения небольшой и простой (т.е. не внедряются дополнительные API-функции) – его почти всегда можно найти в конце исполняемого exe-файла (CodeCave). Механизм внедрения состоит в том, что сначала одну или две команды (в зависимости от необходимого количества байтов) необходимо скопировать и перенести в конец exe-файла. А на их место поставить jmp на новый адрес, куда перенесена команда (команды). После написания в конце exe-файла нового фрагмента последней командой необходимо также поставить команду jmp на адрес возврата – адрес первой неизмененной в начале exe-файла команды.

Алгоритм внедрения простейшего кода (без добавления API-функций) в exe-файл может быть следующий:

1. Исследование файла программой-анализатором PE-файла (например программой CFF_Explorer) для определения языка

программирования (или компилятора), разрядности архитектуры (32/64), упаковщика или криптора. Естественно, если программа является собственной разработкой, то этот пункт опускается.

2. Подсчет числа свободных байтов в секции кода. Пример такого подсчета при помощи программы-анализатора LordPE детально был изложен при рассмотрении PE-формата.

3. Работа в отладчике.

3.1. Открываем в отладчике x64Dbg ранее полученный exe-файл: Файл/Открыть.

3.2. Выполняем программу в отладчике x64Dbg, нажимая кнопку F9.

3.3. Копируем одну (если ее размер равен или больше 5 байтов) или две (для того, чтобы вставить команду jmp с адресом перехода) команды с начала программы.

3.4. Вставляем скопированные команды с начала программы в конец файла и копируем адрес начала этого нового фрагмента.

3.5. На освободившееся место в начале программы вставляем команду jmp с адресом на начало нового в конце кода фрагмента.

3.6. Изменяем и/или дописываем необходимый код.

3.7. Последней командой в конце нового фрагмента должна быть команда jmp с адресом возврата – адресом первой неизмененной в начале программы команды.

3.8. Сохраняем изменения в отладчике x64Dbg.

6.2. Пример внедрения кода

Рассмотрим процесс внедрения на примере программы 5.1, в которой решается уравнение ab+c/d в среде masm64 и осуществляется вывод времени ее выполнения. В полученный exe-файл внедрим новое уравнение 48ab+32c/d и поменяем текст в окне вывода информации.

Программа 6.1. Решение уравнения ab+c/d в среде masm64 и вывод времени ее выполнения:

```
    include \masm64\include64\masm64rt.inc ;
.data
    a1 dq 1 ; резервирование в памяти 8 байтов для переменной
    b1 dq 2 ;
    c1 dq 4 ;
    d1 dq 2
    res1 dq ?

    titl db "Вывод через функцию MessageBox",0; назв. упрощенного окна
```

```
      st1  dq ?,0  ; буфер вывода сообщения
   ifmt  db "Вывод чисел с памяти через MessageBox:",10,9,"ab+c/d",10,
   "Время выполнения  = %d тиков",10,  ;
    "a = %d",10,"b = %d",10,"c = %d",10,"d = %d",10,"res = %d",10,10,
    "Автор программы: Рысованый А.Н., НТУ ХПИ, каф. ВТП",0 ;

.code ;
entry_point proc                    ; ab+c/d

; определение первоначального числа тиков
      rdtsc          ; edx,eax
      shl rdx,32    ; смещение в свободную старшую часть
      add rdx,rax ; ст. и мл. части в одном регистре
      mov r15,rdx

; вычисление уравнения
       mov r10,a1     ; a:=1
       imul r10,r10,2 ; ab
       xor rdx,rdx      ; подготовка к делению
       mov rax,c1       ; c:=4
       div d1           ; c/d
       add rax,r10      ; ab+c/d
       mov res1,rax

; определение окончательного числа тиков
      rdtsc ; edx,eax
       shl rdx,32    ; смещение в свободную ст. часть
       add rdx,rax ; ст. и мл. части в одном регистре
       sub rdx,r15 ;
       mov r15,rdx ;

invoke wsprintf,ADDR st1,ADDR ifmt,r15,a1,b1,c1,d1,res1
invoke MessageBox,0,addr st1,addr titl,MB_ICONINFORMATION;

invoke ExitProcess,0;возвращение упр. ОС и освобожд. ресурсов
      entry_point endp
      end
```

Результат выполнения программы приведен на рис. 6.1.

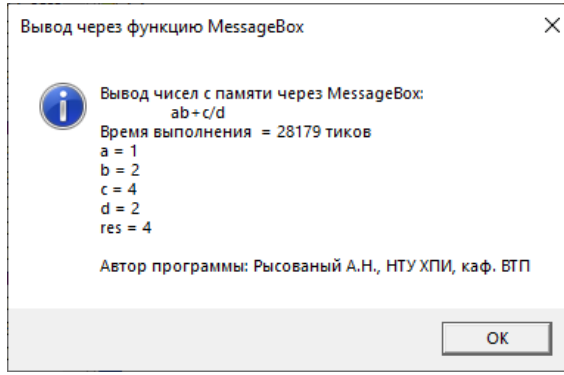


Рис. 6.1. Результат выполнения программы

Первая команда программы – это rdtsc. Команда rdtsc – ассемблерная 32-разрядная инструкция, которая читает счётчик TSC (*Time Stamp Counter*) и возвращающая в регистрах EDX:EAX количество тактов с момента последнего сброса процессора. В регистре EDX содержится старшая часть тиков, а в регистре EAX – младшая. Если анализировать содержимое 64-разрядных регистров RDX и RAX, то в старших половинах этих регистров содержатся нули, т.к. инструкции 32-разрядные. Общий 64-разрядный результат можно получить, если младший 32-рядрядный результат регистра RDX сдвинуть в свободную 32-разрядную старшую часть, а затем сложить с младшей частью регистра RAX. Эти действия выполняет блок кода:

```
rdtsc   ; edx,eax
shl rdx,32 ; смещение в свободную ст. часть
add rdx,rax ; ст. и мл. части в одном регистре
```

В простейшем случае, если блок кода совсем небольшой, то для расчета числа тиков можно использовать только содержимое регистра EAX.

Команда rdtsc чаще всего используется:

– для измерения времени;

– в антиотладочных целях;

– для точного измерения временных интервалов, в том числе при проведении оптимизации (измерение времени, необходимого для выполнения конкретных инструкций или их набора);

– как источник энтропии для генераторов псевдослучайных чисел.

Содержимое ячеек памяти, которые указаны в программе можно просмотреть, если нажать правую кнопку мыши и выбрать Перейти к дампу/ Константа.

Проанализируем exe-файл в программе-анализаторе PE-формата.

В начале процесса внедрения необходимо подсчитать ***количество свободных байт в секции кода***. Для этого необходимо воспользоваться программой-анализатором, например CFF Explorer, LordPE или др.

Воспользуемся программой-анализатором LordPE. Для этого выполним последовательность действий:

– в программе нажать на клавишу меню PE Editor и выбрать исследуемый exe-файл;

– в появившемся окне нажать клавишу меню Sections;

– выбрать секцию, где расположен текст программы (это секция text), нажать правую клавишу мышки и выбрать пункт меню hex edit section;

– в появившемся окне memory buffer выбранная секция выделится темным цветом (рис. 6.2). В конце секции свободные ячейки будут заполнены нулями. Их необходимо просто посчитать.

Для подсчета количества байтов свободных ячеек памяти необходимо из начального адреса следующей секции вычесть первую свободную ячейку памяти после окончания кода:

$$600\ h - 4C4\ h = 13C\ h = 316 \text{ байтов.}$$

Полученный результат свидетельствует о том, что для внедрения небольшого кода и изменения имеющейся функции вывода сообщения такого объема ячеек памяти вполне хватит. Например, функция wsprintf занимает в программе в среднем 80 байтов, а функция MessageBox – в среднем 35 байтов. Но если функция MessageBox используется для вывода большого блока текстовых данных, то размер этой функции резко увеличивается. API-функции с большим количеством параметров занимают в памяти тоже достаточно большой объем.

Проанализируем exe-файл в отладчике x64Dbg.

При анализе кода видно (рис. 6.3), что в начале программы встроены две команды, которые предназначены для резервирования стека и его выравнивания. Эти команды относятся к среде программирования masm64 и в тексте нашей программы отсутствуют. За этими двумя строками начинается код программы.

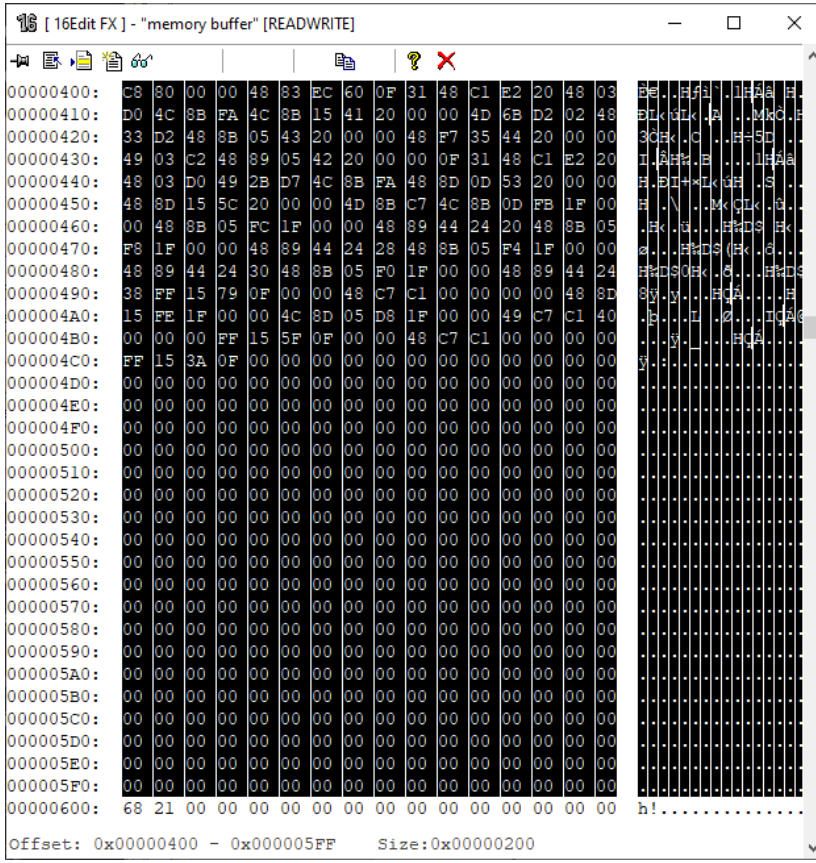


Рис. 6.2. Окно hex редактора секции кода программы LordPE

Программа состоит из блоков:

– получения первоначального количества тиков;

– вычисления функционала программы;

– получения количества тиков после выполнения основного функционала;

– преобразования и вывода информации в упрощенном окне.

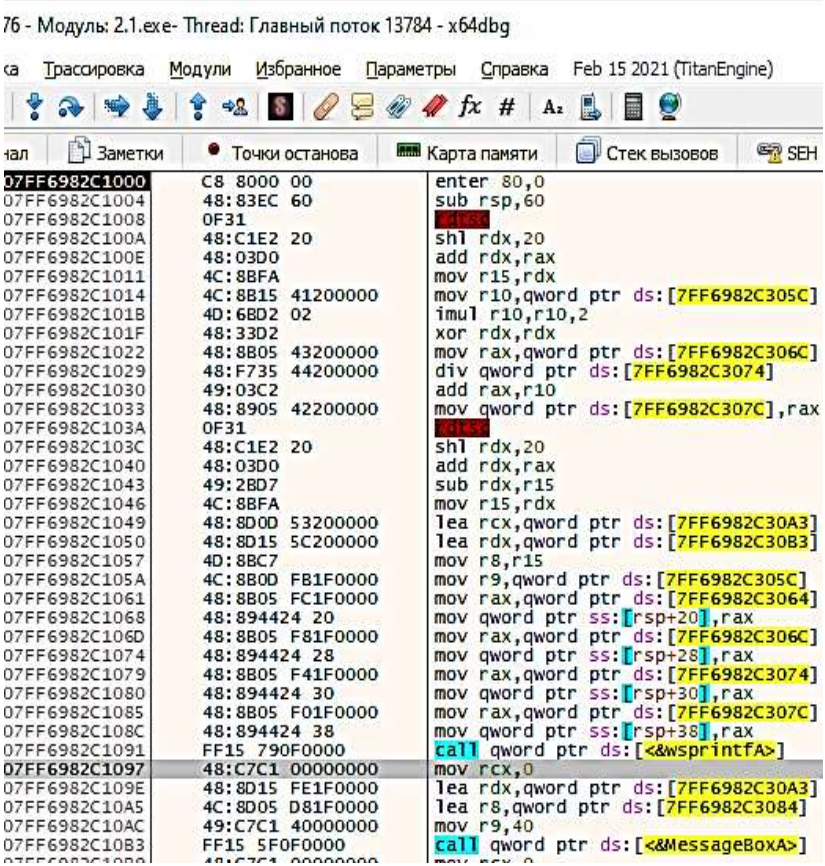


Рис. 6.3. Код программы в отладчике

Рассмотрим внедрение нового кода.

Для внедрения другого функционала проще написать еще одну программу с вычислением нового уравнения. В нашем случае, это уравнение 48ab+32c/d (программа 6.2).

Программа 6.2. Решение уравнения 48ab+32c/d в среде masm64:

```
include \masm64\include64\masm64rt.inc ;
.data
    a1 dq 1 ;
```

```
        b1 dq 2 ;
        c1 dq 4 ;
        d1 dq 2
titl db "Вывод через функцию MessageBox",0; назв. упрощенного окна
st1  dq ?,0  ; буфер вывода сообщения
ifmt db "Вывод чисел с памяти через MessageBox:",10,9,
48ab+32c/d",10,10,  "res = %d",10,10,
   "Автор программы: Рысованый А.Н., НТУ ХПИ, каф. ВТП",0 ;

.code ;
entry_point proc ; 48ab+32c/d
        mov r10,a1     ; a:=1
        imul r10,r10,2 ; ab
        mov rax,48
        mul r10
        mov r11,rax

        xor rdx,rdx    ; подготовка к делению
        mov rax,c1  ; c:=4
        div d1     ; c/d
        mov r12,32
        mul r12
        add rax,r11 ; 48ab+32c/d

invoke wsprintf,ADDR st1,ADDR ifmt,rax
invoke MessageBox,0,addr st1,addr titl,MB_ICONINFORMATION;
invoke ExitProcess,0;возвращение упр. ОС и освобожд. ресурсов
        entry_point endp
        end
```

Результат выполнения этой программы представлен на рис. 6.4.

В программе уже нет смысла еще раз использовать инструкции rdtsc, т.к. их они уже применены в основной программе.

После получения exe-файла откроем его в отладчике x64dbg. Просмотр в отладчике позволяет увидеть код программы и представление параметров в памяти. Кроме того, нам предстоит копировать код из одного отладчика в другой. Таким образом, одновременно будут открыты два отладчика с двумя exe-файлами: первый exe-файл – это файл, в который будет осуществляться внедрение; второй exe-файл – файл, в котором будет осуществляться копирование строк кода в буфер и их вставка в исходный первый файл.

Exe-файл показан в отладчике x64dbg на рис. 6.5.

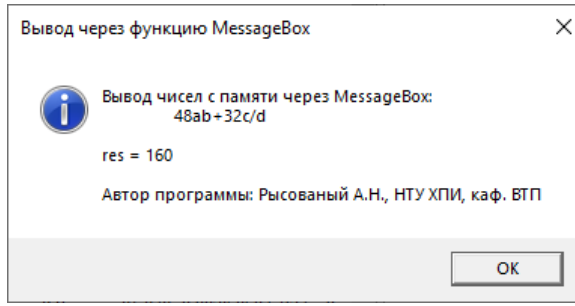


Рис. 6.4. Результат выполнения программы

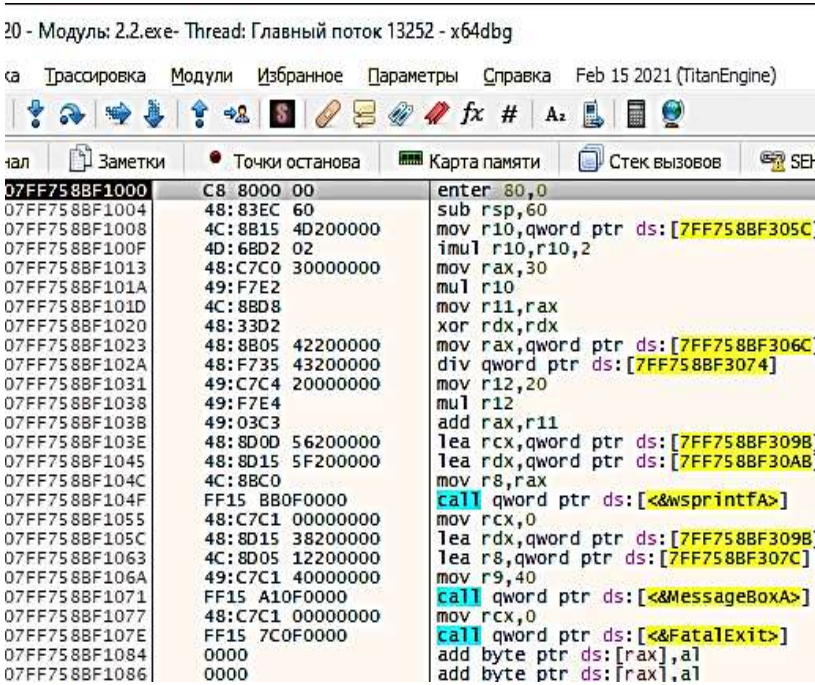


Рис. 6.5. Результат выполнения программы

Открытое окно отладчика с программой 6.2 будет необходимо для переноса команд в отладчик в программу 6.1.

Удаляем ненужный код в основной программе.

Для этого анализируем в исходном коде (код, в который осуществляется внедрение) строки, начиная с которого будем менять функционал. Такой строкой будет строка

```
xor rdx,rdx
```

И ищем строку кода, начиная с которой код остается без изменений. Это строка

```
mov qword ptr ds:[7FF773D8307C],rax
```

Для удаления каждой из строк необходимо выполнить действия:

– выделяем мышкой строку, которую необходимо удалить;

– нажимаем правую кнопки мыши и выбираем Ассемблировать или, для удобства, нажимаем клавишу клавиатуры Space (Пробел);

– в появившемся окне вводим пустую команду NOP и обязательно ставим галочку в поле Заполнить командами NOP (рис. 6.6);

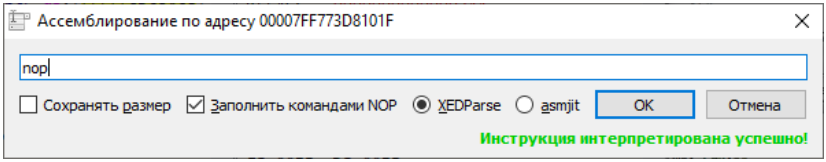


Рис. 6.6. Окно ассемблирования

По очереди удаляем все ненужные строки кода. В результате главное окно отладчика будет иметь вид (рис. 6.7).

В случае, если необходимо только скопировать дизассемблерный код и вставить его, например, в блокнот, то выполняем действия: Shift + правая клавиша мышки (для выбора нескольких команд); «Копировать»/ «Дизассемблерный код».

Осуществляем перенос кода.

Первым действием необходимо поставить команду передачи управления (инструкция JMP) в новый блок кода. Для внедрения кода можно использовать и освободившиеся ячейки. Но в учебных целя внедрение нового блока кода будем осуществлять в конец секции кода после строки вызова функции FatalExit. Эта функции находится внутри обертки ExitProcess.

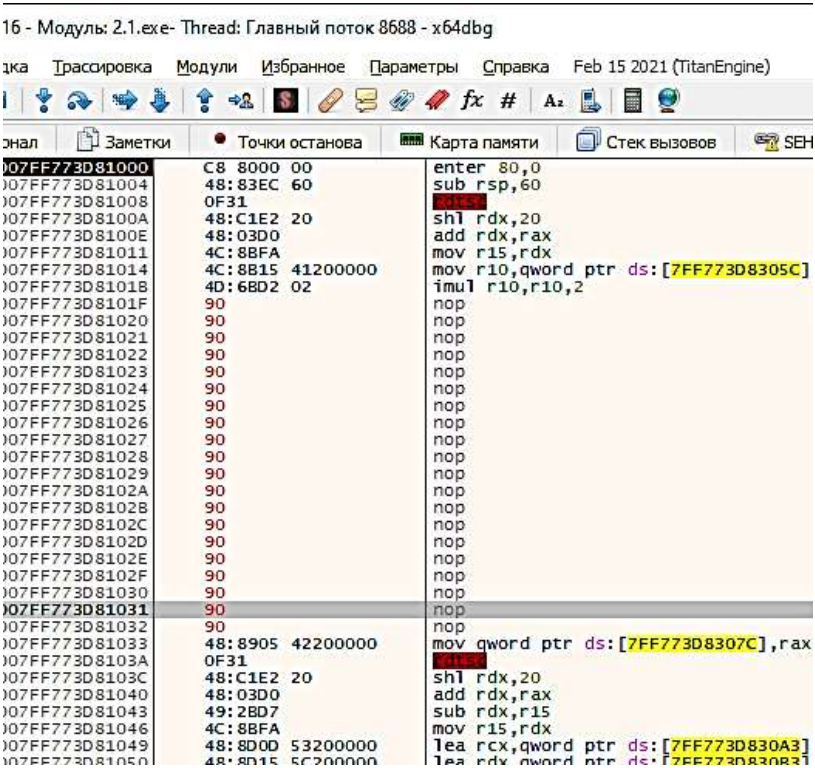


Рис. 6.7. Результат удаления ненужного функционала

Необходимо запомнить адрес первой пустой строки вначале нового блока. Для этого нажимаем на ней правую кнопку мыши и выбираем Копировать/ Адрес.

После этого ставим курсор в начало кода после команды

```
imul r10,r10,2
```

на первую инструкцию NOP и нажимаем клавишу пробела. В появившемся окне ассемблирования вводим инструкцию JMP, затем пробел, а потом вставляем адрес сочетанием клавиш Ctrl + V. Окно ассемблирования будет иметь вид, показанный на рис. 6.8.

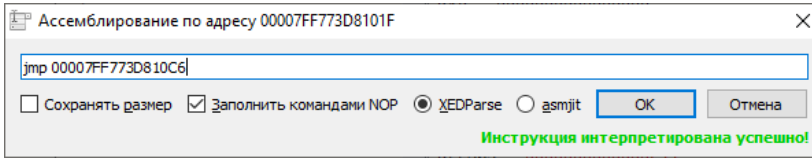


Рис. 6.8. Окно ассемблирования с первой командой jmp

Обращаем внимание на будущее, что команда jmp с адресом перехода имеет размер 5 байтов.

Следующим действием является перенос нового блока:

```
mov rax,30
mul r10
mov r11,rax
xor rdx,rdx
mov rax,qword ptr ds:[7FF758BF306C]
div qword ptr ds:[7FF758BF3074]
mov r12,20
mul r12
add rax,r11
```

в кода в конец кода.

Для перенесения одной строки кода выделяем эту строку.

Нажимаем на выделенной строке правую клавишу мыши и выбираем Копировать / Дизассемблированный код. Проще на необходимой строке нажать Пробел и комбинацию клавиш Ctrl + C.

Выделяем в конце кода свободную строку, нажимаем пробел и в окне ассемблирования вставляем скопированную строку из буфера комбинацией клавиш Ctrl + V.

Нажимаем на этом окне кнопку OK.

Выполняем эти действия для всех переносимых строк.

Последней командой должна быть команда jmp с адресом передачи управления на продолжение неизмененного кода на строку

```
mov qword ptr ds:[7FF773D8307C],rax
```

Для этого, копируем адрес этой строки: Копировать/ Адрес.

Выделяем для команды jmp строку, нажимаем клавишу пробела, вводим jmp и адрес передачи управления комбинацией клавиш Ctrl + V (рис. 6.9).

Рис. 6.9. Результат внедрения кода

Но при копировании кода из одной программы в другую были скопированы старые адреса переменных и в другой программе они уже не являются актуальными. Эти адреса не подсвечиваются желтым цветом. В большинстве случаев их и скопировать не удается – надо сразу править адреса на актуальные. Их необходимо скорректировать.

Выясняем по уравнению, что это адреса переменных C и D.

Для того, чтобы найти, например адрес переменной C, ищем вверху исходной программы строку занесения начала переменных. Первой

переменной секции data является переменная A, которая вызывается строкой кода

```
mov r10,qword ptr:[7FF69BF7305C]
```

Выделяем мышкой эту строку, затем нажимаем правую клавишу мышки и выбираем Перейти к дампу / Константа.

В открывшемся дампе памяти ищем переменную C и выделяем ее (рис. 6.10).

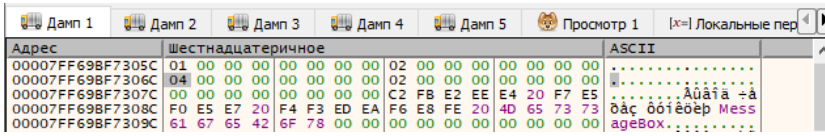


Рис. 6.10. Выделение переменной в дампе памяти

После выделения переменной сохраняем в буфере ее адрес. Для этого нажимаем правую клавишу мышки и выбираем Копировать / Адрес. Теперь меняем адрес на строке

```
mov rax,qword ptr ds:[7FF691B9306C]
```

адрес переменной C. Для этого выделяем строку, нажимаем Пробел и заменяем адрес.

Аналогично меняем адрес нахождения переменной D (рис. 6.11).

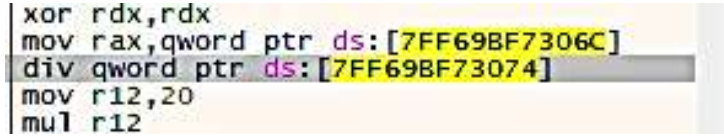


Рис. 6.11. Результат коррекции адресов переменных

Чтобы пропустить шаг коррекции адресов нахождения переменных лучше не использовать обращений к памяти, а константы записывать непосредственно в программе.

Сохраняем внесенные изменения.

Сохранение можно сделать двумя способами.

Первый способ: выбираем Файл/ Исправить файл. В новом окне выбрать пункт Исправить файл. Затем – новое имя и расширение exe. Для того, чтобы иметь возможность вернуться и загрузить наиболее

правильную исправленную версию необходимо сохранять изменения и давать каждый раз новые имена или индексы к ним для сохраненных exe-файлов.

Второй способ: выбрать пиктограмму с названием Исправления, нажать на кнопку Исправить файл, а далее новое имя и расширение exe.

После проведенных преобразований новый exe-файл выводит тоже сообщение, что и на рис. 6.1, но с другим значением времени выполнения блока кода.

Интересной представляется задача **изменения текстовой информации в окне вывода**.

Для этого необходимо изменить параметры функции вывода..

Для внесения изменений в упрощенное окно вывода информации можно использовать функцию преобразования wsprintf. Это позволит также внести изменения в вывод информации в самом окне.

При передаче параметров в функции wsprintf первым параметром является параметр ADDR st1, который передается функции через регистр RCX.

Второй параметр – это ADDR ifmt – строка форматирования, который передается через регистр RDX.

Третий параметр – Z – передается через регистр R8.

Четвертый параметр – X – передается через регистр R9.

Остальные параметры передаются через стек. Например, пятый – параметр Y – через [rsp + 20h]. Шестой – параметр rax – через [rsp + 28h] и т.д.

Можно внести изменения и в функцию MessageBox. Эта функция предназначена для вывода сообщений. В ней можно поменять и титул упрощенного окна, и непосредственное сообщение.

Применяют два способа изменения символов:

– посимвольное использование таблицы кодировки;

– изменение всей строки символов.

При посимвольном изменении в выводе информации необходимо воспользоваться таблицей кодировки символов страницы кодировки 1251, например http://foxtools.ru/ASCII#1251 (рис. 6.12).

Будем менять второй параметр функции wsprintf – это ADDR ifmt – строка форматирования, который передается через регистр RDX.

Сводная таблица кодов ASCII

ASCII таблица кодов символов Windows (Win-1251)

Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ	Dec	Hex	Символ
0	0	спец. NOP	32	20	спец. SP (Пробел)	64	40	@	96	60	`	128	80	Ъ	160	A0		192	C0	А	224	E0	а
1	1	спец. SOH	33	21	!	65	41	A	97	61	a	129	81	Ѣ	161	A1	Ў	193	C1	Б	225	E1	б
2	2	спец. STX	34	22	"	66	42	B	98	62	b	130	82	‚	162	A2	ў	194	C2	В	226	E2	в
3	3	спец. ETX	35	23	#	67	43	C	99	63	c	131	83	ѓ	163	A3	Ј	195	C3	Г	227	E3	г
4	4	спец. EOT	36	24	$	68	44	D	100	64	d	132	84	„	164	A4	¤	196	C4	Д	228	E4	д
5	5	спец. ENQ	37	25	%	69	45	E	101	65	e	133	85	…	165	A5	Ґ	197	C5	Е	229	E5	е
6	6	спец. ACK	38	26	&	70	46	F	102	66	f	134	86	†	166	A6	¦	198	C6	Ж	230	E6	ж
7	7	спец. BEL	39	27	'	71	47	G	103	67	g	135	87	‡	167	A7	§	199	C7	З	231	E7	з
8	8	спец. BS	40	28	(	72	48	H	104	68	h	136	88	€	168	A8	Ё	200	C8	И	232	E8	и
9	9	спец. Табуляция	41	29	)	73	49	I	105	69	i	137	89	‰	169	A9	©	201	C9	Й	233	E9	й
10	0A	спец. LF (Возвр. каретки)	42	2A	*	74	4A	J	106	6A	j	138	8A	Љ	170	AA	Є	202	CA	К	234	EA	к
11	0B	спец. VT	43	2B	+	75	4B	K	107	6B	k	139	8B	‹	171	AB	«	203	CB	Л	235	EB	л
12	0C	спец. FF	44	2C	,	76	4C	L	108	6C	l	140	8C	Њ	172	AC	¬	204	CC	М	236	EC	м
13	0D	спец. CR (Новая строка)	45	2D	-	77	4D	M	109	6D	m	141	8D	Ќ	173	AD	-	205	CD	Н	237	ED	н
14	0E	спец. SO	46	2E	.	78	4E	N	110	6E	n	142	8E	Ћ	174	AE	®	206	CE	О	238	EE	о
15	0F	спец. SI	47	2F	/	79	4F	O	111	6F	o	143	8F	Џ	175	AF	Ї	207	CF	П	239	EF	п
16	10	спец. DLE	48	30	0	80	50	P	112	70	p	144	90	ђ	176	B0	°	208	D0	Р	240	F0	р
17	11	спец. DC1	49	31	1	81	51	Q	113	71	q	145	91	'	177	B1	±	209	D1	С	241	F1	с
18	12	спец. DC2	50	32	2	82	52	R	114	72	r	146	92	'	178	B2	І	210	D2	Т	242	F2	т
19	13	спец. DC3	51	33	3	83	53	S	115	73	s	147	93	"	179	B3	і	211	D3	У	243	F3	у
20	14	спец. DC4	52	34	4	84	54	T	116	74	t	148	94	"	180	B4	ґ	212	D4	Ф	244	F4	ф
21	15	спец. NAK	53	35	5	85	55	U	117	75	u	149	95	•	181	B5	µ	213	D5	Х	245	F5	х
22	16	спец. SYN	54	36	6	86	56	V	118	76	v	150	96	–	182	B6	¶	214	D6	Ц	246	F6	ц
23	17	спец. ETB	55	37	7	87	57	W	119	77	w	151	97	—	183	B7	·	215	D7	Ч	247	F7	ч
24	18	спец. CAN	56	38	8	88	58	X	120	78	x	152	98	[illegible]	184	B8	ё	216	D8	Ш	248	F8	ш
25	19	спец. EM	57	39	9	89	59	Y	121	79	y	153	99	™	185	B9	№	217	D9	Щ	249	F9	щ
26	1A	спец. SUB	58	3A	:	90	5A	Z	122	7A	z	154	9A	љ	186	BA	є	218	DA	Ъ	250	FA	ъ
27	1B	спец. ESC	59	3B	;	91	5B	[	123	7B	{	155	9B	›	187	BB	»	219	DB	Ы	251	FB	ы
28	1C	спец. FS	60	3C	<	92	5C	\	124	7C	|	156	9C	њ	188	BC	ј	220	DC	Ь	252	FC	ь
29	1D	спец. GS	61	3D	=	93	5D	]	125	7D	}	157	9D	ќ	189	BD	Ѕ	221	DD	Э	253	FD	э
30	1E	спец. RS	62	3E	>	94	5E	^	126	7E	~	158	9E	ћ	190	BE	ѕ	222	DE	Ю	254	FE	ю
31	1F	спец. US	63	3F	?	95	5F	_	127	7F		159	9F	џ	191	BF	ї	223	DF	Я	255	FF	я

Рис. 6.12. Таблица кодировки символов 1251

Для этого, выделяем правой кнопкой мыши строку с передачей второго параметра

```
lea rdx,qword ptr ds:[7FF69BF730B3]
```

и выбираем Перейти к дампу/ Константа.

После того, как в дампе памяти будет отображаться нужное содержимое, необходимо подвести курсор к тем байтам дампа памяти, которые будем менять. Надо определиться: какую информацию в окне вывода мы собираемся менять. Если нас устроит только затереть первоначальный текст вывода информации, то применяем символ пробела. Он имеет кодировку 20h. Этой кодировкой можно затереть всю ненужную информацию.

При изменении всей строки символов каждая буква которого кодируется соответствующим символом, все слово, например, Инфицировано пишется как C8 ED F4 E8 F6 E8 F0 EE E2 E0 ED EE. Для того, чтобы изменить не один символ текста, а слово или предложение, необходимо в дампе памяти поставить курсор на байт начала сообщения, нажать на правую клавишу мышки и выбрать Двоичные операции/Редактировать и ввести новую текстовую информацию в строку с надписью ASCII (рис. 6.13).

Рис. 6.13. Окно редактирования сообщения

После ввода изменений необходимо пересохранить файл под новым именем, не забыв поставить расширение exe. Затем – выполнить файл и проверить результат. Если после ввода текста остались

неисправленные символы, то необходимо снова в отладчике в дампе памяти поставить мышку на начало неисправленных символов и повторить первоначальные действия.

Окно с окончательным результатом внедрения приведено на рис. 6.14.

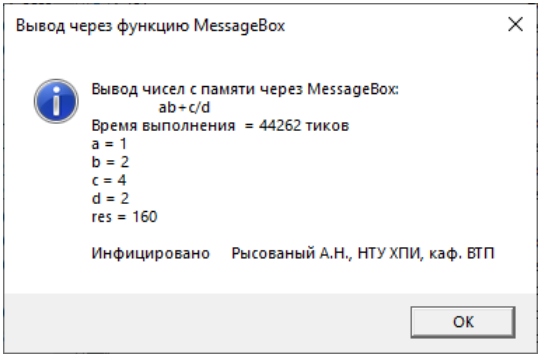


Рис. 6.14. Окно с окончательным результатом внедрения

Таким образом, после проведенного внедрения кода можно сделать выводы:

– при написании программы внедрения как можно меньше использовать переменные, а записывать константы непосредственно в программе;

– проще внедрять изменения в окончательную функцию вывода информации. В рассматриваемом примере – это функция MessageBox;

– после каждого законченного этапа внедрения обязательно делать сохранение exe-файла и каждый раз с новым именем или с индексом для его расширения.

6.3. Внедрение кода в диалоговую программу

Напишем диалоговую программу и осуществим в нее внедрение кода в виде слова «Инфицировано» в существующую функцию. Новую функцию вводить пока не будем.

Детальный процесс создания диалоговой программы рассмотрен в предыдущем разделе.

Этапы создания проекта:

1. Создание при помощи программы ресурсов, например ResEd, программы ресурсов;

2. Изменение пакетного файла с именем makeit.bat для создания программы с ресурсами;

3. Написание исходного кода программы пользователя.

Программа 6.3. Файл ресурсов rc:

```
#define IDM_1 10001
#define IDD_DLG1 1000
#define IDC_IMG1 1001
#define IDC_BTN1 1002
#define IDC_BTN2 1003
#define IDC_GRP1 1004
#define IDC_STC1 1005
#define IDC_STC2 1006
#define IDR_MENU1 10000
#define IDM_2 10002
#define IDM_3 10003

#include "C:/masm64/include64/Resource.h"

10 ICON DISCARDABLE "tongue.ico"
11 ICON DISCARDABLE "user.ico"

IDD_DLG1 DIALOGEX 10,10,372,168
CAPTION "masm64. Диал. окно с меню"
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
BEGIN
  CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,189,6,17
7,156
   CONTROL "Вычисление",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,51,30,87,21
   CONTROL "Выход",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,51,66,87,21
   CONTROL "Управление программой",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,42,15,111,87
   CONTROL "Кафедра ""Вычислительная техника и
программирование""", IDC_STC1,"Static",
WS_CHILDWINDOW|WS_VISIBLE, 21,135,126,18
   CONTROL "НТУ  ХПИ",IDC_STC2,"Static",
WS_CHILDWINDOW|WS_VISIBLE,147,156,36,9
END
```

Программа 6.4. Шаблон диалоговой программы:

```
        include \masm64\include64\masm64rt.inc
        .data?
          hInstance dq ?
          hIcon     dq ?
.code
entry_point proc

    mov hInstance,rv(GetModuleHandle,0)
 mov hIcon,rv(LoadImage,hInstance,10, IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)

        invoke DialogBoxParam,hInstance,1000,0,ADDR main,hIcon
        invoke ExitProcess,0
           ret
        entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,
lParam:QWORD
        .switch uMsg
        .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; значок для диалога
invoke SendMessage,rv(GetDlgItem,hWin,1001),\ ; значок в клиентской обл.
                    STM_SETIMAGE,IMAGE_ICON,lParam

        .case WM_COMMAND
          .switch wParam
            .case 1002
invoke MsgboxI,0,"Кафедра ВТП.  Шаблон программы", "Вычисление",
MB_OK,11
            .case 1003
invoke MsgboxI,0,"Вы подтверждаете выход из программы?", "Выход",
MB_OK,11
          .endsw

        .case WM_CLOSE
 exit1: invoke EndDialog,hWin,0 ;
         .endsw
           xor rax, rax
           ret
        main endp
        end
```

Результат выполнения программы представлено на рис. 6.15.

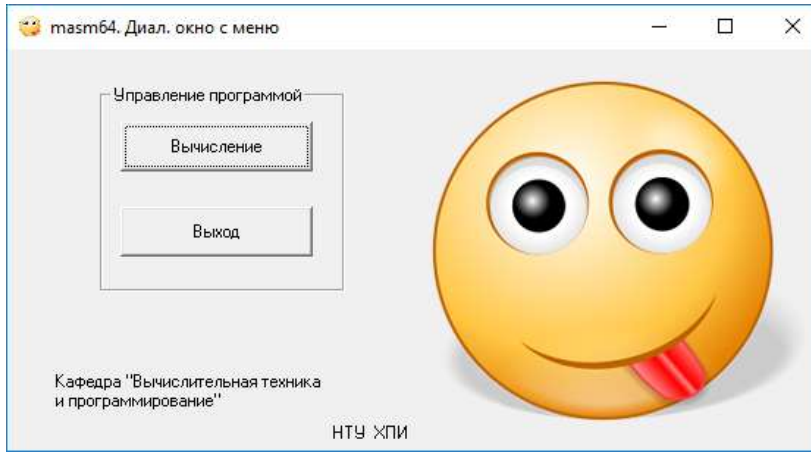


Рис. 6.15. Результат выполнения программы

Для осуществления внедрения в exe-файл необходимо получить информацию о файле:

– разрядность платформы программирования (x32 или x64);

– язык программирования;

– объем свободных ячеек памяти в секции кода.

Для определения разрядности используем одну из лучших программ анализаторов PE-файлов CFF Explorer (рис. 6.16).

Из анализа этого окна видно, что программа написана на платформе программирования x64. Из этого факта можно сделать вывод, что внедрение будет осуществляться через отладчик x64Dbg.

При выборе пункта меню Section Headers программы CFF Explorer открывается окно секций (рис. 6.17), в нижней части которого располагается 16-ричный редактор со значениями ячеек памяти каждой секции. **Для подсчета количества** байтов свободных ячеек памяти необходимо из адреса начала следующей секции вычесть первую свободную ячейку рассматриваемой секции:

$$400h - 23Ah = 1C6h = 454 \text{ байта.}$$

Объем секции кода (из-за того, что программа диалоговая и превышает 200h) равен 400h = 1024 байта. Следовательно, объем свободных ячеек чуть меньше половины объема всей секции кода. Этого

вполне достаточно для серьезного внедрения. По заданию необходимо внедрить слово «Инфицировано».

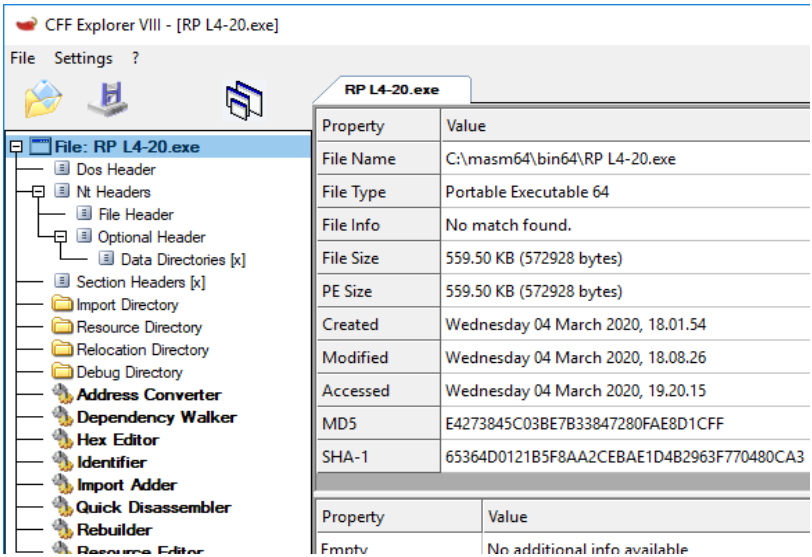


Рис. 6.16. Основное окно программы анализатора CFF Explorer

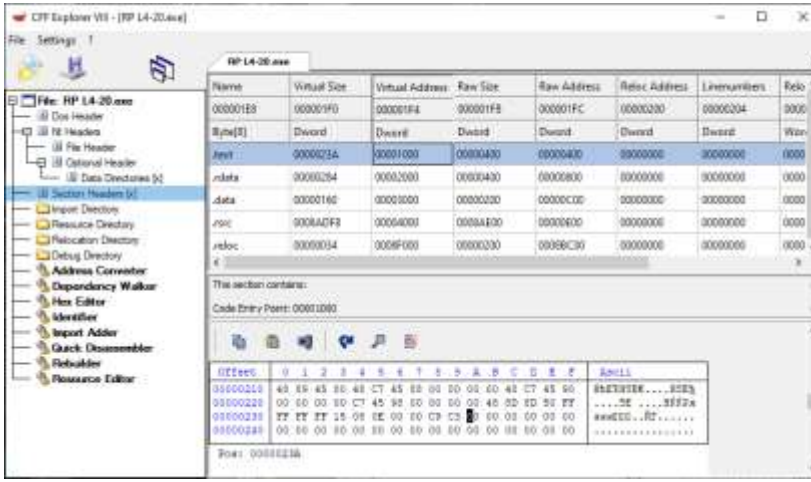


Рис. 6.17. Окно с характеристиками секций

Для внедрения кода откроем отладчик x64Dbg.

Необходимо найти место внедрения. Это могут быть как функции вывода, так и макросы, которые осуществляют вывод. Если применена API-функция, то в отладчике название этой функции прописано. Но а если это макрос вывода, то названия как правило нет, а если и будет, то оно почти не несет информации.

Анализируем все сообщения программы. Их два и они выводятся после нажатия на две клавиши диалогового окна. Будем искать эти сообщения (рис. 6.18).

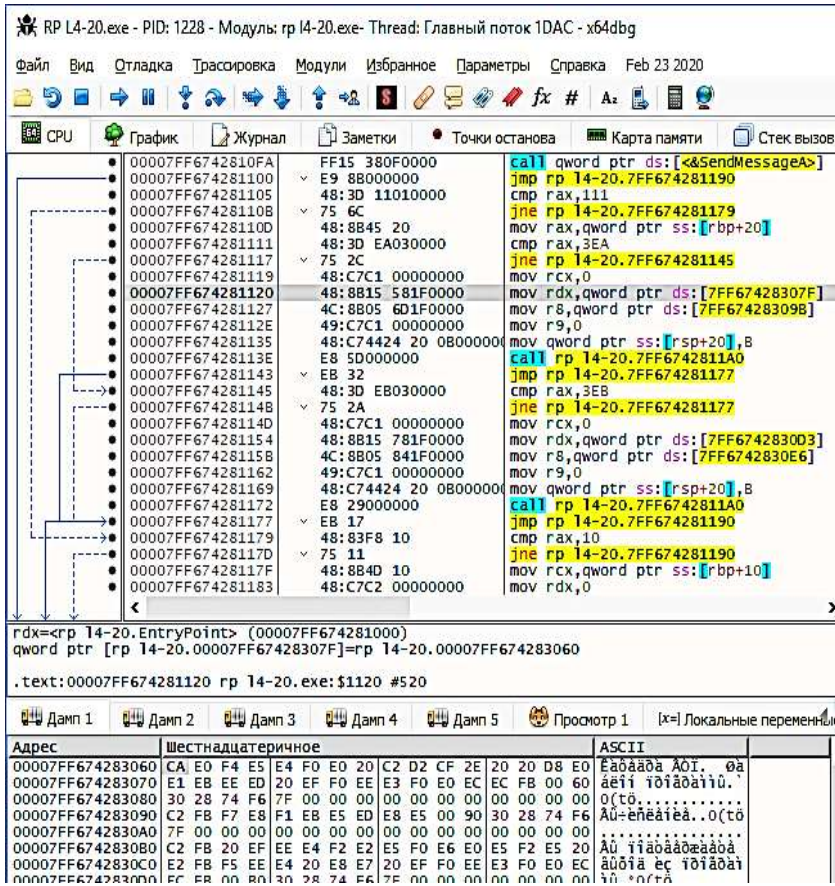


Рис. 6.18. Окно со свободными ячейками секции

При осмотре программы видно, что за вывод информации отвечают макросы с параметрами. Анализируем параметр rdx, в который записываются данные по адресу 00007FF674281120.

Ставим курсор на строку с регистром rdx, нажимаем правую клавишу мышки и переходим по Перейти к дампу/Значение.

После того, как в дампе памяти будет отображаться содержимое памяти, необходимо подвести курсор к тем байтам дампа памяти, которые будем менять. **Для того, чтобы изменить всю строку** необходимо в дампе памяти поставить курсор на байт начала сообщения, нажать на правую клавишу мышки и выбрать Двоичные операции/Редактирование и ввести новую текстовую информацию.

Обязательно вносим изменения путем выбора пиктограммы с названием Исправления, нажать на кнопку Исправить файл, а далее новое имя и расширение exe.

Запускаем новый exe-файл на выполнение с результатом внедрения (рис. 6.19).

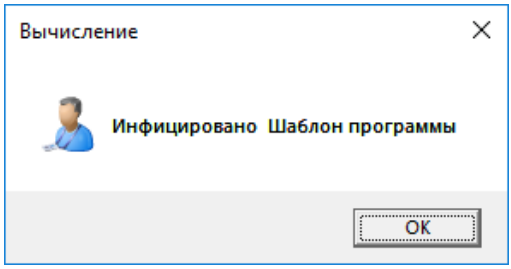


Рис. 6.19. Результат изменения программы

6.4. Лабораторная работа «Внедрение кода в exe-файл в конец секции без добавления новых API-функций»

Цель занятия: приобрести практические навыки исправления ошибок в программном обеспечении, которое находится в эксплуатации;

создание новой функциональности, используя неявное в эксплуатации программное обеспечение для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с комментариями;

– результат выполнения программы (скриншоты упрощенного окна (MessageBoxIndirect) и отладчика);

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Требования к интерфейсу пользователя

Создать диалоговое окно с ресурсами:

– собственной иконкой (иконками) на диалоговом окне. На иконках написать свою фамилию;

– кнопками;

– текстом на окне;

– статическими рамками и др.

Постановка задачи

1. Написать программу в среде masm64 в соответствии с первой частью варианта задания. Вывести время выполнения уравнения и результаты через функцию MessageBox. Программу и результаты привести в отчете.

2. Подсчитать свободное количество байтов для осуществления внедрения.

3. Исправить ранее написанную программу в отладчике x64Dbg в соответствии со второй частью варианта задания. А также изменить в первоначальном exe-файле выводимую информацию (в функции MessageBox). Привести в этой функции свою фамилию. Для этого использовать таблицу ASCII#1251 (например, http://foxtools.ru/ASCII#1251). При выполнении операций умножения и деления использовать логические команды. Вывести время выполнения уравнения. Скриншоты исправленной программы и выводимое окно привести в отчете.

Задание

В соответствии с номером студента в группе выбрать вариант задания и написать в среде masm64 на ассемблере программу вычисления одного из выражений:

1. 2*d*/4*c* ; → 4*ac* – 32*c*/*b;*
2. 4*abc*; → 8*a*/7 – *d*/32*b*;
3. 8*ab*/*c*; → *d*/4*a* + 64*d*/*c*;
4. 16(*d*/*b*)*c*; → 2*b* – *e*/4*b*;
5. 32*e*/3*c*; → 16*d*/4*a* – 4*cd*;
6. 64*d*/*ca*; → *e*/8*b* + 16*ac*;
7. 128*a*/7; → *d*/4*a* + 16*d*/*c;*
8. 18*c*/2*b*; → 32*ab*/2*a* – 4*c*;
9. *c*/16*b;* → 2*b* – *e*/8*b*;
10. 16*d*/4*c*; → 2*d*/*a* – 16*bc*;
11. 10*e*/*b*; → 4*d*/*b* + 8*e*;
12. 22*d*/*b*; → 8*d*/*b* – 32*d*/*c*;
13. 33*c*/*d;* → 2*ab* – 16*c*/*d*;
14. 21*cb*/3; → 16*d*/2*c* – 8*ab*;
15. 24*e*/4; → *d*/*b* – *d*/16*c*;
16. 32*e*/4; → 2*a*/b – *c*/16*e*;
17. 16*d*/4*a*; → 2*ab* – 16*c*/*b*;
18. 8*d*/*b*; → 4*e*/*b* + 32a*c*;
19. 3*d*/c*b*; → 8*d*/2*b* – 4*ac*;
20. 2*d*/*18a*; → 32*b* – *8ac*/*d;*
21. 19*d*/3*c*; → *cb*/2*a* – 8*d*;
22. 28*c*/4*b* → *e*/4*b* – *d*/8*c*;
23. *d*/8*c*; → *d*/4*c* – 32*ab*;
24. 2*d*/16*c*; → 32*d*/2*c* – 8*cd*;
25. 2*a*/17*e*; → 32*a*/8*c* – *d*/2*e*;
26. 8*a*/16*d*; → 4*d*/*a* – 16*bd*;

где *a, b, c, d ,e, f* – числа.

В каждом варианте 2 уравнения: для первоначального вычисления и для дальнейшего исправления.

Для выполнения задания используйте материал, который рассмотрен раньше.

Программа 6.5. Сравнение времени выполнения арифметических операций и операций сдвига (без реализации интерфейса окна):

```
include /masm64/include64/masm64rt.inc

calc_ar proto p_a0: DQ, p_b0: DQ,p_c0: DQ, p_d0: DQ,p_e0: DQ,p_f0: DQ
calc_sh proto p_a0 DQ,p_b0: DQ,p_c0: DQ,p_d0: DQ,p_e0: DQ,p_f0: DQ

.const
    ; d/b/a – cd/e/f
    ; (128/4/2) - (1*128/8/2)
    ; 16 - 8 = 8
    ; параметры для арифметических операций
    a0 dq 2
    b0 dq 4
    c0 dq 1
    d0 dq 128
```

```
        e0 dq 8
        f0 dq 2
; параметры для операций сдвига
        a1 dq 1
        b1 dq 2
        c1 dq 0
        d1 dq 7
        e1 dq 3
        f1 dq 1

title0 db "Сравнение операций", 0
fmt0 db "http://blogs.kpi.kharkov.ua/v2/asm",10,10,
"Уравнение: d/b/a - cd/e/f",10,10,"С арифметическими операциями:",10,
          "  - Результат: %d", 10,"  - Кол. тиков: %d", 10,10,
          "С операциями сдвига:", 10, "  - Результат: %d", 10,
          "  - Кол. тиков: %d", 10, 0
.data
      res0 dq ? ;
      res1 dq ? ;
      time0 dq ? ;
      time1 dq ? ;
      text0 db 300 dup(0) ;

.code
entry_point proc
            invoke calc_ar,a0,b0,c0,d0,e0,f0
            invoke calc_sh,a1,b1,c1,d1,e1,f1
        invoke wsprintf,addr text0,addr fmt0,res0,time0,res1,time1
invoke MessageBox,0,addr text0,addr title0,MB_ICONINFORMATION
            invoke ExitProcess,0
        entry_point endp

calc_ar proc p_a0: DQ,p_b0:DQ,p_c0:DQ,p_d0:DQ, p_e0:DQ, p_f0: DQ
            mov r10,rcx ; a
            mov r11,rdx ; b

            rdtsc
            xchg rdi,rax ;

        ; d/b/a – cd/e/f
            mov rax,r9 ; d -> rax
            xor rdx,rdx
            idiv r11     ; d/b -> rax
            xor rdx,rdx
            idiv r10     ; d/b/a -> rax
```

```
        mov res0,rax ; d/b/a -> res0

        mov rax,r8  ; c -> rax
        imul r9     ; cd -> rax
        xor rdx,rdx
        idiv p_e0   ; cd/e -> rax
        xor rdx,rdx
        idiv p_f0   ; cd/e/f -> rax
        sub res0,rax ; d/b/a - cd/e/f

        rdtsc
        sub rax,rdi
        mov time0,rax
        ret
    calc_ar endp

calc_sh proc p_a0:DQ,p_b0: DQ, p_c0: DQ, p_d0: DQ, p_e0: DQ, p_f0:DQ
        mov r10, rcx ; r10 - a
        mov r11, rdx ; r11 - b

        rdtsc
        xchg rdi, rax

    ; d/b/a – cd/e/f
        mov rax,1
        mov rcx,r9
        shl rax,cl   ; d -> rax

        mov rcx,r11
        shr rax,cl   ; d/b -> rax
        mov rcx,r10
        shr rax,cl   ; d/b/a -> rax

        mov rbx,1
        mov rcx,r8
        shl rbx,cl   ; c -> rbx

        mov rcx,r9
        shl rbx,cl   ; cd -> rbx
        mov rcx,p_e0
        shr rbx,cl   ; cd/e -> rbx
        mov rcx,p_f0
        shr rbx,cl   ; cd/e/f -> rbx

        sub rax,rbx  ; d/b/a – cd/e/f -> rax
```

```
    mov res1,rax

    rdtsc
    sub rax, rdi
    mov time1, rax
    ret
calc_sh endp
end
```

Результат выполнения программы представлен на рис. 6.20.

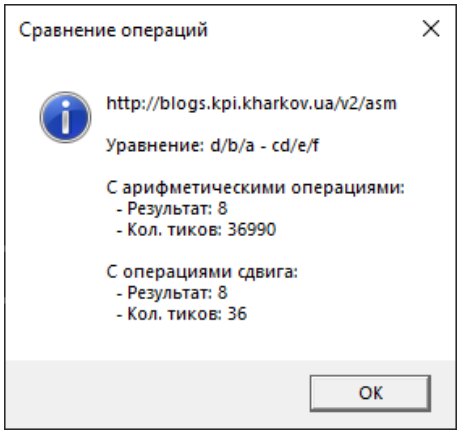


Рис. 6.20. Результат выполнения программы

Число тиков определяется при помощи команды rdtsc. Однако можно было и использовать команду rdtscp (Intel Nehalem, 2008).

7. ВНЕДРЕНИЕ КОДА В EXE-ФАЙЛ В КОНЕЦ СЕКЦИИ КОДА С ДОБАВЛЕНИЕМ НОВЫХ API-ФУНКЦИЙ

7.1. Общие сведения

Внедрение новых API-функций в чужой код почти не отличается от внедрения команд. При внедрении API-функций необходимо не только знать количество и назначение входящих в них параметров, но и как эти параметры отображаются в окне отладчика. Для этих целей в зависимости от задачи, проще всего, написать отдельную программу с использованием необходимых API-функций. Затем откомпилировать такую программу, а потом открыть ее exe-файл в отладчике и проанализировать их использование. После чего можно приступать и к внедрению своей программы в чужой код. В учебных целях в качестве чужой программы тоже будет применяться написанная нами программа.

При использовании отладчика x64Dbg нет необходимости вызывать, затем анализировать таблицу импорта адресов внедряемых функций, и вставлять эти адреса вместо названий функций, что сильно упрощает алгоритм внедрения.

Алгоритм внедрения API-функций рассмотрен в предыдущем разделе.

Перед началом непосредственного внедрения исследуемой файл проверяется программой-анализатором PE-файлов на предмет определения языка программирования (или компилятора), разрядности архитектуры (32/64), упаковщика или криптора.

После чего определяется объем свободных байтов в секции кода.

Последний и самый основной этап внедрения выполняется в отладчике.

7.2. Внедрение кода с добавлением API-функций

Предположим, что существует программа, написанная в среде masm64 с использованием команд SSE, которая вычисляет математическое уравнение. В эту программу необходимо внедрить свое сообщение функцией MessageBox без изменения функционала. Внедрение кода с изменением функционала было выполнено в предыдущем разделе.

Рассмотрим основную программу (программа 7.1), в которую внедрим функцию MessageBox с соответствующим сообщением.

Программа 7.1. Числа {*a,b,c,d*} заданные массивом и имеют размерность Real8. Вычислить уравнение *d/b* sqrt(*a*) + *a:*

```
  ; masm64. Числа {a,b,c,d} заданные массивом
  ; и имеют размерность Real8. Вычислить уравнение d/b sqrt(a) + a

      include \masm64\include64\masm64rt.inc ; подключаемые библиотеки

.data
  mas1 real8 16.,2.,4.,16.   ; a, b, c, d
  tit1 db "masm64. Результат вычисления уравнения на SSE2",0
  buf1 dq ?,0  ; буфер вывода сообщения
  ifmt db "masm64.  Массив a, b, c, d  := 16., 2., 4., 16.",10,10,
  "Уравнение d/b sqrt(a) + a.    Результат: %d ",10,10,
  "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
  9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code                              ; уравнение d/b sqrt(a) + a
 entry_point proc ;

      movsd xmm1,mas1[0]   ; xmm1 - a - переслать двойное слово
      movsd xmm2,mas1[8]   ; xmm2 - b
      movsd xmm3,mas1[16]  ; xmm3 - c
      movsd xmm4,mas1[24]  ; xmm3 - d
      divsd xmm4,xmm2      ; d/b
      sqrtsd xmm5,xmm1     ; sqrt(a)
      mulsd xmm4,xmm5      ; d/b x sqrt(a)
      addsd xmm4,xmm1      ; d/b x sqrt(a) + a
      cvttsd2si eax,xmm4 ;
      movsxd r15,eax

invoke wsprintf,addr buf1,addr ifmt,r15
invoke MessageBox,0,addr buf1,addr tit1,MB_ICONINFORMATION

invoke ExitProcess,0;возвращение упр. ОС и освобожд. ресурсов
      entry_point endp
      end
```

Результатом выполнения программы является окно сообщения (рис. 7.1).

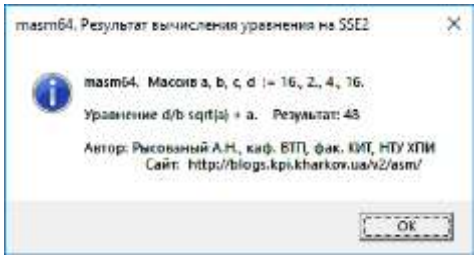


Рис. 7.1. Окно с результатом выполнения основной программы

Определим объем в байтах свободного места в секции кода. Для этой цели воспользуемся программой-анализатором LotdPE. Выполним последовательность действий:

– в программе нажать на клавишу меню PE Editor и выбрать исследуемый exe-файл;

– в появившемся окне нажать клавишу меню Sections;

– выбрать секцию, где расположен текст программы (это секция text), нажать правую клавишу мышки и выбрать пункт меню hex edit section;

– в появившемся окне memory buffer выбранная секция выделится темным цветом (рис. 7.2). В конце секции свободные ячейки будут заполнены нулями.

Для подсчета количества байтов свободных ячеек памяти в секции кода необходимо из начального адреса следующей секции вычесть первую свободную ячейку после окончания кода:

600 h – 477 h = 17D h = 381 байтов.

Этого количества свободных ячеек памяти вполне достаточно для внедрения блока кода с небольшим количеством функций.

Для внедрения функции с ее параметрами необходимо точно знать, как эти параметры отображаются синтаксически в отладчике и применять их точно так же. В основной программе уже присутствует функция MessageBox. Поэтому, писать в качестве примера отдельную программу с этой функцией уже нет необходимости: будем использовать параметры этой функции в exe-файле, анализировать и изменять их под собственные адреса ячеек памяти.

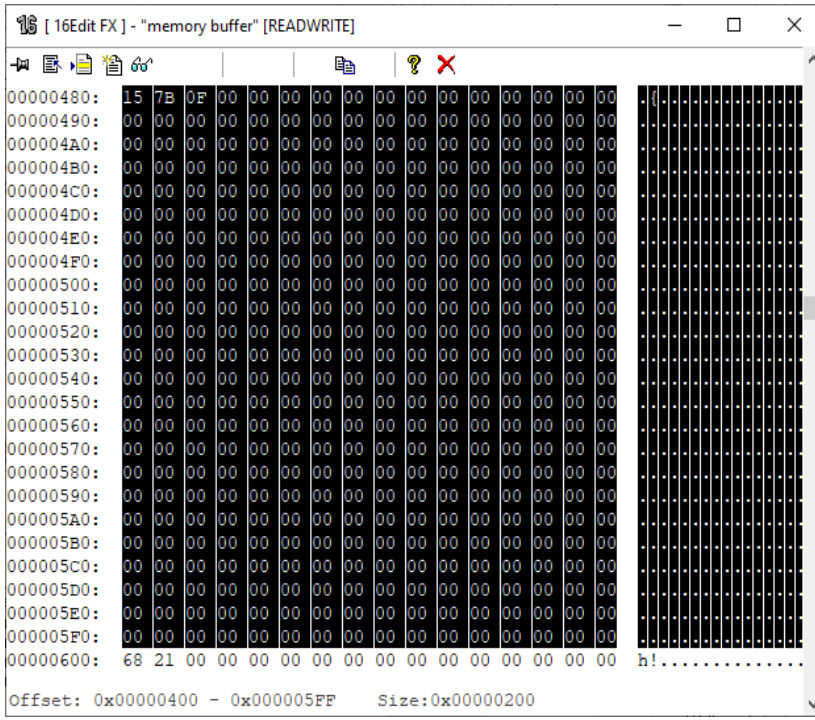


Рис. 7.2. Окно hex редактора секции кода

Открываем exe-файл этой программы в отладчике x64Dbg. При анализе кода в отладчике x64Dbg сначала необходимо продумать свои действия: где удобнее поставить новую функцию MessageBox и в какой последовательности она должна вывести окно сообщения.

Внешний вид основной программы представлен на рис. 7.3.

Проанализируем программу в отладчике.

В нашем самом простом случае необходимо осуществить внедрение функции с ее параметрами в существующий код. Для этого надо в программе освободить место. Внедрить новую функцию MessageBox можно в начале программы, в середине или уже в самом конце. Алгоритм внедрения при выборе любого места не поменяется. Поменяется только последовательность вывода сообщений программой.

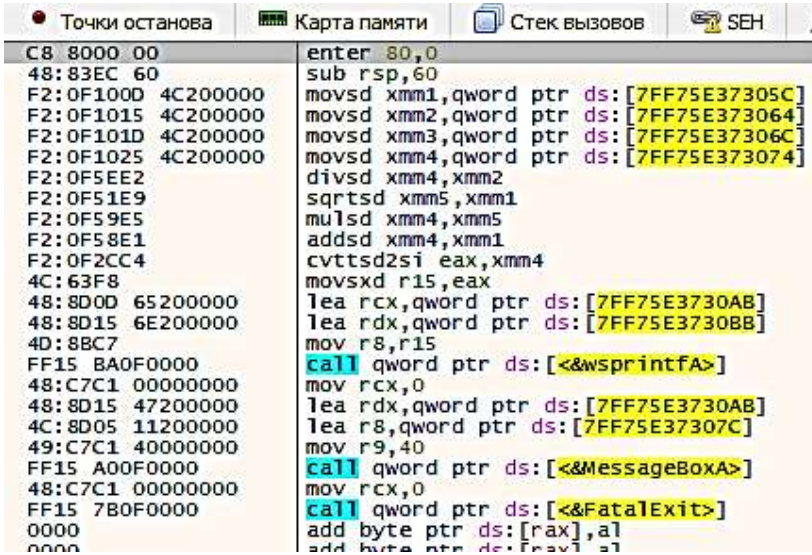


Рис. 7.3. Внешний вид основной программы

Выбираем для внедрения кода начало программы. Первые две команды удалять нельзя, т.к они подготавливают программу к выполнению – выравнивают стек. Их можно перенести в конец кода, а на их место поставить команду jmp с адресом перехода.

Однако, в качестве примера, изменим третью строчку. Для этого перенесем команду из третьей строчки на свободное место после окончания программы. Последней строкой программы является функция ExitProcess.

Затем на освободившееся место ставим команду jmp с адресом перехода на адрес расположения перенесенной команды.

После – вставляем параметры функции MessageBox и вызов (команда call) этой функции.

Последней командой после вызова функции MessageBox снова будет команда jmp, но с адресом возврата на нетронутую строчку кода после вставленной первой команды jmp в начале программы.

И так, команда jmp с адресом перехода имеет размерность 2 байта.

Перенесем третью строчку кода и тем самым освободим место под команду jmp. В третьей строчке программы находится команда, которая занимает 8 байт памяти:

```
00007FF75E371085 F2:0F100D      movsd xmm1,qword ptr
                 CF1F0000       ds:[7FF75E37305C]
```

Поэтому, перенесем эту команду в конец файла, а на ее место – команду jmp с адресом, который соответствует новому месту расположения после окончания программы. А оставшиеся байты заполним автоматически командами NOP.

После команды jmp вставляем функцию MessageBox, скопировав эту функцию в нашей же программе со старыми значениями параметров (теми же адресами ячеек):

```
mov rcx,0
lea rdx,qword ptr ds:[7FF75E3730AB]
lea r8,qword ptr ds:[7FF75E37307C]
mov r9,40
call qword ptr ds:[<&MessageBoxA>]
```

А позже уже заменим адреса на те, которые соответствуют новому месту расположения других параметров.

Для продолжения этой программы поставим команду возврата на не измененную строчку кода, которая расположена после вставленной в начале программы команды jmp.

Последней вставкой по свободным адресам расположим текстовую информацию для функции MessageBox. И эти адреса поставим в новой функции MessageBox.

Итак, приступаем к внедрению. Рассмотрим последовательность внедрения.

1. Освобождаем место для команды jmp. Для этого ставим курсор мышки на строку с первой командой программы, которую будем переносить на место, которое будет располагаться после конца программы. Это команда:

```
movsd xmm1,qword ptr ds:[7FF75E37305C]
```

Копируем строку кода. Для этого нажимаем правую клавишу мышки и выбираем Копировать/ Дизассемблированный код.

Копирование быстрее выполнить, если нажать на клавишу Пробел и нажать комбинацию клавиш Ctrl + C.

2. Переносим скопированную строку кода в конец кода. Для этого ставим курсор на первую позицию после самой последней команды программы. Нажимаем на левую клавишу мышки и вставляем (Ctrl + V) из буфера команду.

Скопированную команду вставлено в конец программы, но в начале кода (третья строка) эта команда еще не удалена.

3. Теперь необходимо заменить в начале программы команду, которую в предыдущем пункте уже переместили в конец файла, на команду jmp.

Для этого сначала копируем адрес перемещенной команды. Для этого ставим курсор в начало программы на команду, которую уже скопировали, но не заменили. Нажимаем правую клавишу мышки и выбираем Копировать / Адрес. После копирования адреса подводим курсор мышки в начало программы к команде, которую необходимо выделить, а затем и заменить. Нажимаем кнопку Пробел и появляется окно ассемблирования.

В всплывающем окне пишем команду jmp, а затем Ctrl + V для вставки адреса из буфера (рис. 7.4).

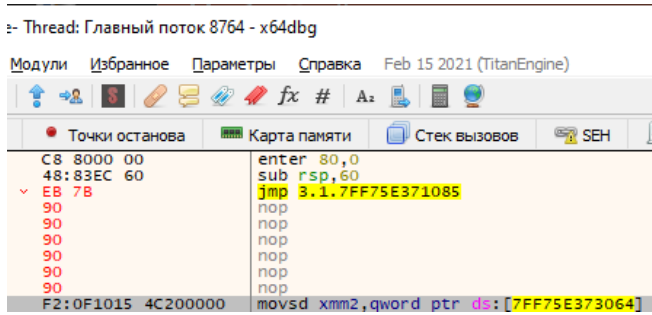


Рис. 7.4. Всплывающее окно с командой jmp

Не забываем после каждого действия сохранять внесенные изменения с заменой имени файла на другое. Для этого выбираем пиктограмму с названием Исправления, а далее новое имя и расширение exe.

4. Теперь вставляем параметры и вызываем саму функцию MessageBox из нашей же программы. В последующем – внесем коррективы.

Необходимо последовательно после первой перенесенной команды вставить строки из нашей же программы:

```
mov rcx,0
lea rdx,qword ptr ds:[7FF75E3730AB]
lea r8,qword ptr ds:[7FF75E37307C]
mov r9,40
call qword ptr ds:[<&MessageBoxA>]
```

Для этого ставим курсор мышки на первый параметр функции MessageBox. Это строка кода:

```
xor ecx,ecx.
```

Нажимаем правую клавишу мышки и выбираем Копировать / Дизассемблерный код, но быстрее если нажать на клавишу Пробел и сочетание клавиш Ctrl + C.

Подводим курсор мышки в конец файла на свободную строку. Нажимаем пробел и нажимаем Ctrl + V для вставки содержимого из буфера. После нажатия кнопки OK новая команда вставится на новое место.

Таким же образом переносятся оставшиеся строки параметров функции MessageBox.

Во вставленных строках параметров функции стоят старые адреса, которые необходимо заменить на новые. Но для того, чтобы не было фрагментации кода, с этими действиями необходимо пока повременить – надо еще вставить команду возврата jmp, а потом уже заменить адреса на реальные.

5. Вставляем команду возврата в конце вставленного кода с адресом возврата на начало оставшегося кода программы.

Для этого сначала подводим курсор к команде

```
movsd xmm2,qword ptr ds:[7FF75E373064]
```

и копируем адрес ее расположения: Копировать / Адрес.

После чего подводим курсор к следующей свободной в конце файла строке, нажимаем Пробел. Пишем jmp, вставляем пропуск и нажимаем комбинацию Ctrl + V (рис. 7.5).

6. Меняем адреса параметров функции MessageBox.

В этой функции в качестве параметров выступают текстовые строки с названиями титула упрощенного окна и строки вывода сообщения.

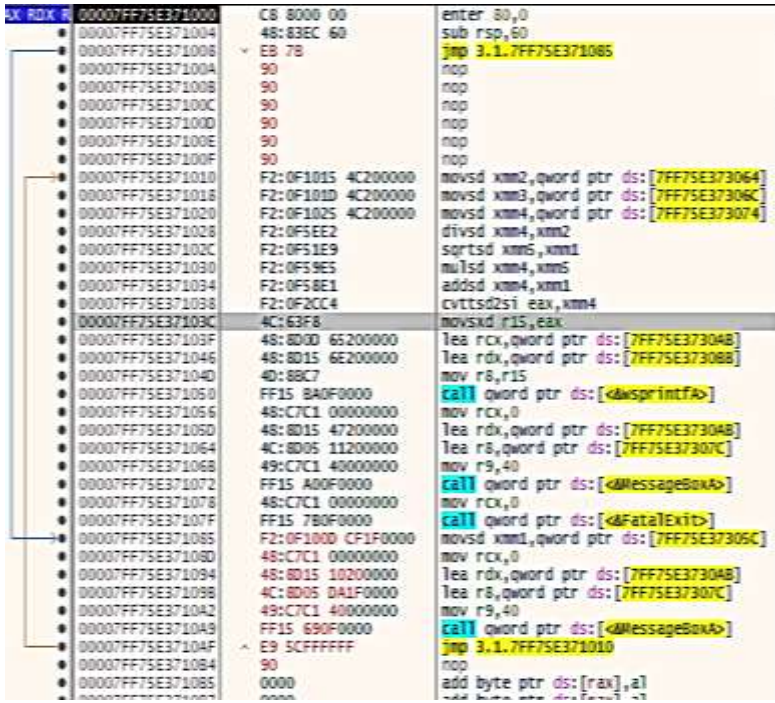


Рис. 7.5. Программа после вставки команды возврата

Если указать для каждого из параметров один адрес, то эти строки будут совпадать. В регистре rdx передается сообщение, а название титула упрощенного окна – через регистр r8.

Напишем слово «Инфицировано» и в эти регистры передадим для простоты один и тот же адрес. Скопируем адрес свободной строки и заменим эти адреса в регистрах rdx и r8. Для этого подводим курсор мышки к свободной строке, нажимаем правую клавишу и выбираем Копировать / Адрес.

После занесения в буфер адреса последовательно вызываем строки

lea rdx, …

lea r8, …

и меняем из буфера адрес на адрес свободной строки, куда будем писать сообщение.

7. Вставляем в ячейки памяти текст сообщения.

Для этого выделяем строку кода, адрес которой записан в параметры функции и выбираем Перейти к дампу / Константа. После чего в дампе памяти отобразится содержимое этих ячеек.

Подводим курсор к содержимому первого байта и нажимаем левую клавишу мышки. Нажимаем правую клавишу мышки и выбираем Двоичные операции /Редактировать. Вводим сообщение (рис. 7.6).

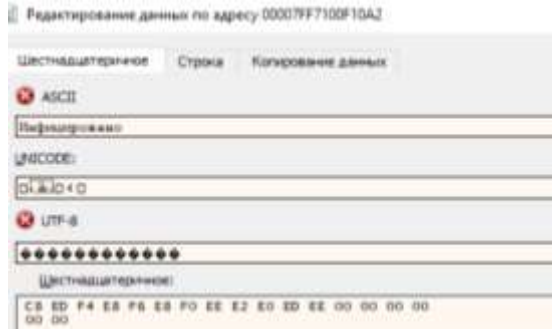


Рис. 7.6. Окно редактирования ячеек памяти

8. Делаем последнее сохранение изменений с заменой имени файла на другое. Для этого выбираем пиктограмму с названием Исправления, а далее новое имя и расширение exe.

9. Запускаем exe-файл. После запуска последовательно выводятся 2 сообщения (рис. 7.7).

Рис. 7.7. Окна с полученными результатами внедрения

Процесс внедрения завершен успешно.

7.3. Внедрение кода в диалоговую программу

Рассмотрим пример внедрения в полноценную диалоговую программу. Незначительной особенностью программы можно считать лишь некоторую сложность в поиске места функции, которая выводит сообщение, с которым предстоит производить необходимое действие.

Внедрим новую API-функцию в существующий код. Этапы создания проекта аналогичны рассмотренным в предыдущем разделе:

Этапы создания проекта:

1). Создание при помощи программы ресурсов, например ResEd, программы ресурсов;

2). Изменение пакетного файла с именем makeit.bat для создания программы с ресурсами;

3). Написание исходного кода программы пользователя.

Программа 7.2. Файл ресурсов rc:

```
#define IDD_DLG1 1000
#define IDC_IMG1 1001
#define IDC_BTN1 1002
#define IDC_BTN2 1003
#define IDC_STC1 1004
#define IDC_GRP1 1005
#include "C:/masm64/include64/Resource.h"
IDD_DLG1 DIALOGEX 10,10,342,165
CAPTION "masm64. Диалоговое окно с дочерними элементами"
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
BEGIN
  CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,9,0,174,
165
  CONTROL "Выполнить",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,207,42,111,18
  CONTROL "Выход",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,207,81,111,18
  CONTROL "Кафедра ""Вычислительная техника и
программирование"", НТУ ХПИ",IDC_STC1,"Static",
WS_CHILDWINDOW|WS_VISIBLE,189,138,123,18
  CONTROL "Управление",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,195,18,135,102
END

10 ICON DISCARDABLE "Smile.ico"
11 ICON DISCARDABLE "Computer.ico"
```

Программа 7.3. Шаблон диалоговой программы:

```
      include \masm64\include64\masm64rt.inc
.data?
        hInstance dq ?
        hIcon     dq ?
.data
      szTitl1 db "masm64. Исследование",0
      szInf1 db "Подтвердите выполнение",0

.code
    entry_point proc
       mov hInstance,rv(GetModuleHandle,0)
mov hIcon,rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)
       invoke DialogBoxParam,hInstance,1000,0,ADDR main,hIcon
      invoke ExitProcess,0
         ret
      entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
      .switch uMsg
      .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; значок для диалога
invoke SendMessage,rv(GetDlgItem,hWin,1001),\ ; ico в клиентской обл.
              STM_SETIMAGE,IMAGE_ICON,lParam
      .case WM_COMMAND
        .switch wParam

         .case 1002
invoke MessageBox,0,ADDR szInf1,ADDR szTitl1,MB_ICONINFORMATION
        .case 1003
invoke MsgboxI,0,"Вы подтверждаете выход из программы","Выход",
MB_OK,11
          jmp exit1
        .endsw
      .case WM_CLOSE
       exit1: invoke EndDialog,hWin,0 ;
       .endsw
         xor rax, rax
         ret
      main endp
      end
```

Результат выполнения программы представлено на рис. 7.8.

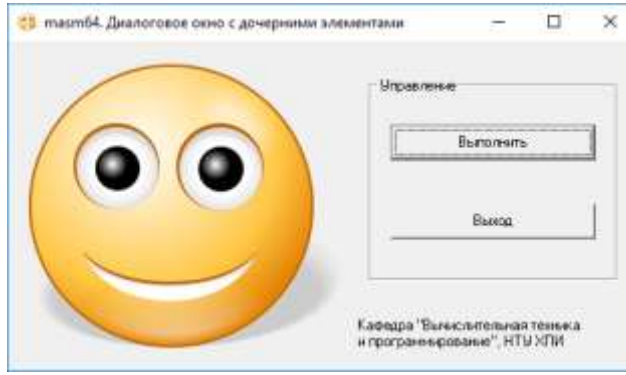


Рис. 7.8. Результат выполнения программы

Для осуществления внедрения в exe-файл необходимо получить информацию о файле:

– разрядность платформы программирования (x32 или x64);

– язык программирования;

– объем свободных ячеек памяти в секции кода.

Для определения разрядности используем одну из лучших программ CFF Explorer (рис. 7.9).

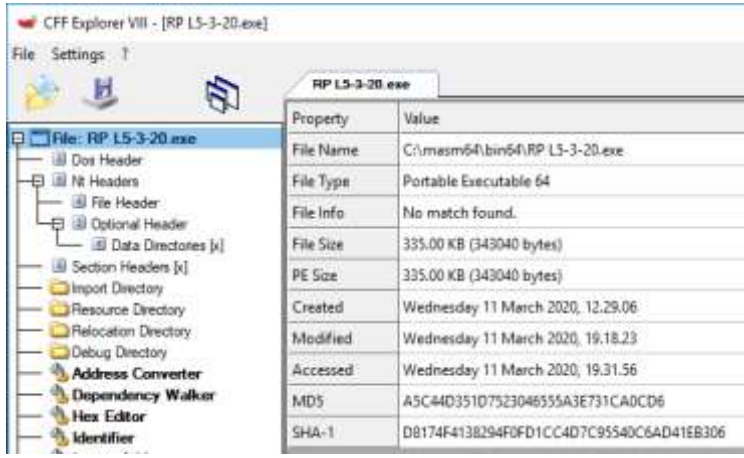


Рис. 7.9. Основное окно программы анализатора CFF Explorer

Из анализа этого окна видно, что программа написана на платформе программирования x64. Из этого факта можно сделать вывод, что внедрение будет осуществляться через отладчик x64Dbg.

Объем свободного места можно определить в программе LordPE.

Для подсчета количества байтов свободных ячеек памяти необходимо из адреса последней свободной ячейки вычесть последнюю занятую ячейку после окончания кода (рис. 7.10):

7FF h – 629 h = 1D6h = 470 байтов.

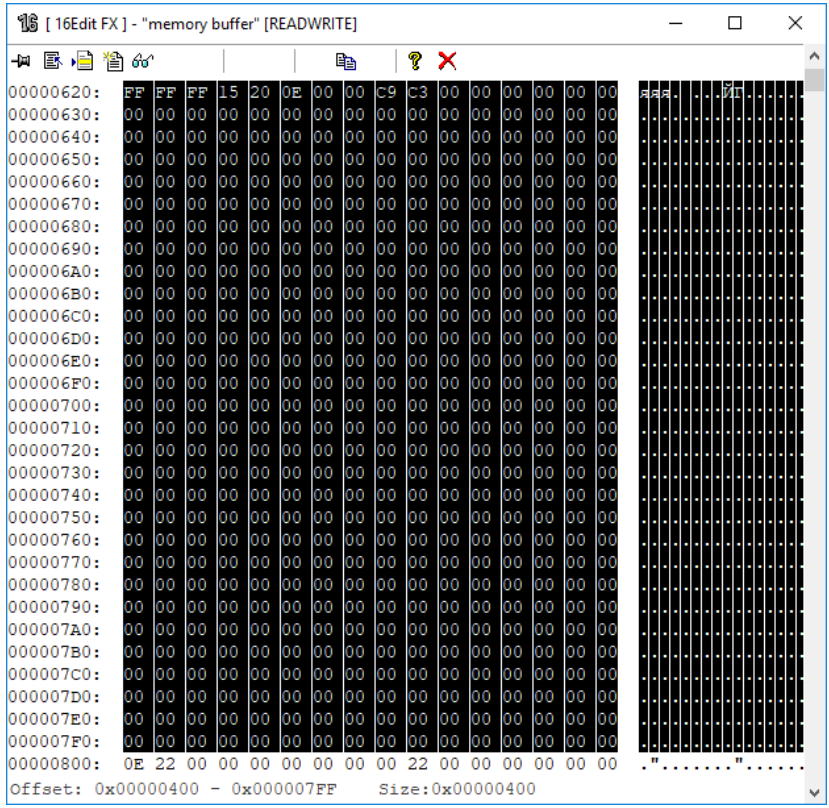


Рис. 7.10. Окно со свободными ячейками секции

Действия по внедрению новой API-функции в exe-файл следующие:

1. Открываем в отладчике x64Dbg ранее полученный exe-файл: Файл/ Открыть.

2. Выполняем программу, нажимая кнопку F9.

3. Определяем место внедрения.

Для нахождения места внедрения необходимо иметь некоторые представления о шаблоне окна и о тех сообщениях, которые выводятся анализированной программой. В программе две кнопки и при нажатии на каждую выводятся сообщения.

После нахождения места внедрения (рис. 7.11) осуществляем этот процесс.

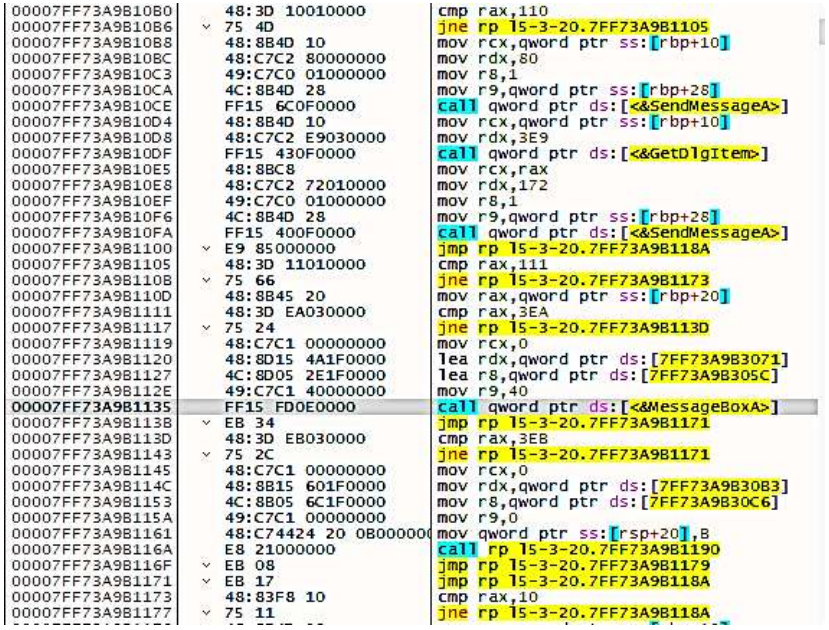


Рис. 7.11. Окно отладчика с местом внедрения

4. Освобождаем место для команды jmp на новый блок кода.

Определились, что команда jmp будет располагаться на месте команды mov rcx,0.

Переносим эту команду на свободное место. Для этого выделяем строку с этой командой, нажимаем правую кнопку мыши и выбираем Копировать/Дизассемблированный код или CTRL+C.

Переходим на конец кода на свободные ячейки.

Команда mov rcx,0 относится к первоначальному коду и ее необходимо или записать в блокнот, или вставить на новое место, оставив достаточное количество байтов для внедрения. А после окончательного ввода нового кода эта команда должна быть предпоследней – перед командой возврата jmp на первоначальный код.

Поэтому пропустив пару десятков свободных ячеек вставляем команду mov rcx,0. Для этого дважды нажимаем правой клавишей мыши на свободной строке (рис. 7.12) и нажимаем CTRL+V (вставляем из буфера).

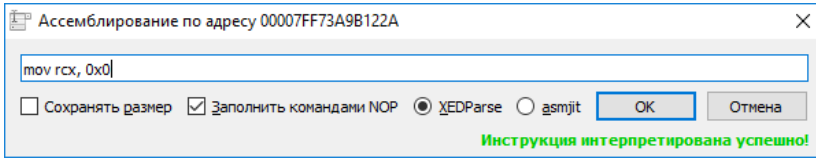


Рис. 7.12. Окно изменения кода

5. Запоминаем адрес свободной строки в конце кода, с которой будет вставляться новый блок (чтобы поставить jmp на этот адрес). Для этого нажимаем на ней правую кнопку мыши и выбираем Копировать/ Адрес.

6. Переходим вверх по коду – на место команды mov rcx,0. Дважды нажимаем на выбранную строку, вводим мнемонику jmp и вставляем адрес из буфера (CTRL+V) (рис. 7.13).

7. Вставляем в конец кода по адресу перехода на новый блок параметры вставляемой новой API-функции.

Для этого первым параметром на начале нового блока записываем команду mov rcx,0.

Остальные параметры и непосредственный вызов MessageBox копируем из исследуемой программы посредством сочетания клавиш CTRL+C и CTRL+V.

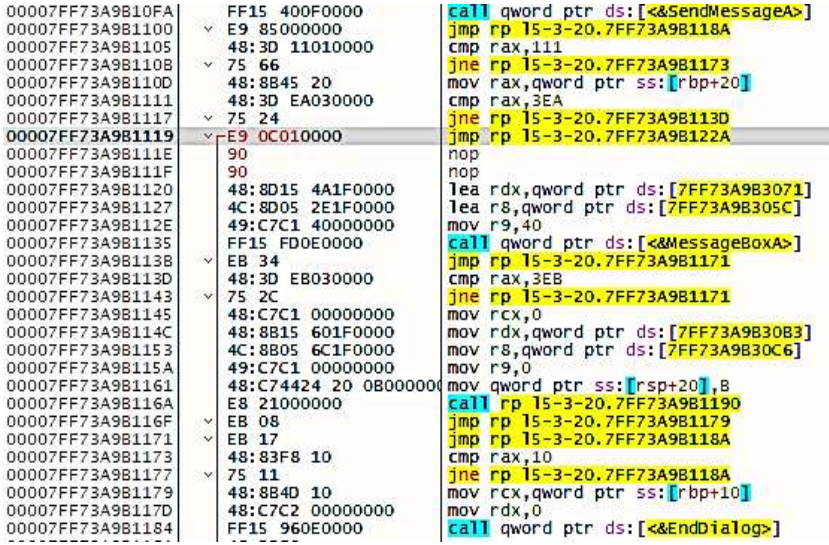


Рис. 7.13. Окно с изменения кода

8. Переносим из конца кода команду mov rcx,0.

9. Записываем последней командой нового блока команду jmp на продолжение программного кода (рис. 7.14).

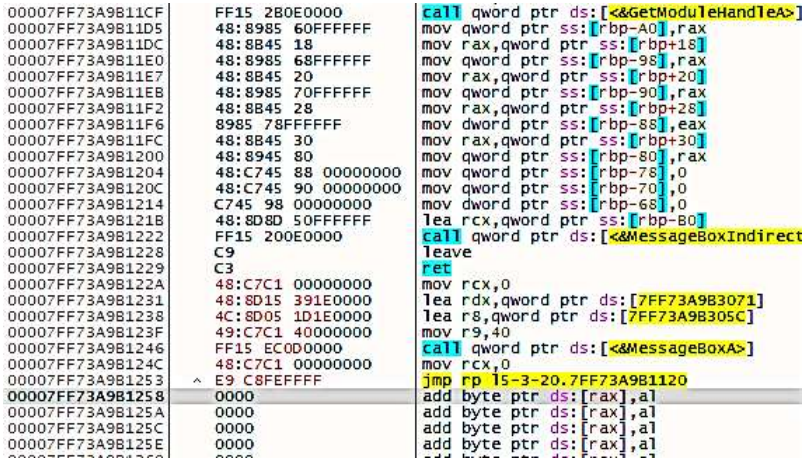


Рис. 7.14. Окно с последними изменениями кода внедрения

10. Изменяем параметры функции MessageBox.

Для этого выделяем свободную строку кода, нажимаем правую кнопку мыши и выбираем Перейти к дампу/ Выделенный адрес.

В **дампе памяти** ставим курсор на первой свободной ячейке и выбираем меню Двоичные операции / Редактировать. В появившемся окне в строке ASCII пишем текст, например, «Инфицировано» (рис. 7.15). После записанного текста необходимо записать байт с содержимым 00 (или оставить 00), что будет являться признаком окончания строки с параметром.

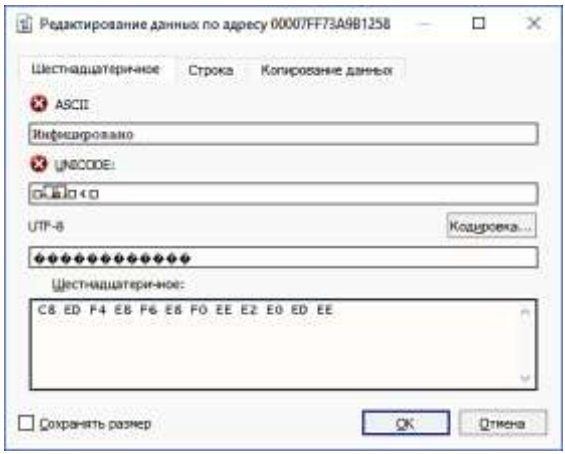


Рис. 7.15. Окно записи в дамп памяти

Копируем адрес начала строки с текстом Инфицировано и меняем адреса в строках

lea rdx,…

lea r8,…

11. Сохраняем все изменения (последовательность сохранения рассмотрена ранее).

7.4. Лабораторная работа «Внедрение кода в exe-файл в конец секции с добавлением новых API-функций»

Цель занятия: приобрести практические навыки исправления ошибок в программном обеспечении, которое находится в эксплуатации;

создание новой функциональности, используя неявное в эксплуатации программное обеспечение для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с комментариями;

– результат выполнения программы (скриншоты упрощенного окна (MessageBox) и отладчика;

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Постановка задачи

Выполнить задачу в несколько этапов:

1. В начале программы осуществить проверку возможности микропроцессора поддерживать команды AVX и AVX2. О чем вывести функциями MessageBox соответствующие сообщения.

2. Написать программу в среде masm64 в соответствии с вариантом задания. Вывести 10 полученных результатов через функцию MessageBox, а также номер варианта, название лабораторной работы, шифр группы, фамилию исполнителя.

3. Произвести подсчет свободного места в секции кода.

4. Осуществить внедрение кода после вывода сообщений о проверке возможности микропроцессора поддерживать команды AVX и AVX2 и перед выводом упрощенного окна с основными результатами программы. Задание на внедрение кода: внедрить дополнительную функцию MessageBox с выводом слова «Инфицировано», а так же шифр группы и фамилию исполнителя.

Внедрить дополнительную функцию MessageBox с выводом слова «Инфицировано», а так же шифр группы и фамилию исполнителя (или столько символов, сколько вмещается).

При использовании инструкций целесообразно воспользоваться справочниками, например https://www.felixcloutier.com/x86/index.html.

Задание. Выполнить действия над массивами (массивом) из 10 чисел и вывести полученные результаты.

1. Выполнение инструкции VSUBPD и извлечение корней из полученных результатов.
2. Выполнение инструкции VADDSUBPD.
3. Выполнение инструкции VDPPD.
4. Выполнение инструкции VMINPD.
5. Выполнение инструкции VMINPD.
6. Выполнение инструкции VPMULHUW.
7. Выполнение инструкции VPMULHW.
8. Выполнение инструкции VPMULLD.
9. Выполнение инструкции VPHMINPOSUW.
10. Выполнение инструкции VPMADDWD.
11. Выполнение инструкции VPMADDUBSW.
12. Выполнение инструкции VPMINSB.
13. Выполнение инструкции VPSLLDQ.
14. Выполнение инструкции VPSLLD.
15. Выполнение инструкции VROUNDPD.
16. Выполнение инструкции VCMPPD.
17. Выполнение инструкции VPCMPEQW.
18. Выполнение инструкции VPCMPGTB.
19. Выполнение инструкции VMINPD.
20. Выполнение инструкции VMAXPD.
21. Выполнение инструкции VANDPD.
22. Выполнение инструкции VPAND.
23. Выполнение инструкции VPOR.
24. Выполнение инструкции VXOR.
25. Выполнение инструкции VPTEST.

Пример программы.

Программа 7.4. Сложение массивов чисел командами vaddpd:

```
title Рысованый А.Н.
; masm64. AVX-команды.
; Сложение массивов чисел командами vaddpd и addss
include \masm64\include64\masm64rt.inc ;
.data
a1 dd 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18 ;18 чисел
b1 dd 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19 ;18 чисел
len1 EQU ($-b1)/type b1
```

```
      res1 dd 18 dup(0),0
      res2 dq len1 dup(0),0
      buf1 dq len1 dup(0),0
  frmt db "Использование команды vaddpd",10,
  "1  2  3  4  5  6  7 8  9 10 11 12 13 14 15 16 17 18",10,
  "2  3  4  5  6  7  8 9 10 11 12 13 14 15 16 17 18 19",10,10,
  " Ответ =", 10 dup("   %d "),10,10,
  "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
  9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0
      tit3 db "Сложение массивов чисел командами vaddpd и addss",0
.code
entry_point proc

.data
    titl db "Проверка микропроцессора на поддержку команд AVX2",0 ;
название окна
    szInf db "Команды AVX ПОДДЕРЖИВАЮТСЯ!!!",0  ;
    inf db "Команды AVX микропроцессором НЕ поддерживаются",0
.code
      ; проверка на поддержку AVX команд
      mov EAX,1 ; при использования 64-разрядной ОС
      cpuid   ; по eax производится идентификация микропроцессора
      and  ecx,10000000h ; ecx:= ecx v 1000 0000h (28 разряд)
      jnz  exit1                ; перейти на exit, если не ноль
      invoke MessageBox,0,addr inf,addr titl,MB_OK
      jmp exit2
exit1:
      invoke MessageBox,0,addr szInf,addr titl,MB_ICONINFORMATION
exit2:

.data
tit2 db "Проверка микропроцессора на поддержку команд AVX2",0 ;
название окна
    szInf2 db "Команды AVX2 ПОДДЕРЖИВАЮТСЯ!!!",0  ;
    inf2 db "Команды AVX2 микропроцессором НЕ поддерживаются",0
.code
      mov eax,7
      mov ecx,0
      cpuid ; по содержимому rax производится идентификация МП
      and  rbx,20h ;   (5 разряд)
      jnz  @f      ; перейти, если не нуль
      invoke MessageBox,0,addr inf2,addr tit2,MB_OK
      jmp exit12
@@:
      invoke MessageBox,0,addr szInf2,addr tit2,MB_ICONINFORMATION
```

```
        jmp exit12

        mov rax,len1 ;
        mov rbx,8    ; 32 x 8 = 256
        xor rdx,rdx
        div rbx ; определение количества циклов в rax и остатка в rdx
        mov rcx,rax
        lea rsi,a1
        lea rdi,b1
        lea rbx,res1

m1: vmovups ymm0,[rsi]
        vmovups ymm1,[rdi]
        vaddpd ymm2,ymm0,ymm1
        vmovups [rbx],ymm2
        add rdi,32     ; 32 x 8 = 256
        add rsi,32     ; смещение на 256
        add rbx,32     ; смещение на 32 байта = 256
        loop m1
        cmp rdx,0h   ; сравнение остатка с нулем
        jz exit10       ; перейти, если ноль
        mov rcx,rdx  ; занесение содержимого rdx в счетчик
m2:
         vmovss xmm3,dword ptr [rsi]     ; занесение элемента mas1
         vaddss XMM3,XMM3,dword ptr [rdi] ; элемент mas2
         vmovss dword ptr [rbx],XMM3 ; занесение содержимого в res1
         add rsi,4                ; смещение указателя на mas1
         add rdi,4                ; смещение указателя на mas2
         add rbx,4                ; смещение указателя на res1
        loop m2
exit10:
        lea rsi,res1
        lea rdi,res2
        mov rcx,len1 ; количество чисел, которые выводятся в окно
m5:
        movsxd r15,dword ptr [rsi]
        mov qword ptr[rdi],r15
        add rsi,4
        add rdi,8
        dec rcx
        jnz m5
          ; перезапись в регистры
        mov rax,res2[0]
        mov rbx,res2[8]
        mov rsi,res2[16]
```

```
        mov rdi,res2[24]
        mov r10,res2[32]
        mov r11,res2[40]
        mov r12,res2[48]
        mov r13,res2[56]
        mov r14,res2[64]
        mov r15,res2[72]
invoke wsprintf,ADDR buf1,ADDR frmt,rax,rbx,rsi,rdi,r10,r11,r12, r13,r14, r15;
        invoke MessageBox,0,ADDR buf1,ADDR titl,MB_OK
exit12:  invoke ExitProcess,0
entry_point endp
        end
```

При запуске exe-файла последовательно выводятся соответствующие сообщения (рис. 7.16).

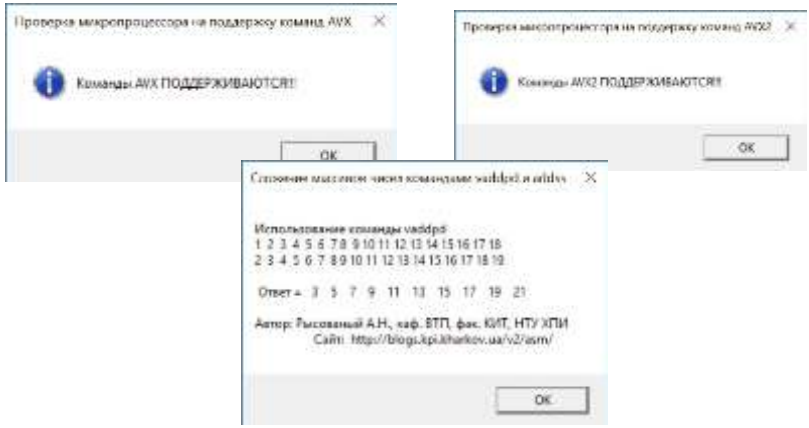


Рис. 7.16. Окно с сообщениями программы

Осуществим внедрение перед выводом последнего сообщения функцией MessageBox.

Сначала определим объем в байтах свободных ячеек в секции кода.

Для этой цели воспользуемся программой-анализатором LotdPE. Выполним последовательность действий:

– в программе нажать на клавишу меню PE Editor и выбрать исследуемый exe-файл;

– в появившемся окне нажать клавишу меню Sections;

– выбрать секцию, где расположен текст программы (это секция text), нажать правую клавишу мышки и выбрать пункт меню hex edit section;

– в появившемся окне memory buffer выбранная секция выделится темным цветом (рис. 7.17). В конце секции свободные ячейки будут заполнены нулями. Их необходимо просто посчитать.

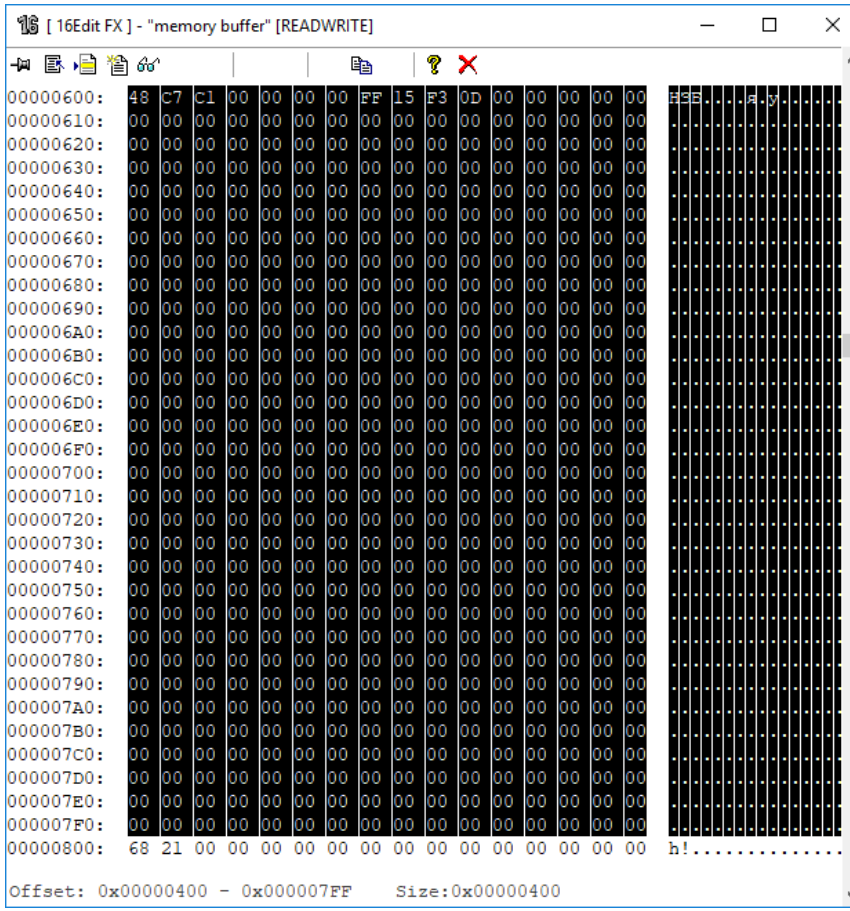


Рис. 7.17. Окно hex редактора секции кода

Для подсчета количества байтов свободных ячеек памяти в секции кода необходимо из последней свободной ячейки секции вычесть последнюю занятую ячейку после окончания кода:

7FF h – 60A h = 1F5 h = 501 байт.

Если предполагаемого размера могло не хватить, то можно добавить новую секцию или увеличить размер этой секции. Но это вопросы следующих рассмотрений.

Загружаем exe-файл в отладчик x64Dbg. В связи с тем, что в соответствии с заданием необходимо внедрить функцию MessageBox перед выводом упрощенного окна с основными результатами программы, то будем искать место в конце программы.

Осуществим внедрение перед выводом последнего сообщения функцией MessageBox. Окно отладчика x64Dbg с окончанием кода программы, куда предполагается внедрить код, представлено на рис. 7.18.

```
00007FF78BFB114D    48:83C7 08          add rdi,8
00007FF78BFB1151    48:FFC9             dec rcx
00007FF78BFB1154  ^ 75 ED               jne rp_15-2-20.7FF78BFB1143
00007FF78BFB1156    48:8B05 DB1F0000    mov rax,qword ptr ds:[7FF78BFB3138]
00007FF78BFB115D    48:8B1D DC1F0000    mov rbx,qword ptr ds:[7FF78BFB3140]
00007FF78BFB1164    48:8B35 DD1F0000    mov rsi,qword ptr ds:[7FF78BFB3148]
00007FF78BFB116B    48:8B3D DE1F0000    mov rdi,qword ptr ds:[7FF78BFB3150]
00007FF78BFB1172    4C:8B15 DF1F0000    mov r10,qword ptr ds:[7FF78BFB3158]
00007FF78BFB1179    4C:8B1D E01F0000    mov r11,qword ptr ds:[7FF78BFB3160]
00007FF78BFB1180    4C:8B25 E11F0000    mov r12,qword ptr ds:[7FF78BFB3168]
00007FF78BFB1187    4C:8B2D E21F0000    mov r13,qword ptr ds:[7FF78BFB3170]
00007FF78BFB118E    4C:8B35 E31F0000    mov r14,qword ptr ds:[7FF78BFB3178]
00007FF78BFB1195    4C:8B3D E41F0000    mov r15,qword ptr ds:[7FF78BFB3180]
00007FF78BFB119C    48:8D0D 2D200000    lea rcx,qword ptr ds:[7FF78BFB31D0]
00007FF78BFB11A3    48:8D15 BE200000    lea rdx,qword ptr ds:[7FF78BFB3268]
00007FF78BFB11AA    4C:8BC0             mov r8,rax
00007FF78BFB11AD    4C:8BCB             mov r9,rbx
00007FF78BFB11B0    48:897424 20        mov qword ptr ss:[rsp+20],rsi
00007FF78BFB11B5    48:897C24 28        mov qword ptr ss:[rsp+28],rdi
00007FF78BFB11BA    4C:895424 30        mov qword ptr ss:[rsp+30],r10
00007FF78BFB11BF    4C:895C24 38        mov qword ptr ss:[rsp+38],r11
00007FF78BFB11C4    4C:896424 40        mov qword ptr ss:[rsp+40],r12
00007FF78BFB11C9    4C:896C24 48        mov qword ptr ss:[rsp+48],r13
00007FF78BFB11CE    4C:897424 50        mov qword ptr ss:[rsp+50],r14
00007FF78BFB11D3    4C:897C24 58        mov qword ptr ss:[rsp+58],r15
00007FF78BFB11D8    FF15 320E0000       call qword ptr ds:[<&wsprintfA>]
00007FF78BFB11DE    48:C7C1 00000000    mov rcx,0
00007FF78BFB11E5    48:8D15 E41F0000    lea rdx,qword ptr ds:[7FF78BFB31D0]
00007FF78BFB11EC    4C:8D05 A0210000    lea r8,qword ptr ds:[7FF78BFB3393]
00007FF78BFB11F3    49:C7C1 00000000    mov r9,0
00007FF78BFB11FA    FF15 180E0000       call qword ptr ds:[<&MessageBoxA>]
00007FF78BFB1200    48:C7C1 00000000    mov rcx,0
00007FF78BFB1207    FF15 F30D0000       call qword ptr ds:[<&FatalExit>]
00007FF78BFB120D    0000                add byte ptr ds:[rax],al
00007FF78BFB120F    0000                add byte ptr ds:[rax],al
```

Рис. 7.18. Окно отладчика x64Dbg перед внедрением кода

Дополнительную функцию MessageBox расположим после последней строки кода ExitProcess. А команду перехода jmp с адресом перехода на

новый блок команд поставим на место первого параметра функции MessageBox (mov rcx,0).

Немного изменим последовательность внедрения.

1. Освобождаем место для команды jmp.

Ставим курсор на строку кода mov rcx,0

Копируем строку. Для этого дважды нажимаем на эту строку (строка выделяется), нажимаем Ctrl+C.

Опускаемся в конец программы, дважды нажимаем (выделяем) первую свободную строку и в появившемся окне вставляем из буфера строку mov rcx,0 (нажимаем Ctrl+V) (рис. 7.19).

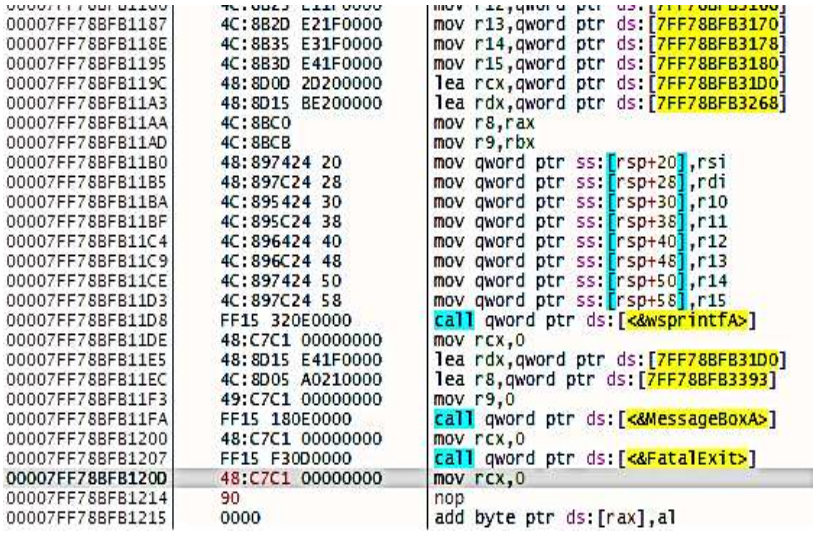


Рис. 7.19. Окно отладчика x64Dbg перед заменой строки

2. Вставляем команду jmp вверху программы.

При ближнем переходе команда jmp имеет размерность 2 байта. Команда jmp требует адрес перехода на новый блок кода в конце программы. Для этого, перемещаем курсор на свободную строку после вызова функции ExitProcess. Нажимаем правую клавишу мышки и

выбираем пункты Копировать / Адрес свободной строки. Адрес заносится в буфер памяти.

Перемещаемся вверх по коду на первый параметр функции MessageBox – на строку mov rcx,0. Дважды нажимаем левую клавишу мышки, набираем слово jmp и вставляем адрес из буфера (нажимаем Ctrl + V) (рис. 7.20).

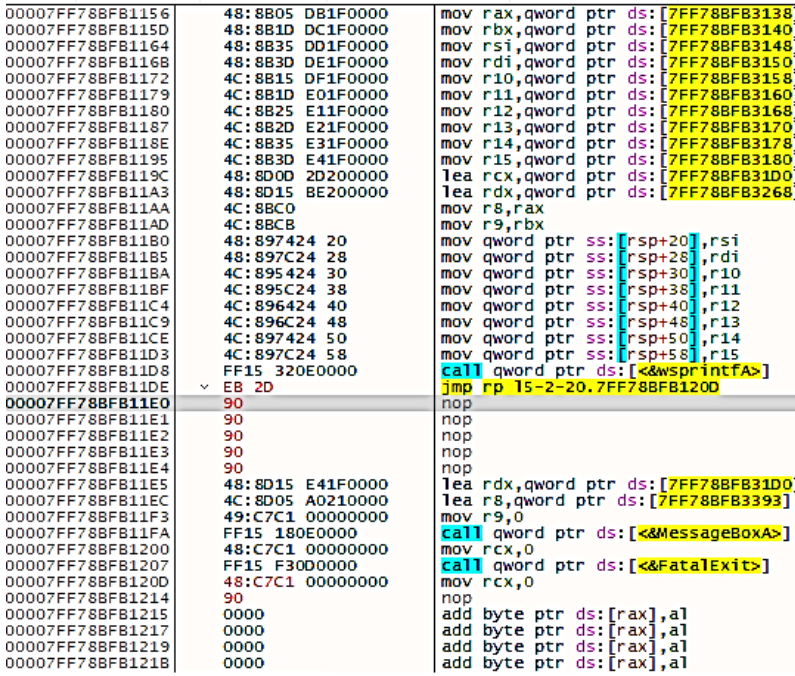


Рис. 7.20. Ввод команды jmp

3. Копируем параметры и саму функцию MessageBox на свободное место.

Необходимо последовательно после первой перенесенной команды вставить строки из нашей же программы:

```
mov rcx,0
lea rdx, ds:[…]
lea r8, ds:[…]
mov r9d, r9d
call qword ptr ds:[<&MessageBoxA>]
```

Для этого, подводим курсор к строке, которую необходимо скопировать. Дважды нажимаем левую клавишу мышки и копируем строку, нажимая сочетание клавиш Ctrl + C. Для вставки скопированной строки на новое место нажимаем сочетание Ctrl + V.

При копировании команды вызова функции (команда call) надо использовать только сочетание клавиш Ctrl + C. Только в этом случае копируется не мнемоника команды, а ***адрес ячейки памяти таблицы импорта***, где располагается эта функция.

Таким образом было скопировано функцию MessageBox с параметрами, которая занимает в памяти в среднем 25 байтов.

4. Вспоминаем, что первой командой, которую мы заменили была команда mov rcx,0. Эту команду заменили на команду jmp.

Следовательно, необходимо в конце нового блока кода вставить команду mov rcx,0.

5. Последней командой вставленного блока должна быть команда возврата jmp с адресом возврата на продолжение старого блока кода (рис. 7.21).

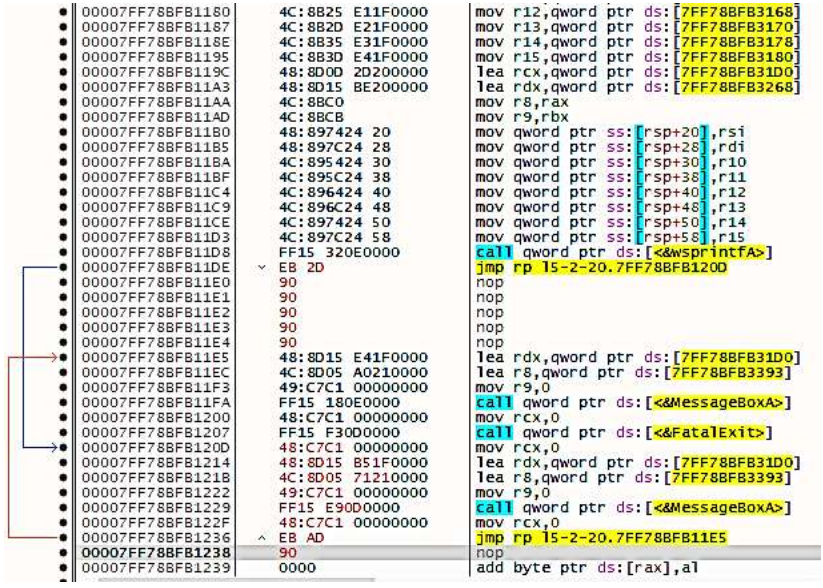


Рис. 7.21. Программа после вставки команды возврата jmp

Необходимо сохранить временные изменения. Почему временные? – Потому что еще не вставлены новые название титула окна и информация в этом окне.

Это можно сделать **двумя способами**.

Первый способ: выбираем Файл/ Исправить файл. В новом окне выбрать пункт Исправить файл. Затем – новое имя и расширение exe. Для того, чтобы иметь возможность вернуться и загрузить наиболее правильную исправленную версию необходимо сохранять изменения и давать каждый раз новые имена или индексы к ним для сохраненных exe-файлов.

Второй способ: выбрать пиктограмму с названием Исправления, нажать на кнопку Исправить файл, а далее новое имя и расширение exe.

5. Изменяем параметры функции MessageBox.

Меняем информацию, которая будет выводиться функцией внутри упрощенного окна.

Сначала изменим параметры, которые передаются через регистр rdx. В регистре rdx передается сообщение, а название титула упрощенного окна – через регистр r8.

Скопируем адрес свободной строки кода, куда будем записывать текстовую информацию, которая располагается в окне и текстовую информацию, которая располагается в титуле. Для этого **выделяем первую свободную строку** секции кода (после последней команды jmp), нажимаем правую клавишу мышки и выбираем пункты меню Копировать/ Адрес.

Подводим курсор к параметру функции MessageBox, который запоминает в регистре rdx адрес начальной ячейки, где должна располагаться информация внутри упрощенного окна и вставляем адрес из буфера сочетанием клавиш Ctrl + V.

Теперь необходимо обратиться к дампу памяти по адресу, который записан в регистре rdx. Для этого выделяем строку кода с командой

lea rdx, qword ptr ds: […],

нажимаем правую клавишу мышки и выбираем пункты меню Перейти к дампу / Константа.

После этого ставим курсор на начальный байт в **дампе** памяти отладчика, нажимаем правую клавишу мышки и выбираем меню Двоичные операции / Редактировать. В появившемся окне в строке ASCII пишем текст, например, «Инфицировано» (рис. 7.22). После записанного текста необходимо записать байт с содержимым 00 (или оставить 00), что будет являться признаком окончания строки с параметром.

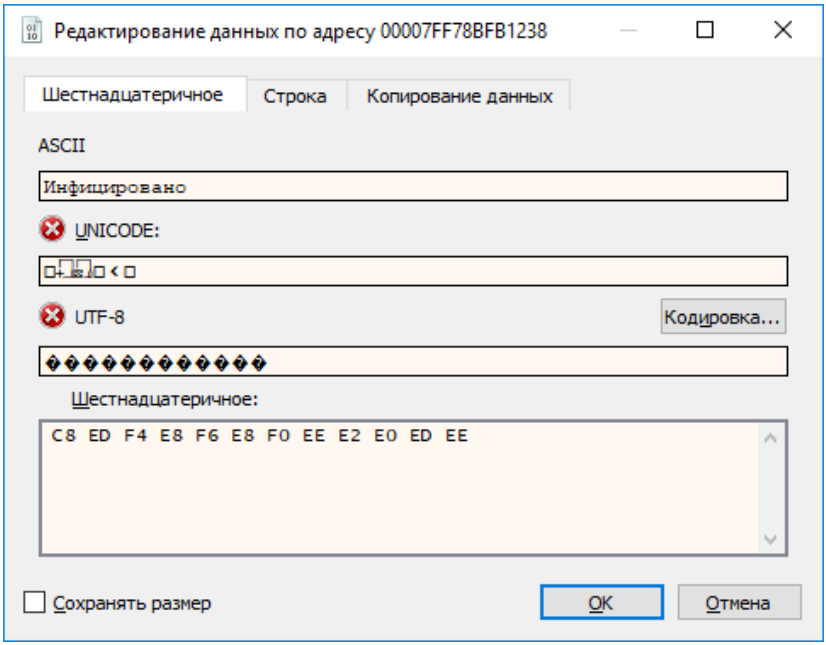


Рис. 7.22. Окно записи в дамп памяти

Теперь необходимо изменить параметры, которые передаются через регистр r8 (текст титула окна). Выполняются аналогичные действия: запоминание адреса свободной строки кода, передача этого адреса в регистр r8, изменение по этому адресу в дампе памяти текста титула окна.

После записанного текста необходимо записать байт с содержимым 00 (или оставить 00), что будет являться признаком окончания строки с параметром.

6. Сохраняем все изменения.

После сохранения файла под новым именем с расширением exe выведется окно сообщения (рис. 7.23).

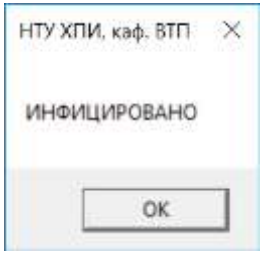


Рис. 7.23. Результат внедрения

Чтобы вывести больше текстовой информации необходимо добавить новую секцию (что является материалам для следующих рассмотрений).

В учебных целях проще увеличить саму программу, в которую собираемся осуществлять внедрение. Это можно осуществить, если зарезервировать, например, перед функцией окончания программы, блок под данные:

```
jmp m111
db 60 dup(0)
m111:
```

Этим блоком зарезервировано 60 ячеек памяти: под функцию MessageBox примерно 25-30 байтов и оставшиеся от 60-ти – для текстовой информации. В последующем в этот блок данных можно осуществлять внедрение, уже не беспокоясь о недостатке места.

8. ВНЕДРЕНИЕ КОДА В НОВУЮ СЕКЦИЮ КОДА

8.1. Основной алгоритм внедрения в новую секцию кода

Решение проблемы доступного пространства при помощи добавления новой секции имеет некоторые недостатки. Практически все антивирусы опознают нестандартные секции, и если, к тому же, там есть полный набор флагов на чтение/запись/выполнение, то эта ситуация выглядит еще более подозрительно. Но такой способ дает огромное преимущество в объеме внедренного кода.

В начале проведения внедрения необходимо получить общие сведения о файле: определение языка программирования (или компилятора), разрядности архитектуры (32/64), название упаковщика или криптора. Для этого надо использовать программы-анализаторы PE-файла, например CFF Explorer, PeStudio или др.

В общем случае ***алгоритм добавления новой секции*** может быть таким:

1. При помощи программ-анализаторов PE-файла.

– определяем язык программирования, название упаковщика или криптора;

– определяем объем свободных ячеек основной секции кода для вставки команды jmp перехода на новый блок;

– добавляем новую секцию, проставляем ее физический и виртуальный размеры, а также флаги секции.

2. При помощи 16-ричного редактора, например FlexHex, заполняем ячейки нового блока нулями.

3. При помощи отладчика кода, например x64Dbg, перемещаем одну или две команды по адресу новой ячейки и на освободившееся место поставить команду перехода jmp на этот адрес.

В новой секции пишем новый блок кода, последней командой которого должна быть команда возврата jmp на продолжение кода в основной секции.

8.2. Внедрение в новую секцию кода

Рассмотрим добавление новой секции на конкретном примере. Предположим, что существует программа (программа 7.1), написанная

в среде masm64 с использованием команд SSE, которая вычисляет математическое уравнение. В эту программу необходимо внедрить свое сообщение функцией MessageBox и добавить его в новую секцию.

Программа 8.1. Сложение 32-разрядных дробных чисел одного массива с использованием команд SSE:

```
        include \masm64\include64\masm64rt.inc ;
.data
        mas1 dd 1.,2.,3.,4.
        titl1 db "Сложение чисел массива ",0 ;
        fmt db "Сложение 32-разрядных дробных чисел одного
массива",10,
        "с использованием команд SSE",10,
        "mas1:  1., 2., 3., 4.", 10,10, "Сумма: %d",10,
        "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
        9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0
         buf1 dq 0,0 ; буфер

.code
entry_point proc
rcall MessageBox,0,"Использование SSE команд","Сложение чисел
массива",MB_OK
;movups - пересылка невыровненных упакованных коротких
вещественных значений
;movaps - пересылка выровненных упаков. коротких веществ. значений
        movups XMM0,mas1          ; XMM0:= 4. 3. 2. 1.
        movaps XMM1,XMM0          ; XMM1:= 4. 3. 2. 1.
        shufps XMM1,XMM1,11111001b ; XMM1:= 4. 4. 3. 2.
        addss XMM0,XMM1           ; XMM0:= 4. 3. 2. 3.
        shufps XMM1,XMM1,11111001b ; XMM1:= 4. 4. 4. 3.
        addss XMM0,XMM1           ; XMM0:= 4. 3. 2. 6.
        shufps XMM1,XMM1,11111001b ; XMM1:= 4. 4. 4. 4.
        addss XMM0,XMM1           ; XMM0:= 4. 3. 2. 10.
        cvttss2si eax,xmm0
        movsxd r15,eax
invoke wsprintf,addr buf1,addr fmt,r15
invoke MessageBox,0,addr buf1,ADDR titl1,MB_ICONINFORMATION
        invoke ExitProcess,0
        entry_point endp
        end
```

Результатами выполнения программы являются последовательно появляющиеся окна сообщений (рис. 8.1).

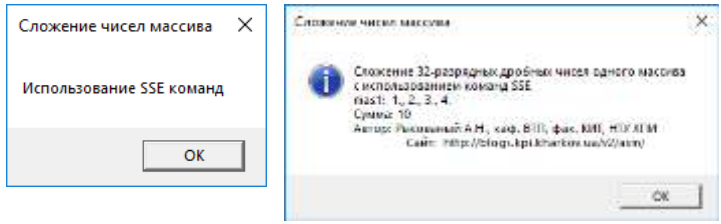


Рис. 8.1. Окна с результатами выполнения основной программы

Рассмотрим процесс добавления новой секции кода и внедрение сообщения при помощи функции MessageBox между двумя функциями MessageBox.

Этапы внедрения рассмотрим по пунктам.

1. Определяем объем свободных ячеек основной секции кода для вставки команды jmp перехода на новый блок. Для этого в программе-анализаторе PE-кода LordPE (рис. 8.2) выполняем действия:

– открываем исследуемый файл;

– нажимаем на кнопку меню Sections;

– выделяем мышкой секцию кода text;

– нажимаем правую клавишу мышки и выбираем кнопку меню hex edit section…

– подсчитываем объем свободных ячеек основной секции кода. Содержимое выделенной секции выводится на черном фоне. Для расчета необходимо из начального адреса следующей секции вычетаем адрес первой свободной ячейки после окончания кода:

$$600\ h - 493\ h = 16D\ h = 365 \text{ байтов.}$$

Делаем вывод, что этого размера вполне хватает на внедрение небольшого кода. Но мы рассматриваем создание новой секции.

2. Добавляем новую секцию кода. Для этого в программе-анализаторе PE-кода LordPE выполняем действия:

– нажимаем на кнопку меню Sections;

– выделяем любую секцию и добавляем новую секцию при помощи кнопки меню add section header (рис. 8.3);

– в появившемся окне выделяем новую секцию и для изменения свойств новой секции нажимаем на кнопку меню edit section header… (рис. 8.4). При этом необходимо запомнить смещение ROffset

(необработанное смещение) новой секции. В рассматриваемом случае оно равно 0000 0A00h;

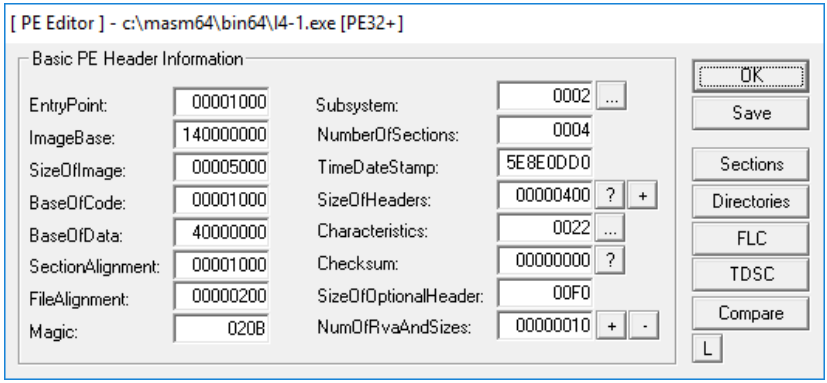


Рис. 8.2. Основное окно программы LordPE

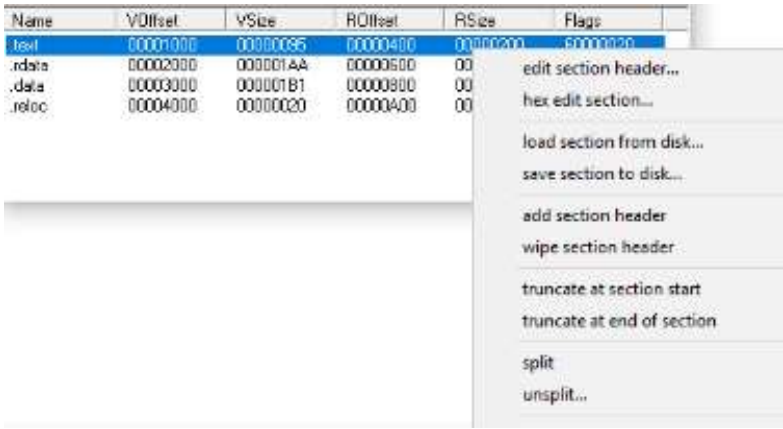


Рис. 8.3. Добавление новой секции

– в окне свойств секции (edit section header) вводим имя секции или оставляем прежнее, а также физический размер секции в файле RSize (необработанный размер), который в большинстве случаев равен 200h и виртуальный размер секции в памяти VSize (виртуальный размер).

Расположение секций в файле на диске всегда отличается от его расположения после загрузки в память. Это происходит из-за

управления виртуальной памятью на основе страниц, которое использует Windows.

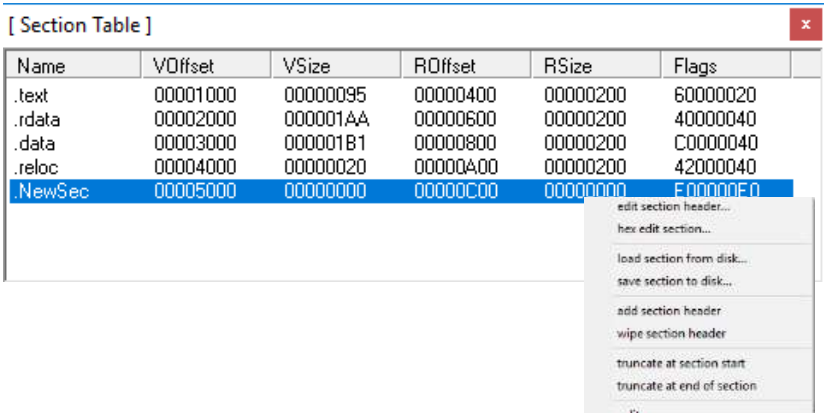


Рис. 8.4. Вызов окна чтения свойств секции

Если предполагается использовать почти весь объем секции, то и виртуальный размер VSize должен быть таким же как и физический или больше чтобы не тормозить запись и чтение данных секции (рис. 8.5);

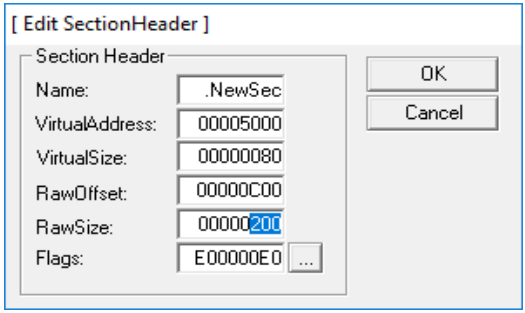


Рис. 8.5. Окно свойств секции

– для изменения признаков секции выбираем кнопку меню Flags (рис. 8.6).

Если после проведенных действий попытаться открыть exe-файл исследуемой программы в отладчике x64Dbg, то такая попытка не завершится успешно. Дело в том, что новая секция объявлена, в ней проставлены признаки, но ничего не записано (даже нули).

Рис. 8.6. Окно признаков секции

3. Заполняем ячейки нового блока нулями.

Для записи в новую секцию нулей необходимо использовать 16-ричный редактор. Выберем, например 16-ричный редактор FlexHex. Перед запуском программы FlexHex необходимо освободить исследуемый exe-файл. Открываем программу с новой секцией в редакторе FlexHex и перемещаемся в конец секции.

Ищем место в секции, где расположена (объявлена) новая секция. А она начинается с адреса RawOffset (ROffset). В нашем случае – это адрес 0000 0C00h (рис. 8.7).

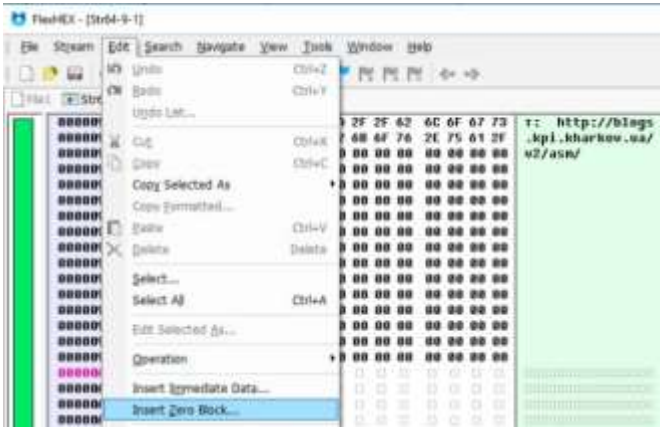


Рис. 8.7. Основное окно редактора 16-ричного редактора FlexHex

Ставим курсор на первый байт новой секции (по адресу 0000 0C00h).

Выбираем пункты меню Edit / Insert Zero Block.

– в появившемся окне проставляем размер новой секции: Block Size = 200h (рис. 8.8);

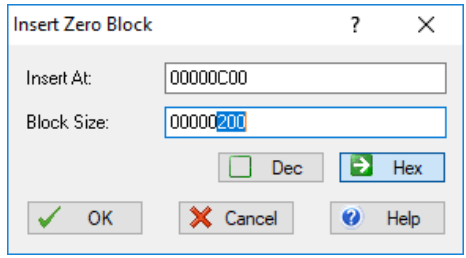


Рис. 8.8. Окно заполнения размера секции

– сохраняем изменения File / Save;

– желательно проверить готовность новой секции программой-анализатором LordPE. Для этого необходимо выбрать новую секцию, нажать правую клавишу мышки и нажать пункт меню hex edit section. В появившемся окне memory buffer выбранная секция выделится темным цветом (рис. 8.9).

При нажатии левой клавиши мышки на любом окне программы LordPE темный фон исчезает.

4. Вносим изменения в код, используя отладчик x64Dbg.

В отладчике x64Dbg открываем исследуемую программу и еще раз убеждаемся, что новая секция готова к использованию. Для этого выбираем строку основного меню и в ней пункт Карта памяти, после чего откроется окно (рис. 8.10).

Выделяем строку с названием нашего exe-файла и если ниже этого названия не раскрыты названия наших секций, то нажимаем на этой строке правую клавишу мышки и выбираем пункт Переключить вид.

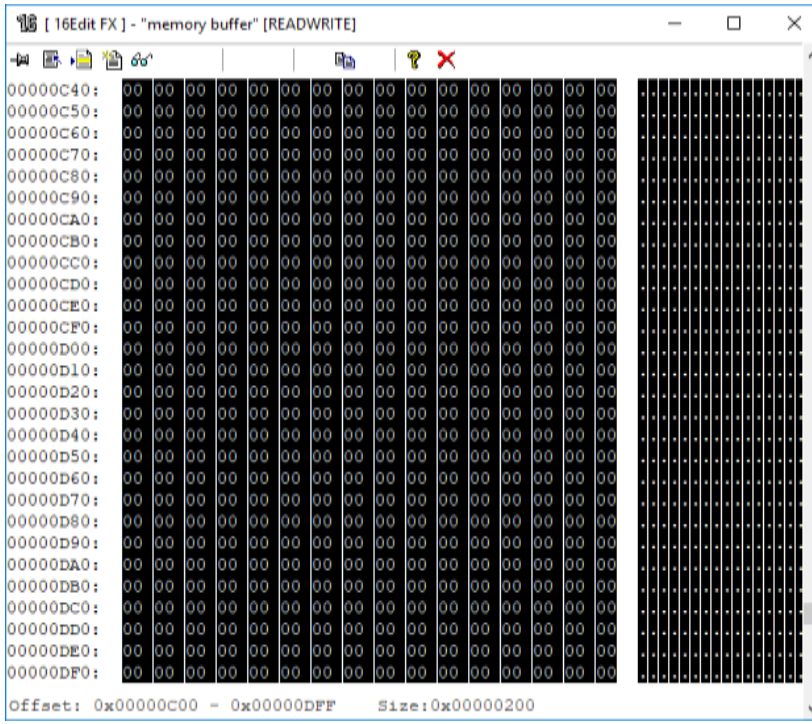


Рис. 8.9. Окно новой секции в LordPE

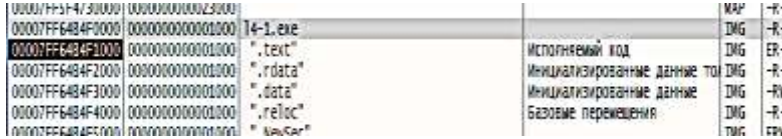


Рис. 8.10. Окно меню Карта памяти в отладчике x64Dbg

5. Проанализируем исследуемую программу в отладчике x64Dbg. Внешний вид кода показан на рис. 8.11.

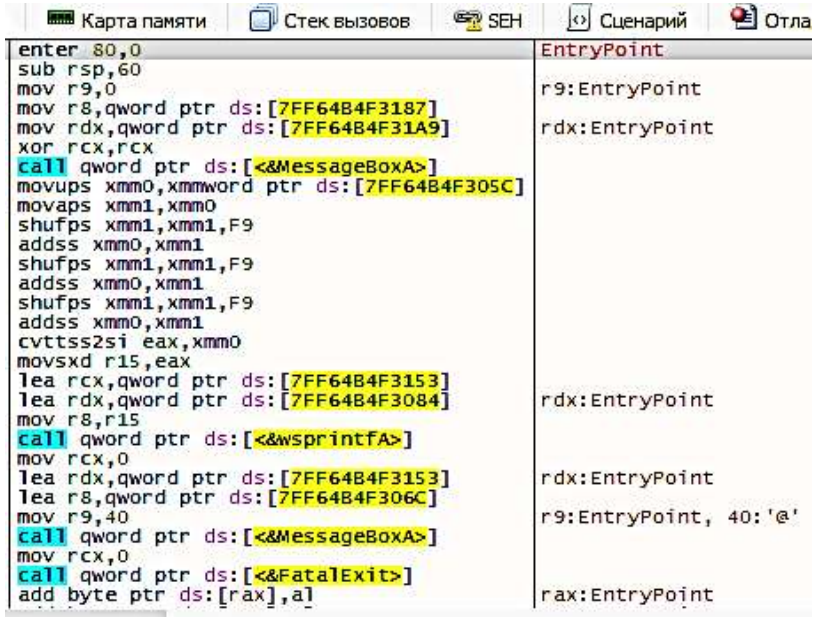


Рис. 8.11. Код программы в отладчике x64Dbg

Между двумя функциями MessageBox необходимо вставить новую функцию MessageBox. По заданию новая функция MessageBox должна размещаться в новой секции, которую мы уже создали. Для перехода на эту секцию кода надо в исследуемую программу вставить команду jmp на адрес этой секции.

6. Освободим место для команды jmp.

Выберем команду

```
movups xmm0,xmmword ptr ds:[7FF64B4F305C].
```

и перенесем ее в новую секцию.

Для этого дважды нажимаем левую клавишу мышки на этой строке (рис. 8.12) и копируем ее в буфер (нажимаем Ctrl + C).

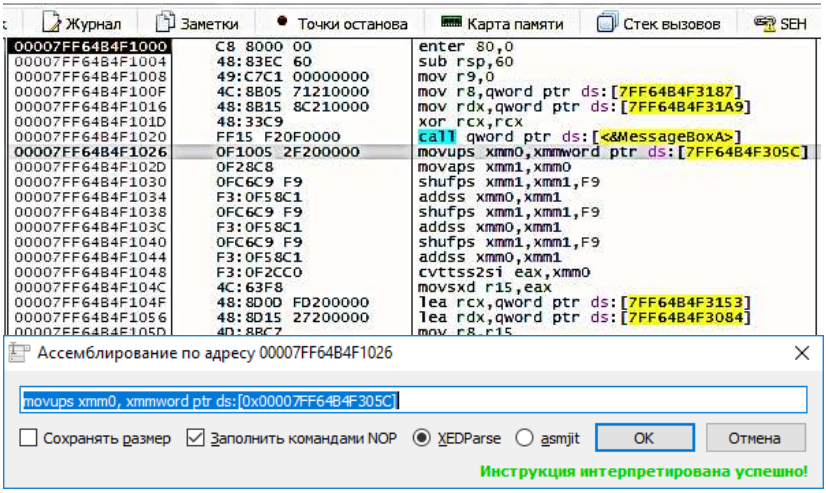


Рис. 8.12. Окно копирования строки кода

7. Переносим скопированную команду в новую секцию.

В строке дополнительного меню выбираем пункт Карта памяти и ищем имя нашего exe-файла. Выделяем эту строку и если ниже не раскрыты названия наших секций, то нажимаем на этой строке правую клавишу мышки и выбираем пункт Переключить вид. Подводим курсор к строке с названием новой секции и дважды нажимаем на этой строке.

Попадаем в новую секцию. Дважды нажимаем на первую строчку кода. Появляется окно (рис. 8.13), в которое вставляем перенесенную строку при помощи сочетания клавиш Ctrl + V.

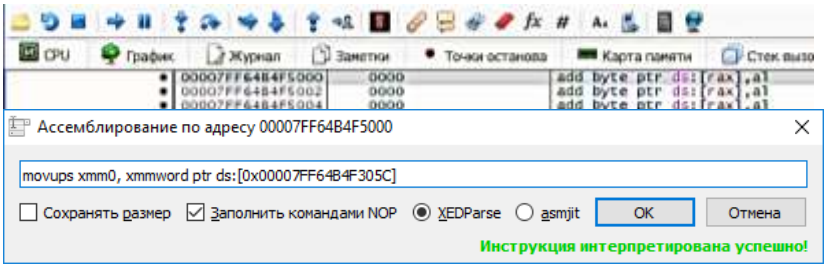


Рис. 8.13. Окно с содержимым первой строки новой секции

8. Копируем адрес первой строки, чтобы на нее поставить команду перехода jmp в основной секции кода. Для этого, на первой строке секции выбираем кнопки меню Копировать / Адрес.

9. Переходим на основную секцию кода. Для этого в нижней (последней) строке основного меню выбираем пункт Карта памяти (см. рис. 7.10). После этого подводим курсор на строку кода с названием основной секции кода text, дважды нажимаем на этой строке и попадаем в основную секцию.

Дважды нажимаем на строку кода, которую уже перенесли в новую секцию, пишем слово jmp, а затем вставляем из буфера Ctrl + V (рис. 8.14).

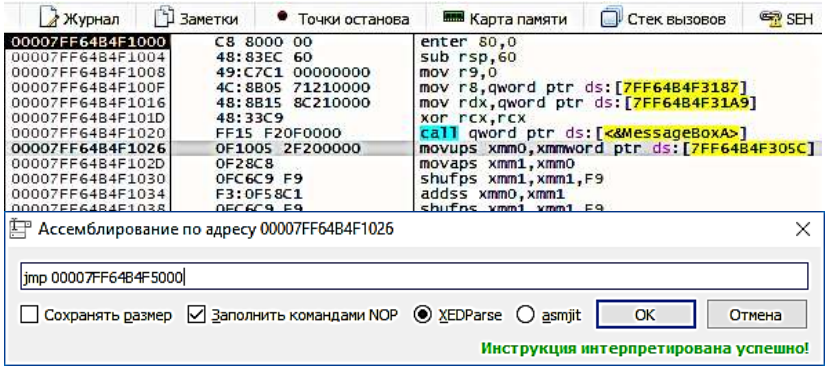


Рис. 8.14. Окно с подготовленной для вставки командой

Результат вставки первой команды показан на рис. 8.15.

Рис. 8.15. Окно с командой перехода на новую секцию

10. В качестве примера вставки выбираем вторую функцию MessageBox и построчно переносим сначала параметры функции, а затем и вызов самой функции в новую секцию. Для этого, для каждой переносимой строчки выполняем последовательность действий:

– копируем строчку кода. На выбранной строчке кода дважды нажимаем левую клавишу мышки и нажимаем Ctrl + C;

– выбираем пункт Карта памяти;

– дважды нажимаем на строчке кода с названием новой секции;

– выбираем место для вставки кода и нажимаем Ctrl + V.

Результат переноса параметров и самой функции показан на рис. 8.16.

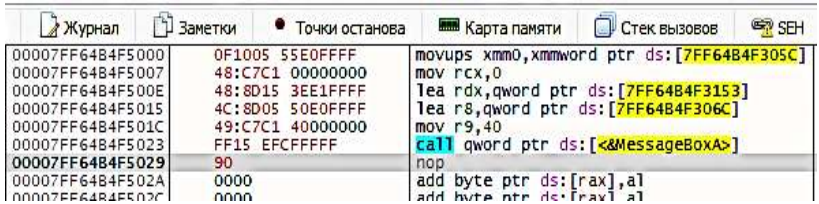


Рис. 8.16. Промежуточный результат переноса команд в новую секцию

11. Последней командой должна быть команда возврата jmp на адрес продолжения кода в основной секции. Для этого перейдем на основную секцию и скопируем адрес на неизмененную строку кода (рис. 8.17).

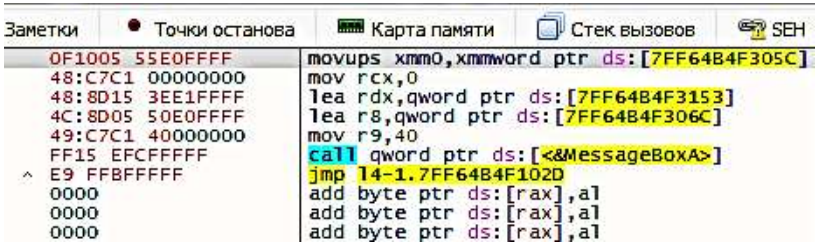


Рис. 8.17. Содержимое новой секции кода

12. Правим содержимое параметров функции MessageBox.

Меняем адреса ячеек в параметрах. Будем выводить в титуле и окне функции MessageBox одну и туже информацию. Для этого выделяем первую свободную строку новой секции кода (после последней команды jmp), нажимаем правую клавишу мышки и выбираем пункты меню Перейти к дампу/ Выделенный адрес. После чего отображается содержимое дампа памяти.

В параметрах функции через регистр rdx передается содержимое окна функции MessageBox.

Через регистр r8 передается название титула.

Поэтому меняем адреса ячеек памяти, в которых будет располагаться новая информация.

Для смены адреса расположения названия ставим курсор на начальный байт в дампе памяти отладчика, нажимаем правую клавишу мышки и выбираем меню Двоичные операции / Редактировать. В появившемся окне в строке ASCII пишем текст текст Инфицировано (рис. 8.18).

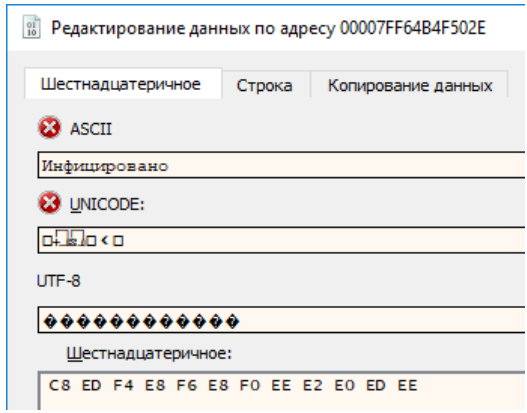


Рис. 8.18. Редактирование данных окна

Снова ставим курсор на начало этого сообщения и копируем адрес. После чего подводим курсор на строку кода с передачей адреса в регистр rdx и меняем адрес.

После этого необходимо написать текст в окне, адрес которого передается через регистр r8. Но теперь необходимо пропустить хотя бы одну ячейку дампа памяти с нулевым содержимым.

Поэтому копируем адрес следующей строки дампа памяти и заменяем адрес, которого передается через регистр r8. После чего ставим курсор на пустую ячейку дампа памяти, нажимаем правую кнопку мышки и выбираем пункты Двоичные операции / Редактировать. В появившемся окне в строке ASCII пишем текст текст (рис. 8.19).

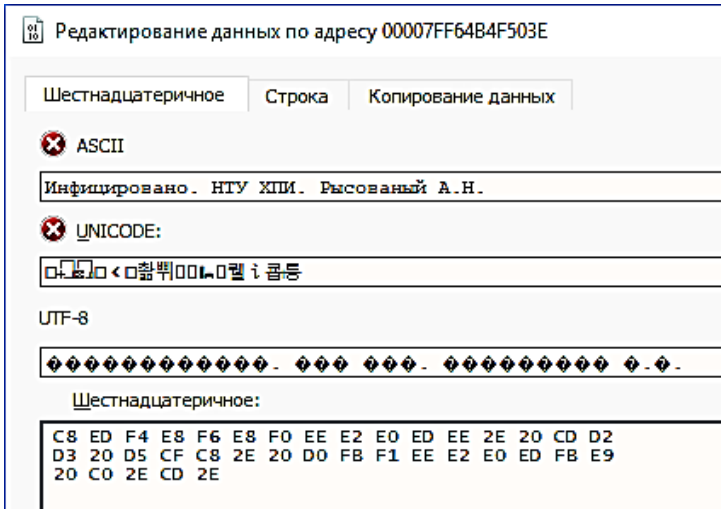


Рис. 8.19. Редактирование данных титула окна

13. Сохраняем все изменения.

Для этого можно выбрать пиктограмму с названием Исправления, нажать на кнопку Исправить файл, а далее новое имя и расширение exe.

После сохранения файла под новым именем с расширением exe выведется окно сообщения (рис. 8.20).

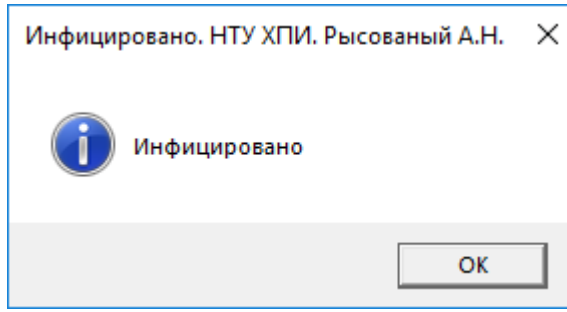


Рис. 8.20. Результат роботы программы

Добавление новой секции и внедрение в нее успешно осуществлено.

8.3. Лабораторная работа «Добавление новой секции»

Цель занятия: приобрести практические навыки исправления ошибок в программном обеспечении, которое находится в эксплуатации;

создание новой функциональности, используя неявное в эксплуатации программное обеспечение для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна (MessageBox) и отладчика;

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Постановка задачи

Выполнить задачу в несколько этапов:

1. Написать программу в среде masm64 в соответствии с заданием. В начале программы осуществить проверку возможности микропроцессора поддерживать команды AVX и AVX2. О чем вывести функциями MessageBox соответствующие сообщения. В конце программы вывести полученные результаты через функцию MessageBox, а также номер варианта, название лабораторной работы, шифр группы, фамилию исполнителя.

2. Произвести подсчет свободного места в секции кода.

3. Добавить новую секцию кода, в которую осуществить внедрение функции вывода сообщений MessageBox с выводом теста «Инфицировано. ФИО, номер группы». Этот вывод осуществить после вывода сообщения о проверке возможности микропроцессора поддерживать команды AVX и AVX2 и перед выводом упрощенного окна с основными результатами программы.

При использовании инструкций целесообразно воспользоваться справочниками, например https://www.felixcloutier.com/x86/index.html.

Задание.

1. *a√(bc) – df/e;*
2. *a/b + df√c;*
3. *bc/d – ef√a;*
4. *a/b/c + f√(de);*
5. *a + bc – df√e;*
6. *ab + c/d – f√e;*
7. *(a/b) + cdf√e;*
8. *ed/c – af√b;*
9. *√a + √b + cdf/e;*
10. *c√(de) + df/a;*
11. *ab/cd + f√e;*
12. *a/cd – √e + fb;*
13. *ce + √a + bdf;*
14. *a/b/c + f√(de);*
15. *√(ab) + cf/d – e;*
16. *af√(ce) – bd;*
17. *bd/c + √a – fe;*
18. *c√e + a/b + df;*
19. *de√b + cf/a;*
20. *a/e + bf√cd,*
21. *ab/c + df√e;*
22. *a/bc + f√(de);*
23. *abc√d – e√f;*
24. *ab√cd) – f/e;*
25. *de√c + bcf/a,*

где *a, b, c, d ,e, f* – числа, числа a є{a1,a2,a3,a4} заданы массивом.

Пример программы.

Программа 8.2. Числа *a* є{*a*1,*a*2,*a*3,*a*4} заданы массивом и имеют размерность Real8. Вычислить уравнение *a sqrt(b) + c/d*:

```
; masm64. Числа a є{a1,a2,a3,a4} заданы массивом
; и имеют размерность Real8. Вычислить уравнение a sqrt(b) + c/d

include \masm64\include64\masm64rt.inc
.data
mas1 dq 1.,2.,4.,9. ; массив чисел B
len1 equ ($-mas1)/type mas1
```

```
        mas2 dq 4.0,9.0,3.0         ; a, c, d
tit1 db "masm64. AVX. Результат вычисления уравнения a sqrt(b) + c/d.",0
        res dq len1 DUP(0),0  ;
        buf1 dd len1 DUP(0),0  ; буфер вывода сообщения
        ifmt db "masm64.  Массив bi = 1., 2., 4., 9.",10,
        9,"Числа: a, c, d  := 4., 9., 3.",10,
        "Результаты вычисления: %d ,%d ,%d ,%d ",10,10,
        "Автор: Рысованый А.Н., каф. ВТП, фак. КИТ, НТУ ХПИ",10,
        9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code               ; уравнение a sqrt(b) + c/d
entry_point proc
        mov rcx,len1
        lea rdx,res
        lea rbx,mas1
        vmovsd xmm1,mas2[0]; xmm1 - a - переслать real8
        vmovsd xmm2,mas2[8]  ; xmm2 - c
        vmovsd xmm3,mas2[16] ; xmm3 - d
        vdivsd xmm3,xmm2,xmm3 ; c/d

@@:
        vmovups xmm0,[rbx]              ; xmm0 - b
        vsqrtsd xmm4,xmm4,xmm0     ; sqrt(b)
        vcvttsd2si eax,xmm4             ; для отладки
        vmulsd xmm4,xmm4,xmm1     ; a sqrt(b)
        vaddsd xmm4,xmm4,xmm3     ; a sqrt(b) + c/d
        vcvttsd2si eax,xmm4             ; преобразование
        movsxd r15,eax ;
        mov [rdx],eax                   ; сохранение результата
        add rbx,8
        add rdx,8
        dec rcx
        jnz @b ; ссылка на предыдущую метку @@ (наверх)
        mov r10,res
        mov r11,res[8]
        mov r12,res[16]
        mov r13,res[24]
invoke wsprintf,addr buf1,addr ifmt,r10,r11,r12,r13
invoke MessageBox,0,addr buf1,addr tit1,MB_ICONINFORMATION
        invoke ExitProcess,0
        entry_point endp
        end
```

Результат выполнения программы приведен на рис. 8.21.

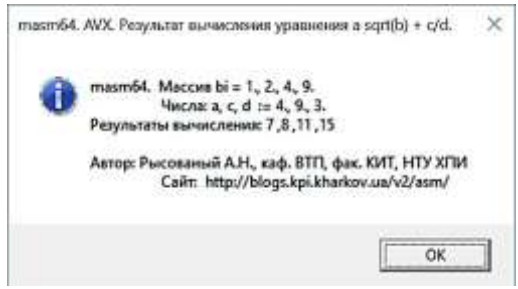


Рис. 8.21. Результат выполнения программы

Окно отладчика x64Dbg с кодом программы, куда предполагается внедрить код, представлено на рис. 8.22.

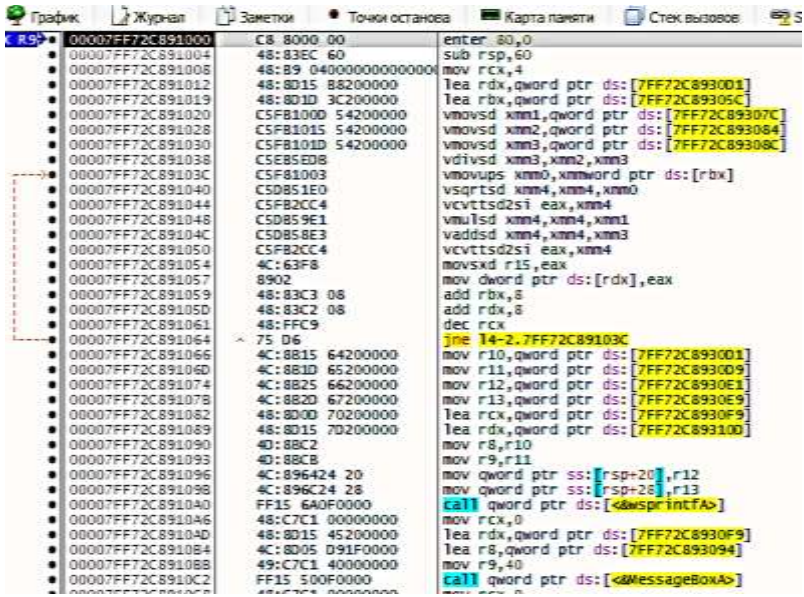


Рис. 8.22. Окно отладчика x64Dbg с кодом программы

Для оформления отчета по лабораторной работе необходимо использовать материал, который рассмотрен выше.

9. ВНЕДРЕНИЕ КОДА В РАЗЛИЧНЫЕ СЕКЦИИ ДАННЫХ

9.1. Общие сведения

Естественной записью кода программы является его запись в секцию кода с именем text. В другие секции код никогда не записывается. Спрятать код или его часть в другие секции можно только рукотворно и сделать это может только хакер.

Рассмотрим процесс внедрения кода к секции данных, хотя внедрение в другие секции отличается только предварительной настройкой свойств этих секций.

К секциям данных относятся: data, bss, rdata, idata, также можно отнести и секцию rsrc.

Секция data содержит инициализированные данные.

Секция bss содержит неинициализированные данные.

Секция rsrc содержит ресурсы модуля.

Секция rdata содержит каталог отладки.

Секция idata содержит информацию о функциях и данных, которые импортируются из других DLL.

Exe-файл, полученный в результате компиляции шаблона со строками:

```
.code
entry_point proc
 ret
entry_point endp
end
```

имеет уже 2 секции: text и rdata.

Рассмотрим программу 9.1.

Программа 9.1. Шаблон программы:

```
include \masm64\include64\masm64rt.inc
.code
entry_point proc
  invoke ExitProcess,0
  entry_point endp
  end
```

При компиляции программы 9.1 даже с одной функцией ExitProcess полученный exe-файл имеет уже 4 секции: text, rdata, data и reloc. Секция reloc содержит таблицу базовых поправок.

Каждая секция имеет минимальный объем 200h. Секция text увеличивается в зависимости от величины кода с приращением 200h.

Естественно, что общий размер exe-файла суммарно складывается из этих объемов. На практике все секции имеют неиспользуемые ячейки. Самые продвинутые антивирусы должны проверять аттрибуты (свойства) всех секций и если они отличаются от исходных, то такой файл должен признаваться зараженным. Но в учебных целях можно попытаться осуществить внедрение кода и в них.

По умолчанию можно внедрять код только в секцию кода. Секция кода имеет атрибуты:

– исполняемый файл;

– для чтения;

– содержит код,

которые показаны на рис. 9.1.

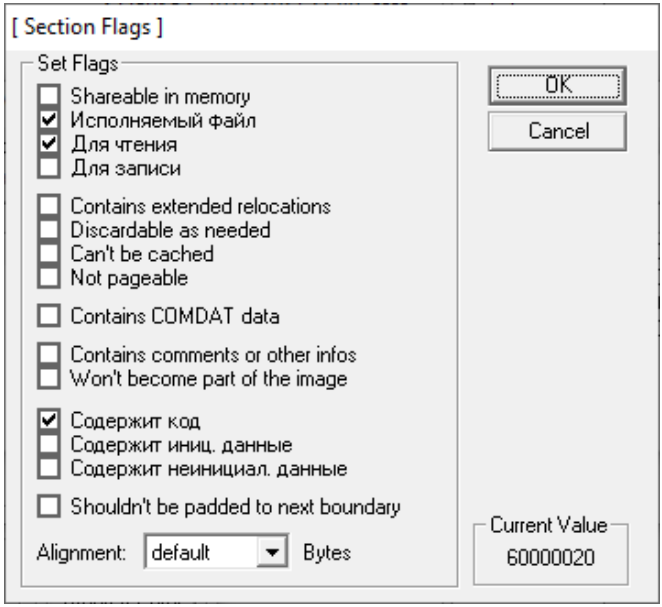


Рис. 9.1. Окно свойств секции кода

В программе отсутствуют данные, поэтому и на рис. 9.1 признаки использования данных отсутствуют.

Для того, чтобы в свободные ячейки этих секций можно было внедрять код, необходимо принудительно дать такое разрешение. Для этого воспользуемся возможностями программы анализатора PE-файла с названием LordPE.

9.2. Алгоритм внедрения кода в секции данных

Общий алгоритм внедрения кода в различные секции может быть следующим:

1. Исследование PE-файла для определения языка программирования (или компилятора), разрядности архитектуры (32/64), упаковщика или криптора. Естественно, если программа является собственной разработкой, то этот пункт опускается.
2. Подсчет числа свободных байтов в секции (секциях, в которые планируется осуществить внедрение).
3. Разрешение записи в секции при помощи программы-анализатора PE-файлов, например, с именем LordPE.
4. Написание программы для внедрения и получение ее exe-файла.
5. Работа в отладчике (отладчиках для каждого exe-файла).

5.1. Открыть исходный exe-файл в отладчике x64Dbg.

5.2. При помощи программы-анализатора включить признак разрешения записи в секции.

5.3. Перенести код в требуемые секции через кнопку меню Карта памяти с командами jmp для организации передачи управления между секциями.

5.4. Сохранить изменения.

При внедрении (переносе) кода в секции данных проще избегать использования ячеек памяти из других секций, т.к в этом случае не нужно следить за правильно установленными признаками секций, а также неоднократно копировать эти адреса.

Рассмотрим программу 9.2, которая выполняет уравнение ab и выводит кроме результата время выполнения этого уравнения. Необходимо вычислить уравнение ab + c/d + 32 путем внедрения недостающего кода в секции: rdata и data.

Одну часть уравнения внедрим в секцию rdata. А сложение константы 32 выполним в секции data.

Программа 9.2. Решение уравнения ab:

```
        include \masm64\include64\masm64rt.inc ;
.data
        a1 dq 1 ; резервирование в памяти 8 байтов для переменной
        b1 dq 2 ;
        res1 dq ?
titl db "Вывод через функцию MessageBox",0; назв. упрощенного окна
        st1  dq ?,0  ; буфер выведения сообщения.
   ifmt  db "Вывод чисел с памяти через MessageBox:",10,9,"ab",10,
   "Время выполнения  = %d тиков",10,"res = %d",10,10,
    "Автор программы: Рысованый А.Н., НТУ ХПИ, каф. ВТП",0 ;

.code ;
entry_point proc ;       уравнение = ab
        rdtsc           ; edx,eax
        shl rdx,32    ; смещение в свободную ст. часть
        add rdx,rax ; ст. и мл. части в одном регистре
        mov r15,rdx

        mov r10,a1     ; a:=1
        imul r10,r10,2 ; ab
        mov res1,r10

        rdtsc           ; edx,eax
         shl rdx,32   ; смещение в свободную ст. часть
         add rdx,rax ; ст. и мл. части в одном регистре
         sub rdx,r15 ;
         mov r15,rdx ;

invoke wsprintf,ADDR st1,ADDR ifmt,r15,res1
invoke MessageBox,0,addr st1,addr titl,MB_ICONINFORMATION;
invoke ExitProcess,0 ;возвращение упр. ОС и освобожд. ресурсов
        entry_point endp
        end
```

Для внедрения кода с вычислением недостающих частей уравнения напишем программу 9.3.

Программа 9.3. Решение уравнения c/d + 32:

```
        include \masm64\include64\masm64rt.inc ;
.code ;
entry_point proc ;  c/d + 32
        mov rax,7  ; c:=1
        mov r11,2  ; d:=2
        xor rdx,rdx
```

```
        div r11        ; c/d
        add rax,32
        ret
   entry_point endp
        end
```

Такой exe-файл необходим для того, чтобы увидеть представление в отладчике кода и копировать его для вставки в исходный exe-файл.

В общем случае последовательность выполнения программы с кодом внедрения в две секции с некоторыми допущениями может быть такой:

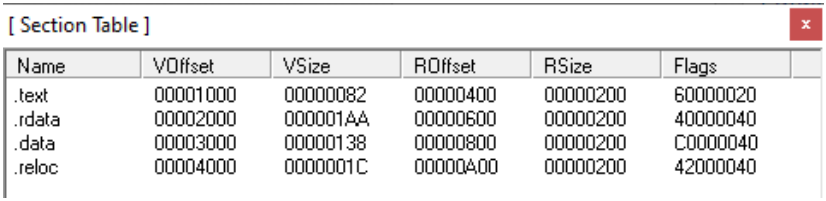
```
        ;…
        mov r10,a1     ; a:=1
        imul r10,r10,2 ; ab
        jmp rdata
 end: rdtsc ;
        shl rdx,32 ;
        ;…..

rdata:
        mov res1,r10
        mov rax,7  ; c:=1
        mov r11,2  ; d:=2
        xor rdx,rdx
        div r11       ; c/d
        jmp data

        ;…
data:
        add rax,32
        add res1,rax; для соблюдения логики работы
        jmp end
```

Разрешения на запись в секции rdata выполним при помощи программы-анализатора PE-файл с именем LordPE. Для этого откроем окно с секциями (рис. 9.2).

Выделим левой клавишей мышки секцию rdata. Нажмем правую клавишу мышки и выберем пункт edit section header и в появившемся окне свойств секции (рис. 9.3) напротив строки с именем Flags нажмем на три кнопки.



Name	VOffset	VSize	ROffset	RSize	Flags
.text	00001000	00000082	00000400	00000200	60000020
.rdata	00002000	000001AA	00000600	00000200	40000040
.data	00003000	00000138	00000800	00000200	C0000040
.reloc	00004000	0000001C	00000A00	00000200	42000040

Рис. 9.2. Окно секций в программе LordPE

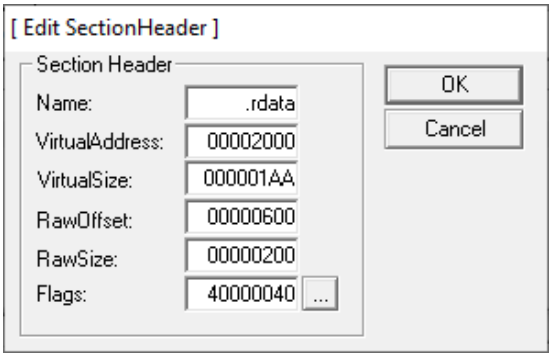


Рис. 9.3. Окно свойств секции программы LordPE

В окне установки признаков секции выделим признаки Исполняемый файл и Содержит код (рис. 9.4). Перевод на русский язык выполнен в программе Resource Hacker и только тех пунктов, которыми пользуемся наиболее часто.

Разрешения на запись в секцию data выполним аналогично секции text.

Переключение просмотра содержимого секций будем осуществлять через кнопку меню Карта памяти отладчика x64dbg.

Откроем в отладчике исходную программу, в которую необходимо осуществить внедрение (рис. 9.5).

Вставляем команду jmp на передачу управления из секции кода в секцию rdata. Для команды jmp выбираем строку кода

```
mov qword ptr ds:[7FF78EDD307C],r10
```

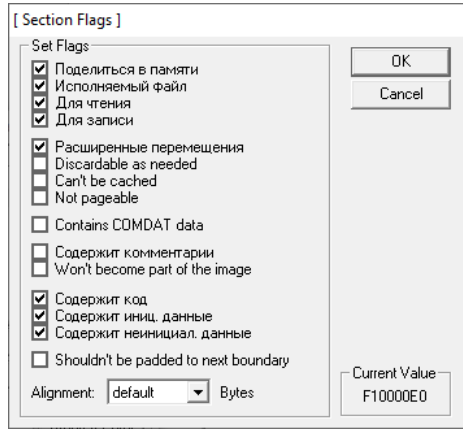


Рис. 9.4. Окно установки признаков секции программы LordPE

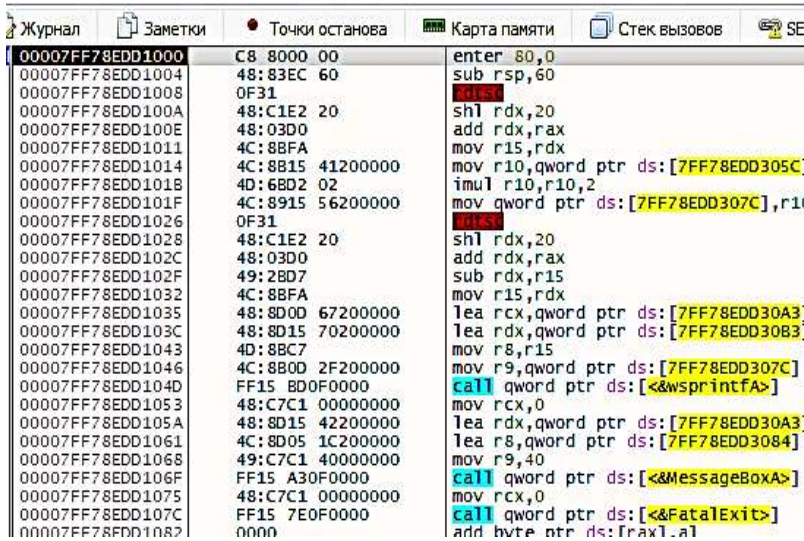


Рис. 9.5. Окно исходной программы в отладчике

Для этого, сначала сохраним эту команду в буфере. Чтобы записать команду в буфер необходимо выделить строку левой клавишей мышки, нажать правую клавишу мышки и выбрать пункты Копировать /

Дизассемблированный код или проще – нажать клавишу Пропуск и нажать комбинацию CTRL+C.

Команду, которая находится в буфере памяти необходимо перенести в секцию с именем rdata.

Для этого выбираем строку основного меню и в ней пункт Карта памяти, после чего откроется окно (рис. 9.6).

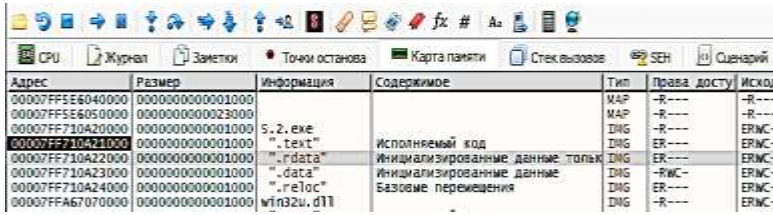


Рис. 9.6. Окно меню Карта памяти в отладчике x64Dbg

Дважды нажимаем левую клавишу мышки на строке с именем секции rdata. Открывается секция.Ищем в конце этой секции начало свободных ячеек.

Выделяем в секции rdata свободную строку, нажимаем клавишу Пробел, а затем комбинацию клавиш Ctrl+V. Строка вставлена.

Пока открыта секция rdata скопируем адрес этой строки в буфер. Для этого выделяем эту строку, нажимаем правую клавишу мышки и выбираем пункты меню Копировать / Адрес.

Теперь надо удалить такую же встроку в секции text. Для выбора секции нажимаем пункт Карта памяти в основном меню, дважды нажимаем на правую клавишу мышки на строке с именем секции text. Открывается выбранная секция. Выделяем строку, которую хотим удалить, нажимаем на ней клавишу Пробел, в открывшемся окне пишем команду jmp, делаем пробел и вставляем адрес перехода на секцию rdata комбинацией Ctrl+V (рис. 9.7).

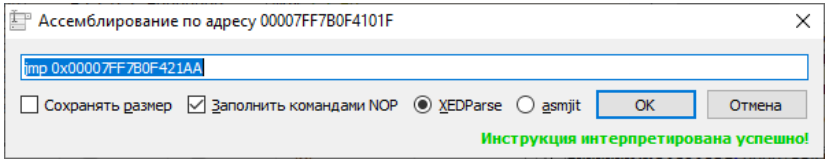


Рис. 9.7. Окно ассемблирования с вводом команды jmp

Открываем секцию rdata через пункт меню Карта памяти. Ищем вставленную нами команду и продолжаем переносить после нее код.

Не закрывая первый отладчик **запускаем еще один отладчик** с exe-файлом с кодом для внедрения (рис. 9.8).

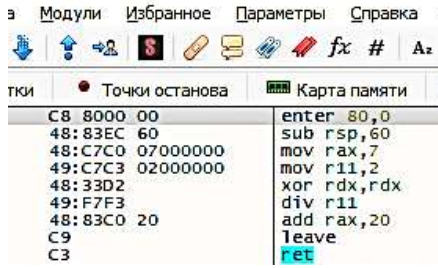


Рис. 9.8. Код внедрения в отладчике

По условию задачи необходимо код разбить на две секции.

Выделяем первую строку нашего кода для внедрения

```
mov rax,7
```

Нажимаем на этой строке Пробел и комбинацию клавиш Ctrl+C. Строка скопирована.

Переходим в отладчик, в который вставляем код и после строки

```
mov qword ptr ds:[7FF710A2307C],r10
```

производим выделение свободной. Нажимаем Пробел и в появившееся окно ассемблирования вводим команду из буфера комбинацией Ctrl+V. Подтверждаем действия клавишей OK.

При помощи аналогичной последовательности действий переносим все строчки, кроме последней. Эту последнюю строку перенесем в еще одну секцию – секцию data (рис. 9.9).

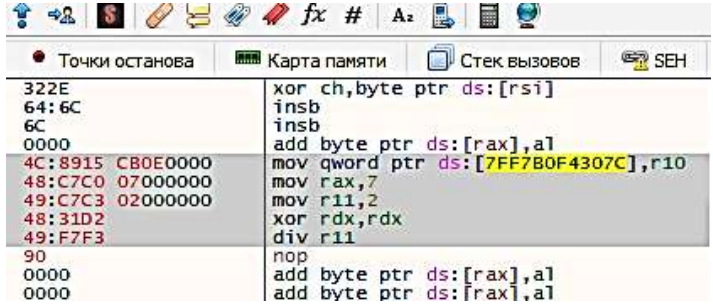


Рис. 9.9. Код внедрения в секцию rdata

Этот блок необходимо закончить вводом команды передачи управления jmp еще на одну секцию – секцию data. Для этого, необходимо найти и запомнить адрес начала свободных ячеек в секции data.

Переходим на секцию data.

Для этого выбираем кнопку меню Карта памяти, дважды нажимаем на строке с именем data. В этой секции ищем начало свободных ячеек. Запоминаем адрес свободной строки.

Переходим на секцию rdata и заканчиваем блок командой jmp перехода на новую секцию с адресом, который находится в буфере.

Результат внедрения в секцию rdata показан на рис. 9.10.

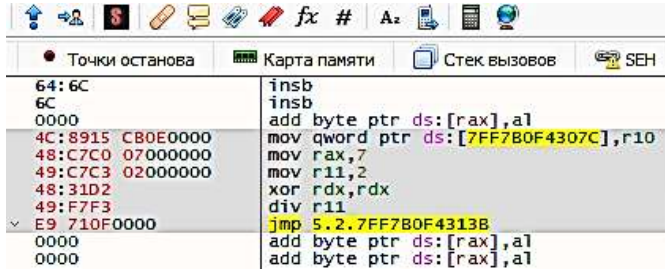


Рис. 9.10. Результат внедрения в секцию rdata

Теперь на свободном месте в секции data вставляем последнюю строку

```
add rax,20
```

Если после этой строки вставить команду jmp на секцию кода, то нарушается логика вывода результата. Ведь вывод осуществляется ячейки результата. Поэтому, надо исправить или параметр функции MessageBox, или дописать еще одну срочку

```
mov qword ptr ds:[0x00007FF7B0F4307C], rax
```

И для того, чтобы вернуть управление на завершение кода необходимо скопировать в буфер адрес возврата на продолжение кода в секции text.

Скопируем этот адрес и вставим последнюю команду возврата с адресом возврата (рис. 9.11).

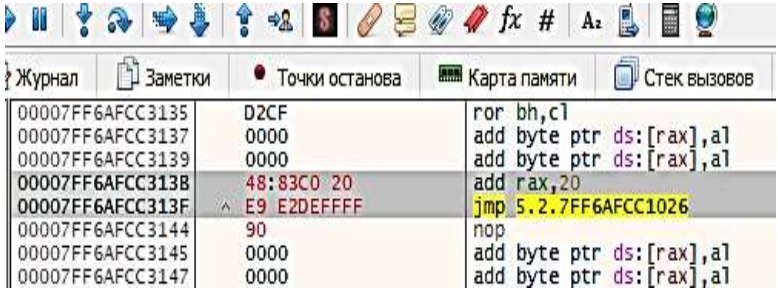


Рис. 9.11. Результат внедрения в секцию data

Сохраняем все изменения.

Для этого можно выбрать пиктограмму с названием Исправления, нажать на кнопку Исправить файл, а далее новое имя и расширение exe.

После сохранения файла под новым именем с расширением exe выведется окно сообщения (рис. 9.12).

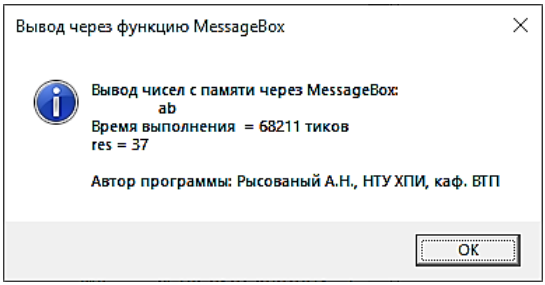


Рис. 9.12. Результат работы программы после внедрения

Добавление новой секции и внедрение в нее успешно осуществлено.

9.3. Создание программ с несколькими диалоговыми окнами

9.3.1. Создание программы с тремя диалоговыми окнами, одно из которых – окно ввода пароля

Программа с несколькими диалоговыми окнами за счет применения различных иконок улучшает эргономику, а за счет вынесения большой группы кнопок меню на другое диалоговое окно увеличивает информационность. При этом есть и недостаток – усложнение управления многодиалоговой программой.

Программа пользователя имеет только одно основное (родительское) окно. Остальные окна являются дочерними. Поэтому, попытка написать диалоговую программу с двумя и более родительскими окнами завершается безуспешной.

Процесс создания диалогового окна состоит из этапов:

1. Создание файла ресурсов;
2. Создание диалоговой программы;
3. Перенастройка (создание) командного (пакетного) файла.

Для создания файла ресурсов воспользуемся программой ResEd.

Рассмотрим последовательность создания файла ресурсов.

1. В программе ResEd создаем новый файл ресурсов (File – New Project и вводим имя). В правой верхней панели появляется значок папки и имя файла.

2. Нажимаем правой кнопкой мыши на значок папки и выбираем «Add Dialog».

3. Выставляем характеристики окна:

- выставляем мышкой необходимый размер окна;
- в поле свойств окна Caption меняем название диалогового окна.

4. Для размещения **картинки** (в нашем случае иконки) выбираем кнопку и перетаскиваем ее мышкой на диалоговое окно.

5. Для размещения **кнопки** выбираем в поле инструментов кнопку с изображением ОК и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим название кнопки.

6. Для написания **текста** на окне выбираем в поле инструментов кнопку с изображением буквы «А» и перетаскиваем ее мышкой на выбранное место. В поле Caption вводим текст.

7. Для создания **рамки** (если необходимо) выбираем поле с изображением рамки. В поле Caption вводим название рамки.

8. Добавляем файл ресурсов.

Добавление файла констант. Для этого, нажимаем правой кнопкой мыши по значку папки, выбираем «Include file», затем «Add». Выбираем в окне программы три точки и вводим путь, например C:\masm64\include64\RESOURCE.H.

9. Сохраняем программу.

10. Открываем созданный rc-файл и добавляем строку с ресурсом курсора:

10 ICON DISCARDABLE " sent.ico”

В этой строке число **10** обозначает цифровой идентификатор иконки (ресурс). Он может быть любым, но и в rc- и asm-файлах он должен быть одним и тем же.

В результате выполненных действий первое диалоговое окно программы должно быть таким, как приведено на рис. 9.13.

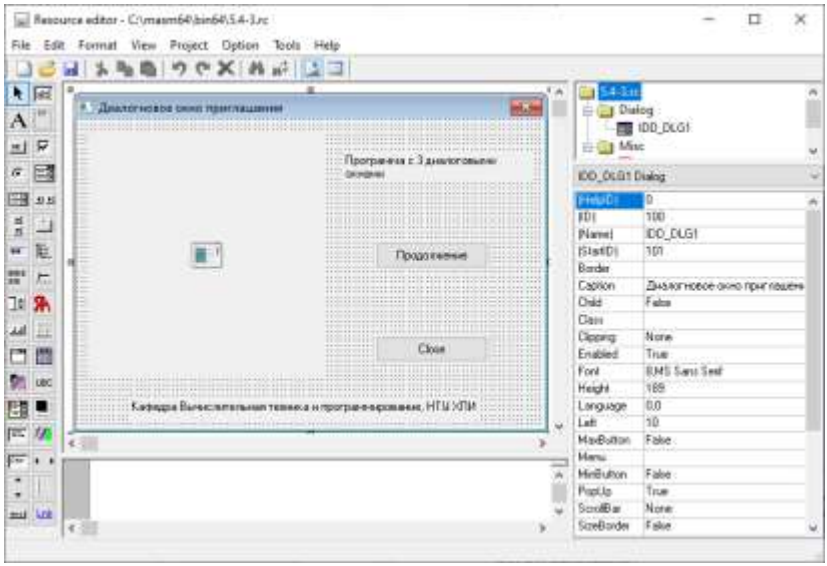


Рис. 9.13. Результат создания первого окна в программе ResEd

11. Создаем **второе диалоговое** окно. Для этого выделяем название файла и после нажатия правой клавишей мышки выбираем пункт ADD / Dialog.

Выставляем характеристики окна, создаем место для иконки и добавляем необходимые элементы окна, аналогично первому окну.

12. Создаем **третье диалоговое окно** по аналогии с первым.

Кроме аналогичных инструментов, размещенных в первых двух диалоговых окнах, в этом окне размещен **календарь**, который выбран из левой части программы из панели инструментов. Идентификатор календаря является статическим и в программе не обрабатывается.

13. Сохраняем все изменения с присвоением имени и расширением rc.

В результате проведенных действий получается программа 9.4.

Программа 9.4. Файл ресурсов rc:

```
#define IDD_DLG1 100
#define IDC_IMG1 102
#define IDC_BTN1 103
#define IDC_BTN2 104
#define IDC_STC1 105
#define IDC_STC2 106
#define IDC_STC3 101
#define IDD_DLG2 200
#define IDC_IMG2 202
#define IDC_BTN3 205
#define IDC_BTN4 204
#define IDC_BTN5 203
#define IDC_GRP1 207
#define IDC_BTN6 206
#define IDC_STC4 201
#define IDC_GRP3 208
#define IDD_DLG3 300
#define IDC_IMG3 302
#define IDC_BTN8 303
#define IDC_BTN9 305
#define IDC_BTN10 306
#define IDC_GRP2 307
#define IDC_STC5 301
#define IDC_STC6 304
#include "C:/masm64/include64/Resource.h"

10 ICON DISCARDABLE "21-001.ico"
11 ICON DISCARDABLE "21-002.ico"
12 ICON DISCARDABLE "21-003.ico"

IDD_DLG1 DIALOGEX 10,10,327,189
CAPTION "Диалоговое окно приветствия.  Окно 1."
FONT 8,"MS Sans Serif",0,0,0
STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
```

```
        BEGIN
    CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,9,171,153
    CONTROL "Продолжение",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,216,108,84,18
    CONTROL "EXIT",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,216,138,84,18
    CONTROL "Кафедра <Вычислительная техника и программирование>",
IDC_STC1,"Static",WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,189,15,129,18
    CONTROL "Курсовой проэкт <Разработка...>", IDC_STC2, "Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,189,57,132,33
    CONTROL "учебная группа КИТ-...,  ...",IDC_STC3,"Static",
WS_CHILDWINDOW|WS_VISIBLE,96,171,147,12
        END

        IDD_DLG2 DIALOGEX 10,10,336,180
        CAPTION "Требования к программе. Диалоговое окно 2."
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
    CONTROL "",IDC_IMG2,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,3,180,162
    CONTROL "К интерфейсу",IDC_BTN3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,105,84,18
    CONTROL "Основные",IDC_BTN4,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,75,84,18
    CONTROL "Назначение",IDC_BTN5,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,222,21,84,18
    CONTROL "ТРЕБОВАНИЯ",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,216,57,108,75
    CONTROL "Продолжение",IDC_BTN6,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,141,84,18
    CONTROL "уч. группа КИТ-...  ...",IDC_STC4,"Static",
WS_CHILDWINDOW|WS_VISIBLE,180,168,150,9
    CONTROL "Назначение программы",IDC_GRP3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,210,6,108,39
        END

        IDD_DLG3 DIALOGEX 10,10,315,183
        CAPTION "Расчеты. Диалоговое окно 3"
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
    CONTROL "",IDC_IMG3,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,9,174,150
```

```
    CONTROL "P(x)=5mod3",IDC_BTN8,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,63,84,18
    CONTROL "?",IDC_BTN9,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,114,84,18
    CONTROL "Выход",IDC_BTN10,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,138,84,18
    CONTROL "degP(x)",IDC_GRP2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,192,48,105,51
    CONTROL "учебная группа КИТ-...,  ...",IDC_STC5,"Static",
WS_CHILDWINDOW|WS_VISIBLE,159,168,150,12
    CONTROL "Кафедра <Вычислительная техника и программирование>",
IDC_STC6,"Static",WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,186,9,120,27
      END
```

Диалоговая программа представлена в файле 9.5.

Программа 9.5. Файл asm:

```
  include \masm64\include64\masm64rt.inc
.data?
     hInstance dq ?
     hIcon     dq ?
     hIcon3    dq ?
     hImg1     dq ?
     hImg3 dq ?;
     tEdit     dq ?
     buffer db 8 dup(?);  BYTE 64 dup(?);;;
.data
      pbuf dq buffer,0 ;;;
szPas db "12345",0 ; пароль
szStr1 db "Вы ввели корректный пароль. Поздравляем!",0
szStr2 db "Вы ввели неправильный пароль.",0
titl db "программа проверки пароля",0

.code
entry_point proc
     mov hInstance, rvcall(GetModuleHandle,0)
mov hIcon, rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)
     invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
      .exit
entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,
lParam:QWORD
       .switch uMsg
```

```
            .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку в
строке заголовка
        mov hImg1, rvcall(GetDlgItem,hWin,102)
rcall SendMessage,hImg1,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области
            .case WM_COMMAND
              .switch wParam
                .case 103    ; кнопка <Ввод текста>
mov hIcon,rv(LoadImage,hInstance,11,IMAGE_ICON,128,128,
LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,200,0,ADDR GetUserText,hIcon
                  test rax,rax  ; нажата ли кнопка отмены
                  jz notext
rcall MessageBox,hWin,pbuf,"Пользовательский текст",
MB_ICONQUESTION

        ; проверка пароля
        lea r14,pbuf; для того, чтобы убрать лишний служебный символ
        mov r15,[r14]
invoke lstrcmpi,addr szPas,r15;addr pbuf ; сравнение строк символов
          .if rax==0
        invoke MessageBox,0,addr szStr1,addr titl,MB_ICONINFORMATION
          .else
         invoke MessageBox,0,addr szStr2,addr titl,MB_ICONERROR
        jmp exit_dialog
          .endif

mov hIcon3,rv(LoadImage,hInstance,12,IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,300,0,ADDR Dial3,hIcon3
        notext:
                .case 101   ; кнопка <Close>
                  jmp exit_dialog
              .endsw
            .case WM_CLOSE
        exit_dialog:
              rcall EndDialog,hWin,0
          .endsw
          xor rax, rax
          ret
        main endp

GetUserText proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,
lParam:QWORD
```

```
        .switch uMsg
            .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; иконка в строке
заголовка
          mov hImg1, rvcall(GetDlgItem,hWin,205); извлекает дескриптор
rcall SendMessage,hImg1,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области
              mov tEdit,rvcall(GetDlgItem,hWin,201)
        invoke SetFocus,tEdit ; установка курсора в поле ввода

          .case WM_COMMAND

          .switch wParam
            .case 202 ; кнопка OK
rcall GetWindowText,tEdit,pbuf,sizeof buffer; записать текст по адресу
буфера
                  .if rax == 0
rcall MessageBox,hWin,"Введите текст или нажмите Cancel","Текст
отсутствует", MB_ICONWARNING
                    rcall SetFocus,tEdit ; установка фокуса на окно Edit
                  .else
                    rcall EndDialog,hWin,1
                  .endif
            .case 203 ;
                  jmp exit_dialog
          .endsw
            .case WM_CLOSE
        exit_dialog:
            rcall EndDialog,hWin,0
        .endsw
          xor rax, rax
          ret
        GetUserText endp

        Dial3 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,
lParam:QWORD

        .switch uMsg
            .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку в
строке заголовка
        mov hImg3,rvcall(GetDlgItem,hWin,302)
rcall SendMessage,hImg3,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области
```

```
        .case WM_COMMAND

      .switch wParam
        .case 303 ;
invoke MsgboxI,0,"Рысованый А.Н., каф. ВТП","Автор",MB_OK,13

        .case 304 ;
              jmp exit_dialog3
        .endsw

          .case WM_CLOSE
      exit_dialog3:
            rcall EndDialog,hWin,0
      .endsw
        xor rax, rax
        ret

      Dial3 endp
        end
```

В программе используется три API-функции **DialogBoxParam.**

Первая API-функции **DialogBoxParam** создает родительское диалоговое окно с идентификатором 100 и вызывает процедуру **main** обработки сообщений своего окна.

В процедуре main при нажатии на кнопку с идентификатором 103 кроме действий, связанных с нажанием на эту кнопку, вызывается вторая API-функции **DialogBoxParam** создания второго диалогового окна с идентификатором 200 и именем GetUserText.

Третье диалоговое окно также вызывается из процедуры **main**, т.к. это процедура родительского окна и в ней располагается последовательность вызова всех окон. Третье окно вызывается API-функцией **DialogBoxParam** после успешной проверки пароля и вывода соответствующего сообщения функцией MessageBox. Третья API-функция **DialogBoxParam** имеет идентификатор 300 и имя процедуры обработки сообщений Dial3.

Таким образом, за создание дочерних окон отвечает процедура родительского окна.

Т.о., при вызове exe-файла выводится главное (родительское) окно программы (рис. 9.14).

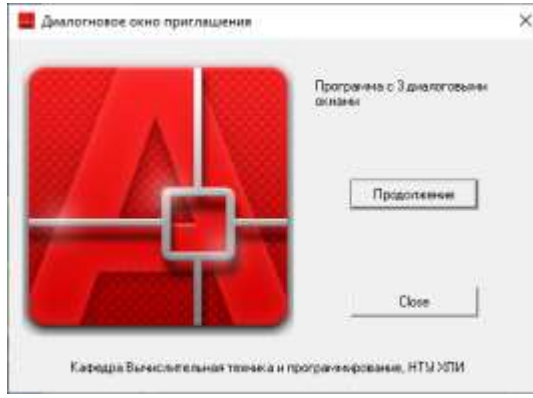


Рис. 9.14. Первое диалоговое окно программы

При нажатии на кнопку Продолжение выводится окно (рис. 9.15)

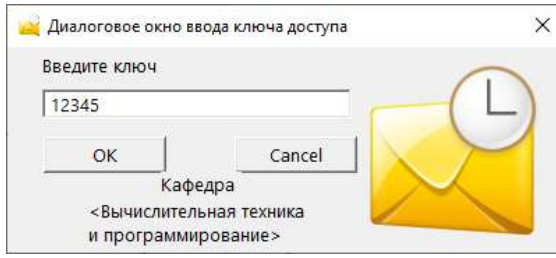


Рис. 9.15. Второе диалоговое окно программы

При вводе правильного пароля и нажатии на кнопку ОК выводится сообщение функцией MessageBox с отображением введенных символов (рис. 9.16).

Рис. 9.16. Окно с отображением введенных символов

При нажатии кнопки ОК выводится сообщение (рис. 9.17)

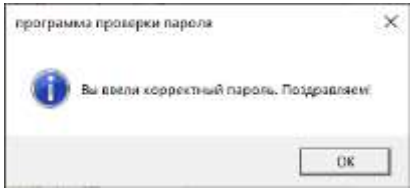


Рис. 9.17. Окно с сообщением

После ввода символа ОК выводится третье диалоговое окно (рис. 9.18)

Рис. 9.18. Третье диалоговое окно

При проверке пароля за сравнение двух строк символов отвечает API-функция **lstrcmpi.** Конструкция программы такова, что из-за применения строк кода

```
buffer db 8 dup(?);
pbuf dq buffer,0
```

при проверке строк в буфер записывается служебный символ. Его было обнаружено при первичной отладке в отладчике. Если этот символ заранее не убрать, то успешное сравнение не происходит. Поэтому были применены строки кода

```
lea r14,pbuf; для того, чтобы убрать лишний служебный символ
mov r15,[r14]
```

При применении другого способа записи в буфер этого символа быть не должно. Например, успешное сравнение происходит при консольном вводе и последующем сравнении.

Естественно, что применение функции сравнения строк резко помогает хакеру обнаружить место анализа пароля. Для этого лучше применять команды не непосредственного сравнения (команда cmp), а одну из команд sub, test, and.

При нажатии на кнопку Автор выводится сообщение (рис. 9.19).

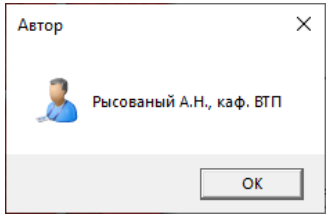


Рис. 9.19. Окно вывода информации об авторе программы

За вывод этого сообщения отвечает макрос Msgboxl, основой которого является API-функция MsgBoxIndirect, последним параметром которого есть числовой идентификатор:

```
invoke Msgboxl,0,"Рысованый А.Н., каф. ВТП","Автор",MB_OK,13
```

Этот идентификатор описывается в файле ресурсов программы.

Процедура обработки сообщений 3-го окна:

```
Dial3 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,
lParam:QWORD
.switch uMsg
     .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку
в строке заголовка
mov hImg3,rvcall(GetDlgItem,hWin,302)
rcall SendMessage,hImg3,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области

.case WM_COMMAND ; если есть системное сообщение от кнопок
.switch wParam
  .case 303          ; кнопка Автор
invoke Msgboxl,0,"Рысованый А.Н., каф. ВТП","Автор",MB_OK,13
  .case 304          ; кнопка Exit
```

```
            jmp exit_dialog3
    .endsw
        .case WM_CLOSE; если есть системное сообщение о выходе
exit_dialog3:
        rcall EndDialog,hWin,0
.endsw
    xor rax, rax
    ret
Dial3 endp
```

Процедуры обработки сообщений отличаются количеством обработки сообщений от кнопок.

Естественно, чем меньше выводится сообщений, тем сложнее хакеру найти место анализа пароля и осуществить поиск правильного пароля.

9.3.2. Создание программы с последовательным вызовом диалоговых окон

Общий алгоритм программы с последовательным вызовом диалоговых окон состоит в том, что из предыдущего окна при выборе одной из кнопок управления вызывается последующее диалоговое окно.

Процесс создания диалогового окна состоит из этапов:

1. Создание файла ресурсов;
2. Создание диалоговой программы;
3. Перенастройка (создание) командного (пакетного) файла.

Для создания файла ресурсов воспользуемся программой ResEd. Детальный процесс создания файла ресурсов описан в пункте 9.3.1.

Файл ресурсов представлен в программе 9.6.

Программа 9.6. Файл ресурсов rc:

```
#define IDD_DLG1 100
#define IDC_IMG1 102
#define IDC_BTN1 103
#define IDC_BTN2 104
#define IDC_STC1 105
#define IDC_STC2 106
#define IDC_STC3 101
#define IDD_DLG2 200
#define IDC_IMG2 202
```

```
        #define IDC_BTN3 205
        #define IDC_BTN4 204
        #define IDC_BTN5 203
        #define IDC_GRP1 207
        #define IDC_BTN6 206
        #define IDC_STC4 201
        #define IDC_GRP3 208
        #define IDD_DLG3 300
        #define IDC_IMG3 302
        #define IDC_BTN8 303
        #define IDC_BTN9 305
        #define IDC_BTN10 306
        #define IDC_GRP2 307
        #define IDC_STC5 301
        #define IDC_STC6 304
#include "C:/masm64/include64/Resource.h"

        10 ICON DISCARDABLE "21-001.ico"
        11 ICON DISCARDABLE "21-002.ico"
        12 ICON DISCARDABLE "21-003.ico"

        IDD_DLG1 DIALOGEX 10,10,327,189
        CAPTION "Диалоговое окно приветствия.  Окно 1."
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
    CONTROL "",IDC_IMG1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,9,171,153
    CONTROL "Продолжение",IDC_BTN1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,216,108,84,18
    CONTROL "EXIT",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,216,138,84,18
    CONTROL "Кафедра <Вычислительная техника и программирование>",
IDC_STC1,"Static",WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,189,15,129,18
    CONTROL "Курсовой проект <Разработка...>",IDC_STC2,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,189,57,132,33
    CONTROL "учебная группа КИТ-...,  ...",IDC_STC3,"Static",
WS_CHILDWINDOW|WS_VISIBLE,96,171,147,12
        END

        IDD_DLG2 DIALOGEX 10,10,336,180
        CAPTION "Требования к программе. Диалоговое окно 2."
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
```

```
        CONTROL "",IDC_IMG2,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,3,180,162
        CONTROL "К интерфейсу",IDC_BTN3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,105,84,18
        CONTROL "Основные",IDC_BTN4,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,75,84,18
        CONTROL "Назначение",IDC_BTN5,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,222,21,84,18
        CONTROL "ТРЕБОВАНИЯ",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,216,57,108,75
        CONTROL "Продолжение",IDC_BTN6,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,225,141,84,18
        CONTROL "уч. группа КИТ-...  ...",IDC_STC4,"Static",
WS_CHILDWINDOW|WS_VISIBLE,180,168,150,9
        CONTROL "Назначение программы",IDC_GRP3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,210,6,108,39
      END

      IDD_DLG3 DIALOGEX 10,10,315,183
      CAPTION "Расчеты. Диалоговое окно 3"
      FONT 8,"MS Sans Serif",0,0,0
      STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
      BEGIN
        CONTROL "",IDC_IMG3,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,6,9,174,150
        CONTROL "P(x)=5mod3",IDC_BTN8,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,63,84,18
        CONTROL "?",IDC_BTN9,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,114,84,18
        CONTROL "Выход",IDC_BTN10,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,138,84,18
        CONTROL "degP(x)",IDC_GRP2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,192,48,105,51
        CONTROL "учебная группа КИТ-...,  ...",IDC_STC5,"Static",
WS_CHILDWINDOW|WS_VISIBLE,159,168,150,12
        CONTROL "Кафедра <Вычислительная техника и программирование>",
IDC_STC6,"Static",WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,186,9,120,27
      END
```

Диалоговая программа представлена в файле 9.7.

Программа 9.7. Файл asm:

```
include \masm64\include64\masm64rt.inc
.data?
   hInstance dq ?
```

```
        hIcon  dq ?
       hIcon3 dq ?
        hImg1  dq ?
       hImg3  dq ?;
    .data
avtor1 db "КІТ-...  ...",10,10, "Для вывода использован макрос MsgboxI",0
szTitle203 db "НАЗНАЧЕНИЕ ПРОГРАММЫ",0
szInf203 db "Программа разработана в рамках  работы по теме: ",10,
       "-...",10,10, "Программа позволяет рассчитывать ... ",10,
       " Автор программы: ..., ",10, " КІТ-..., кафедра ВТП",0
szTitle204 db "Основные требования к программе",0 ;
szInf204 db "- Получает результаты",10,
"- Все диалоговые окна имеют возможность закрытия  (см. далее);",0
szInf2042 db "- Выполняет сохранение результатов работы программы в
файл;",0;
       szTitle205 db "Дополнительные требования к интерфейсу окна",0 ;
szInf205 db "- Используется графический объект;",10,\
" - Используются авторские несистемные иконки;",10,
" - Для управления программой используются кнопки и статические
объекты.",0
       szTitle305 db "Краткие сведения",0 ;
szInf305 db "64-bit",10,10,"Расчет циклов генерирования и длин
циклов",10,10, "Версия 2021.05.01",10,10,
" Автор программы: ..., ",10,
" КІТ-...,",10,"кафедра <Вычислительная техника и программирование>", 0
       szTitle303 db "Выполнение расчетов",0 ;
       szInf303 db " Расчеты выполнены. Файл с результатами создан",0
       RunPx5 db "PolPx5mod3.exe",0

.code
       entry_point proc
        mov hInstance,rvcall(GetModuleHandle,0) ; идентификатор окна
 mov hIcon,rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
         .exit
       entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD, lParam:QWORD
          .switch uMsg
            .case WM_INITDIALOG
    rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку в
строке заголовка
    mov hImg1,rvcall(GetDlgItem,hWin,102)
    rcall SendMessage,hImg1,STM_SETIMAGE,IMAGE_ICON,lParam ;
иконка в клиентской области
```

```
        .case WM_COMMAND
              .switch wParam
               .case 103    ; кнопка <Продолжение>
mov hIcon,rv(LoadImage,hInstance,11,IMAGE_ICON,256,256, LR_DEFAULTCOLOR)
       invoke DialogBoxParam,hInstance,200,0,ADDR Dial2,hIcon
               .case 104   ; кнопка <EXIT>
                 jmp exit_dialog
               .endsw
             .case WM_CLOSE
       exit_dialog:
              rcall EndDialog,hWin,0
          .endsw
          xor rax, rax
          ret
       main endp

Dial2 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD, lParam:QWORD
       .switch uMsg
            .case WM_INITDIALOG
       rcall SendMessage,hWin,WM_SETICON,1,lParam ; иконка в строке
заголовка
         mov hImg1, rvcall(GetDlgItem,hWin,202); извлекает дескриптор
       rcall SendMessage,hImg1,STM_SETIMAGE,IMAGE_ICON,lParam ;
иконка в клиентской области
         .case WM_COMMAND
        .switch wParam
          .case 206    ; кнопка <Продолжение>
mov hIcon3,rv(LoadImage,hInstance,12,IMAGE_ICON,256,256,LR_DEFAULTCOLOR)
       invoke DialogBoxParam,hInstance,300,0,ADDR Dial3,hIcon3
          .case 203     ; кнопка <Назначение>
       invoke MsgboxI,0,ADDR szInf203,ADDR szTitle203,MB_OK,12
          .case 204     ; кнопка <Основные>
       invoke MsgboxI,0,ADDR szInf204,ADDR szTitle204,MB_OK,12
       invoke MsgboxI,0,ADDR szInf2042,ADDR szTitle204,MB_OK,12
          .case 205     ; кнопка <К интерфейсу>
       invoke MsgboxI,0,ADDR szInf205,ADDR szTitle204,MB_OK,12
          .endsw
        .case WM_CLOSE
              rcall EndDialog,hWin,0
       .endsw
          xor rax, rax
          ret
       Dial2 endp

Dial3 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD, lParam:QWORD
```

```
        .switch uMsg
            .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку в
строке заголовка
        mov hImg3,rvcall(GetDlgItem,hWin,302)
rcall SendMessage,hImg3,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области
          .case WM_COMMAND
        .switch wParam
          .case 303 ; кнопка P(x)=5
        invoke WinExec,ADDR RunPx5,SW_HIDE
invoke MessageBox,0,addr szInf303,addr szTitle303, MB_ICONINFORMATION
;invoke MsgboxI,0,ADDR avtor1,"Автор разработки програмы",MB_OK,12
          .case 305 ; кнопка <?>
        invoke MsgboxI,0,ADDR szInf305,ADDR szTitle305,MB_OK,12
          .case 306 ; кнопка <Выход>
                jmp exit_dialog3
          .endsw
            .case WM_CLOSE
        exit_dialog3:
              rcall EndDialog,hWin,0
        .endsw
           xor rax, rax
           ret
        Dial3 endp
           end
```

При разработке программы внимание уделяется на те идентификаторы из файла ресурсов, на которые программа должна реагировать.

В начале программы блок кода

```
        mov hInstance,rvcall(GetModuleHandle,0) ; идентификатор окна
 mov hIcon,rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,LR_DEFAULTCOLOR)
          invoke DialogBoxParam,hInstance,100,0,ADDR main,hIcon
```

получает номер (идентификатор) окна в системе и сохраняет в ячейке hInstance.

Вторая строка кода вызывает иконку, идентификатор которой в файле ресурсов и в тексте программы равен 10, указывается размер иконки 256 на 256.

Третья строка вызывает родительское окно с идентификатором 100, иконкой и передается управление процедуре с именем main.

В процедуре с именем main обработка выбора кнопок происходит в цикле при обработке системного сообщения **WM_COMMAND.**

Родительское окно представлено на рис. 9.20.

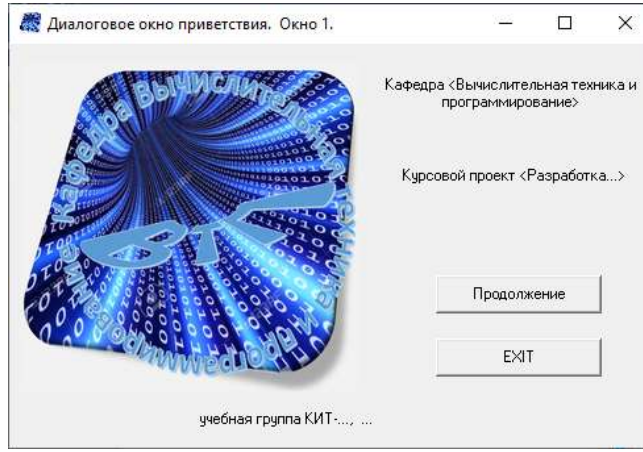


Рис. 9.20. Родительское окно программы

При нажатии на кнопку Продолжение выводится второе диалоговое дочернее окно (рис. 9.21).

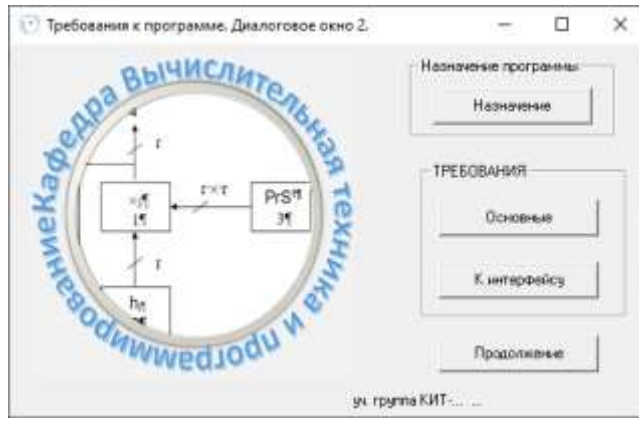


Рис. 9.21. Второе диалоговое окно программы

При нажатии на кнопку Продолжение во 2-м диалоговом окне

выводится третье диалоговое дочернее окно (рис. 9.22).

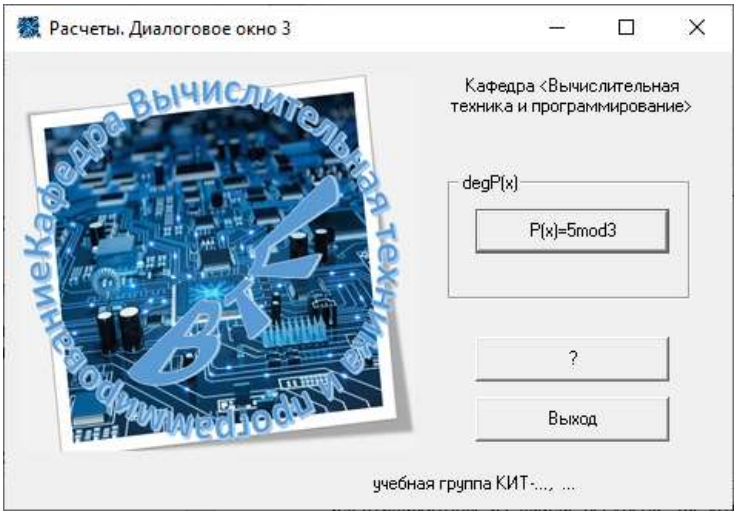


Рис. 9.22. Диалоговое окно 3

При выводе небольших сообщений с отображением несистемной иконки используется макрос MsgboxI.

9.3.3. Создание программы с параллельным вызовом диалоговых окон

Смысл параллельного вызова заключается в том, что все кнопки вызова дочерних окон расположены на родительском окне. Этот прием наиболее часто применяется при программировании простых задач.

Пример файла ресурсов представлен в программе 9.8.

Программа 9.8. Файл ресурсов rc:

```
#define IDD_DLG1 1000
#define IDC_IMG1 1001
#define IDC_BTN1 1002
#define IDC_BTN2 1003
#define IDC_GRP1 1004
#define IDC_BTN3 1005
#define IDC_BTN4 1006
```

```
        #define IDC_GRP2 1007
        #define IDC_STC1 1008
        #define IDC_STC2 1009
        #define IDC_BTN5 1010
        #define IDC_STC3 1011
        #define IDC_STC4 1012
        #define IDC_GRP3 1013
        #define IDC_STC5 1014
        #define IDC_BTN6 1015
        #define IDC_BTN7 1016
        #define IDD_DLG2 200
        #define IDC_IMG2 202
        #define IDC_STC6 203
        #define IDC_STC7 204
        #define IDC_STC8 205
        #define IDC_STC9 206
        #define IDC_STC10 207
        #define IDC_STC11 208
        #define IDC_STC12 209
        #define IDC_STC13 210
        #define IDC_STC14 211
        #define IDC_STC21 212
        #define IDC_STC22 213
        #define IDC_BTN8 220
        #define IDD_DLG3 300
        #define IDC_IMG3 302
        #define IDC_STC15 303
        #define IDC_STC16 304
        #define IDC_STC17 310
        #define IDC_STC18 311
        #define IDC_STC20 305
        #define IDC_BTN9 320
#include "C:/masm64/include64/Resource.h"

        10 ICON DISCARDABLE "icon1.ico"
        11 ICON DISCARDABLE "icon2.ico"
        12 ICON DISCARDABLE "icon3.ico"
        13 ICON DISCARDABLE "user.ico"

        IDD_DLG1 DIALOGEX 10,10,387,336
        CAPTION "Программа определения характеристик P(x)"
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
    CONTROL "",IDC_IMG1,"Static", WS_CHILDWINDOW|WS_VISIBLE
```

```
|SS_CENTERIMAGE|SS_ICON,9,66,213,180
    CONTROL "P(X)4mod3",IDC_BTN1,"Button", WS_CHILDWINDOW|
WS_VISIBLE|WS_TABSTOP,255,69,96,18
    CONTROL "P(X)10mod3",IDC_BTN2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,255,129,96,18
    CONTROL "Расчеты",IDC_GRP1,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,234,42,135,117,WS_E
X_CLIENTEDGE|WS_EX_DLGMODALFRAME
    CONTROL "Возможности осн.",IDC_BTN3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,255,189,93,18
    CONTROL "Возможности доп.",IDC_BTN4,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,255,225,93,18
    CONTROL "Интерфейс",IDC_GRP2,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,234,168,135,96,WS_E
X_CLIENTEDGE|WS_EX_DLGMODALFRAME
    CONTROL "кафедра ВТП, НТУ ХПИ ",IDC_STC1,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,24,270,129,24
    CONTROL "Определение характеристик
P(x):",IDC_STC2,"Static",WS_CHILDWINDOW|WS_VISIBLE,30,6,126,12,W
S_EX_DLGMODALFRAME
    CONTROL "О программе",IDC_BTN5,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,204,276,72,18
    CONTROL "- матриц состояний для degP(x)",IDC_STC3,"Static",
WS_CHILDWINDOW|WS_VISIBLE,3,36,225,9
    CONTROL "- длин циклов P(x);",IDC_STC4,"Static",
WS_CHILDWINDOW|WS_VISIBLE,6,21,195,9
    CONTROL "Автор",IDC_GRP3,"Button",
WS_CHILDWINDOW|WS_VISIBLE|BS_GROUPBOX,15,249,150,51
    CONTROL "Тема <...>",IDC_STC5,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,24,309,345,24
    CONTROL "P(X)9mod3",IDC_BTN6,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,255,99,96,18
    CONTROL "Выход",IDC_BTN7,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,306,276,72,18
      END

      IDD_DLG2 DIALOGEX 10,10,318,327
      CAPTION "Основные возможности интерфейса"
      FONT 8,"MS Sans Serif",0,0,0
      STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
      BEGIN
   CONTROL "",IDC_IMG2,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,69,12,195,153
    CONTROL "- получает результаты ;",IDC_STC6,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,174,276,9
```

```
    CONTROL "- получает P(x) ;",IDC_STC7,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,186,288,12
    CONTROL "- рассчитывает матрицы состояний P(x);",IDC_STC8,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,201,240,12
    CONTROL "- рассчитывает длину циклов P(x) ;",IDC_STC9,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,213,285,12
    CONTROL "",IDC_STC10,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,228,270,9
    CONTROL "- используются диалоговые окна;",IDC_STC11,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,228,288,12
    CONTROL "- используются собственные иконки;",IDC_STC12,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,243,261,12
    CONTROL "- результаты расчетов записываются в
файлы;",IDC_STC13,"Static", WS_CHILDWINDOW|WS_VISIBLE,6,258,270,12
    CONTROL "- выводится время расчетов.",IDC_STC14,"Static",
WS_CHILDWINDOW|WS_VISIBLE,9,270,231,15
    CONTROL "Кафедра",IDC_STC21,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,66,291,87,12
    CONTROL "Вычислительная техника и
программирование",IDC_STC22,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,0,309,228,9
    CONTROL "Выход",IDC_BTN8,"Button",
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,237,300,72,18
        END

        IDD_DLG3 DIALOGEX 10,10,306,294
        CAPTION "Дополнительные возможности интерфейса"
        FONT 8,"MS Sans Serif",0,0,0
        STYLE WS_VISIBLE|WS_OVERLAPPEDWINDOW
        BEGIN
    CONTROL "",IDC_IMG3,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTERIMAGE|SS_ICON,57,6,195,171
    CONTROL "- использует несистемные иконки;",IDC_STC15,"Static",
WS_CHILDWINDOW|WS_VISIBLE,12,186,249,12
    CONTROL "- использует статические объекты
ресурсов;",IDC_STC16,"Static",
WS_CHILDWINDOW|WS_VISIBLE,12,201,258,12
    CONTROL "Кафедра",IDC_STC17,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,81,246,57,18
    CONTROL "Вычислительная техника и
программирование",IDC_STC18,"Static",
WS_CHILDWINDOW|WS_VISIBLE|SS_CENTER,3,270,216,15
    CONTROL "- использует кнопки управления в основном диалоговом
окне.",IDC_STC20,"Static", WS_CHILDWINDOW|WS_VISIBLE,12,216,255,18
    CONTROL "Выход",IDC_BTN9,"Button",
```

```
WS_CHILDWINDOW|WS_VISIBLE|WS_TABSTOP,228,267,72,18
        END
```

В программе 9.9 представлена диалоговая программа.

Программа 9.9. Файл asm:

```
        include \masm64\include64\masm64rt.inc
.data?
          hInstance dq ?
          hIcon     dq ?
          hIcon2 dq ?
          hIcon3 dq ?
          hImg2  dq ?;
          hImg3  dq ?;
.data
        szRun4  db "PolPx4m3.exe",0
        szRun9  db "PolPx9m3.exe",0
        szRun10 db "PolPx10m3.exe",0

szTitle9 db "Выполнение расчетов",0
szInf9 db "Нажмите Ok  и примерно через 30 С ожидайте окончания
расчетов.",10,10,
"Размер файла расчетов: 551 124 байта.",0
szInf10 db "Нажмите Ok и ожидайте размера файла",10, "расчетов: 1 653
372 байта. ",10,10, "Время ожидания - до 7 минут.",0

        szTitle5 db "О программе",0
szInf5 db "Программа осуществляет:",10,10,"- расчет циклов P(x);",10,
        "- расчет длин циклов P(x)",10,10,"64-bit",10,"Версия 2021.11.01",0

        .code
        entry_point proc
         mov hInstance,rv(GetModuleHandle,0)
 mov hIcon,rv(LoadImage,hInstance,10,IMAGE_ICON,256,256,LR_DEFAULTCOLOR)
         invoke DialogBoxParam,hInstance,1000,0,ADDR main,hIcon
        invoke ExitProcess,0
           ret
        entry_point endp

main proc hWin:QWORD,uMsg:QWORD,wParam:QWORD,lParam:QWORD
        .switch uMsg
        .case WM_INITDIALOG
invoke SendMessage,hWin,WM_SETICON,1,lParam ; значок для диалога
invoke SendMessage,rv(GetDlgItem,hWin,1001),\ ; значок в клиентской обл.
                    STM_SETIMAGE,IMAGE_ICON,lParam
```

```
        .case WM_COMMAND
          .switch wParam
            .case 1002  ; кнопка <P(x)4mod3>
             invoke WinExec,ADDR szRun4,SW_HIDE
            .case 1015   ;  кнопка <P(x)9mod3>
        invoke MsgboxI,0,ADDR szInf9,ADDR szTitle9,MB_OK,11
            .case 1003   ;  кнопка <P(x)10mod3>
        invoke MsgboxI,0,ADDR szInf10,ADDR szTitle9,MB_OK,12
           .case 1005    ; кнопка <Возможности осн.>
mov hIcon2,rv(LoadImage,hInstance,11,IMAGE_ICON,256,256,LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,200,0,ADDR Dial2,hIcon2
           .case 1006   ; кнопка <Возможности доп.>
mov hIcon3,rv(LoadImage,hInstance,12,IMAGE_ICON,256,256,
LR_DEFAULTCOLOR)
        invoke DialogBoxParam,hInstance,300,0,ADDR Dial3,hIcon3
           .case 1010; Кнопка О программе
           invoke MsgboxI,0,ADDR szInf5,ADDR szTitle5,MB_OK,13
        .case 1016   ; кнопка <EXIT>
                jmp exit1
          .endsw
        .case WM_CLOSE
         exit1: invoke EndDialog,hWin,0 ;
         .endsw
           xor rax, rax
           ret
        main endp

        ; Окно 2 Осн. возм. интерфейса
Dial2 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD, lParam:QWORD
        .switch uMsg
             .case WM_INITDIALOG
        rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку
в строке заголовка
        mov hImg2,rvcall(GetDlgItem,hWin,202)
        rcall SendMessage,hImg2,STM_SETIMAGE,IMAGE_ICON,lParam ;
иконка в клиентской области
        .case WM_COMMAND
          .switch wParam
            .case 220
          jmp exit2
          .endsw
         .case WM_CLOSE
         exit2:
           rcall EndDialog,hWin,0
         .endsw
```

```
        xor rax, rax
        ret
    Dial2 endp

    ; Окно 3. Доп. возм. интерфейса
Dial3 proc hWin:QWORD,uMsg:QWORD,wParam:QWORD, lParam:QWORD
    .switch uMsg
        .case WM_INITDIALOG
rcall SendMessage,hWin,WM_SETICON,1,lParam ; установить иконку в
строке заголовка
    mov hImg3,rvcall(GetDlgItem,hWin,302)
rcall SendMessage,hImg3,STM_SETIMAGE,IMAGE_ICON,lParam ; иконка
в клиентской области
    .case WM_COMMAND ; если есть сообщение от кнопки
      .switch wParam
        .case 320
     jmp exit3
      .endsw

     .case WM_CLOSE      ; если есть сообщение о выходе
    exit3:
        rcall EndDialog,hWin,0
     .endsw
        xor rax, rax
        ret
    Dial3 endp
    end
```

Родительское окно представлено на рис. 9.23.

При выборе каждой кнопки из поля Интерфейс вызываются дочерние диалоговые окна (рис. 9.24 и 9.25). За вызовы этих окон отвечает функция DialogBoxParam. Функция DialogBoxParam создает модальное диалоговое окно из шаблона ресурса блока диалога.

Параметрами являются: дескриптор окна; идентификатор окна в файле ресурсов; 0 - дескриптор окна, которое владеет диалоговым окном; указатель на процедуру диалогового окна; идентификатор иконки.

Статическими элементами на диалоговых окнах являются надписи, для которых есть свои идентификаторы, но которые в текстовом файле не обрабатываются.

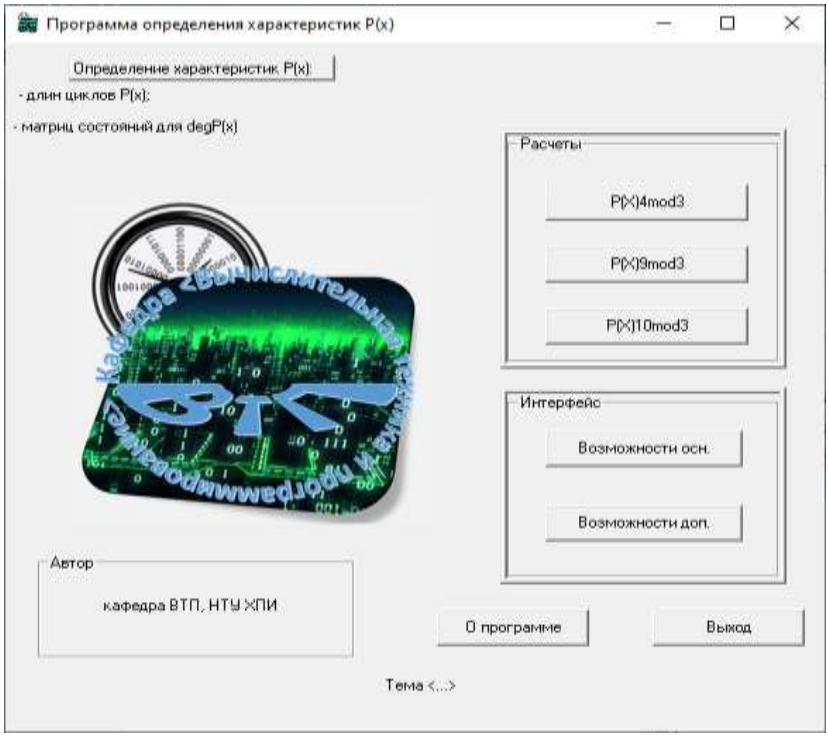


Рис. 9.23. Родительское диалоговое окно

Дочерние окна содержат по одной кнопке для выхода из них. Для этого во втором дочернем диалоговом окне применяется блок кода

```
.case WM_COMMAND
  .switch wParam
    .case 320
 jmp exit3
  .endsw
.case WM_CLOSE
exit3:
   rcall EndDialog,hWin,0
 .endsw
```

Если используется больше дочерних окон, чем в рассматриваемой программе, то использовать системное сообщение WM_CLOSE более двух раз по предложенному шаблону становится очень проблематично.

Рис. 9.24. Первое дочернее диалоговое окно

Рис. 9.25. Второе дочернее диалоговое окно

9.3.4. Создание нестандартного окна, представленного в виде BMP-файла

Стандартное окно имеет прямоугольное окно. Нестандартное окно имеет произвольную форму. При создании нестандартного окна, представленного в виде BMP-файла, файл ресурсов не используется.

Перед применением BMP-файла его необходимо подготовить. Для этого сначала необходимо фон залить черным цветом, а уже в программе его сделать прозрачным. Подготовленный файл представлен на рис. 9.26.

Рис. 9.26. Подготовленный BMP-файл

Программа 9.10. Файл asm:

```
include \masm64\include64\masm64rt.inc ; библиотеки
include \masm64\include64\winmm.inc ; для PlaySound
includelib \masm64\lib64\winmm.lib ;

.data
  BitMap db "pic11.bmp", 0
  width_ dq ?
  height dq ?
  xCtr dq ?
  yCtr dq ?
  pzsFiles db "meow.wav", 0
```

```
        AppName db "App", 0
        ClassName db "Class", 0
        isCaptures db 0
        hitResult dq ?
        hInstance HINSTANCE ?
        hBitmap HBITMAP ?

  .code
  entry_point proc
      mov hInstance,rv(GetModuleHandle,0) ; получение и сохр.
дескриптора программы
        invoke WinMain,hInstance,SW_SHOWDEFAULT
        invoke ExitProcess,rax ;
      entry_point endp

      WinMain proc hInst:HINSTANCE, CmdShow:DWORD
      LOCAL wc:WNDCLASSEX ; Резервирование стека под структуру
класса окна
      LOCAL msg:MSG ; резервирование стека под структуру MSG
      LOCAL hwnd:HWND ; резервирование стека под дескриптор окна

      pop wc.hInstance   ; Возвращение дескриптора в поле структуры
mov wc.cbSize,SIZEOF WNDCLASSEX ; Кол. байтов структуры
WNDCLASSEX
mov wc.style,CS_HREDRAW or CS_VREDRAW  ; Стиль и поведение окна
    lea rax,WndProc            ; Получение адреса процедуры WndProc
    mov wc.lpfnWndProc,rax             ; Адрес процедуры WndProc
    mov wc.hbrBackground,COLOR_WINDOW+1   ; Цвет окна
    invoke GetStockObject,WHITE_BRUSH     ; Чтение описателя кисти
    mov wc.hbrBackground,rax       ; Цвет заполнения окна
    mov wc.lpszMenuName,0          ; Имя ресурса меню
lea rax, ClassName      ; Получение адреса переменной с именем класса
      mov wc.lpszClassName,rax       ; Имя класса
invoke LoadIcon, NULL, IDI_APPLICATION ; Загрузка отображаемой
пиктограммы
      mov wc.hIcon, rax       ; Дескриптор "большой" пиктограммы
mov wc.hIconSm, rax          ; Дескриптор пиктограммы маленького окошка
      invoke LoadCursor,0,IDC_ARROW ; Загрузка курсора
      mov wc.hCursor,rax      ; Курсор
mov wc.cbClsExtra,0     ; Количество доп. байтов для структуры класса
mov wc.cbWndExtra,0   ; Количество доп. байтов для структуры окна
invoke RegisterClassEx,ADDR wc ; Функция регистрации класса окна

                ; Установка размеров и центрирование окна
        mov width_, 370
```

```
        mov height, 550
        invoke GetSystemMetrics,SM_CXSCREEN
        shr rax,2
        mov rbx,rax
        mov rax,width_
        shr rax,2
        sub rbx,rax
        mov xCtr,rbx
        invoke GetSystemMetrics, SM_CYSCREEN
        shr rax,2
        mov rbx,rax
        mov rax,height
        shr rax,2
        sub rbx,rax
        mov yCtr,rbx

invoke CreateWindowEx, WS_EX_TOOLWINDOW, ADDR ClassName,\ ;
ADDR AppName, WS_POPUP,\  ; Адр. имени окна и базовый стиль окна
      xCtr,yCtr,width_,height,0,0,hInst,0 ; Координаты окна и дескрипторы
        mov hwnd,rax ; Сохранение дескриптора окна
mov hBitmap,rv(LoadImage,hInstance,addr BitMap,IMAGE_BITMAP,
0,0,LR_LOADFROMFILE)
      invoke ShowWindow,hwnd,SW_SHOWNORMAL ; Видимость окна
    m0: ; WHILE TRUE. Главный цикл программы
      invoke GetMessage,ADDR msg,0,0,0 ; Чтение сообщения
      or rax,rax ; Формирование признаков
        jz Quit ; Перейти на метку Quit, если rax = 0
invoke DispatchMessage, ADDR msg ; Отправление смс к WndProc
        jmp m0 ; Окончание цикла обрабатывания сообщений
      Quit:
        mov rax, msg.wParam
        ret ; Возвращение из процедуры WinMain
      WinMain endp ; Окончание процедуры WinMain

WndProc proc hWnd:HWND, uMsg:UINT, wParam:WPARAM,
lParam:LPARAM
      LOCAL hdc:HDC       ; Резервирование стека под дескриптор окна
      LOCAL ps:PAINTSTRUCT ; Резервирование стека под структуру
PAINTSTRUCT

      LOCAL bitmap : BITMAP
      LOCAL hdcMem : HDC
      LOCAL oldBitmap : HANDLE  ; HGDIOBJ = HANDLE. Описатель
предыдущего изображения
```

```
        .switch uMsg
        .case WM_DESTROY ; Уничтожение окна
          invoke PostQuitMessage,0 ; Отправка сообщения WM_QUIT

        .case WM_NCRBUTTONDOWN ; Нажата правая кнопка мыши в
неклиентской области
        invoke SendMessage,hWnd,WM_DESTROY,0,0 ; Отправка
сообщения об уничтожении окна
invoke PlaySound,ADDR pzsFiles,0,SND_FILENAME OR SND_ASYNC
          invoke Sleep,1500

        .case WM_NCHITTEST ; IF uMsg == WM_NCHITTEST (Тест нажатия
кнопки мыши в неклиентской области)
          invoke DefWindowProc, hWnd, uMsg, wParam, lParam
          ; Перенаправление нажатия в клиентской области в
неклиентскую
          .if rax == 1 ; 1 = HTCLIENT
          mov rax, 2   ; 2 = HTCAPTION
          ret
          .endif
          ret

        .case WM_PAINT ; IF uMsg == WM_PAINT (Перерисовка области окна)
invoke GetWindowLong,hWnd,GWL_EXSTYLE ; Извлекает данные о
расширенном стиле окна
          or eax,WS_EX_LAYERED ; Добавление стиля WS_EX_LAYERED
invoke SetWindowLong,hWnd,GWL_EXSTYLE,eax ; Установка нового
стиля окна

   invoke SetLayeredWindowAttributes,hWnd,0,0,LWA_COLORKEY ;
Установка прозрачности для окна

   mov hdc,rv(BeginPaint,hWnd,ADDR ps)  ; Вызов процедуры и
заполнение структуры
   mov hdcMem,rv(CreateCompatibleDC,hdc)  ; Создание контекста
устройства в памяти
   mov oldBitmap,rv(SelectObject,hdcMem,hBitmap) ; выбор нового
изображения

      invoke GetObject,hBitmap,sizeof bitmap,addr bitmap
invoke BitBlt,hdc,0,0,bitmap.bmWidth,bitmap.bmHeight,hdcMem,0,0,SRCCOPY ;
Отображение изображения
          invoke SelectObject,hdcMem,oldBitmap
           invoke EndPaint,hWnd,ADDR ps        ; Окончание рисования
```

```
        .default   ; Иначе
invoke DefWindowProc,hWnd,uMsg,wParam,lParam ; Вызов оконной проц.
          ret     ; Возвращение из процедуры
        .endsw     ; End switch
           xor eax, eax
          ret      ; Возвращение из процедуры

        WndProc endp ; Окончание процедуры WndProc
        end          ; Окончание программы
```

Результат выполнения программы представлен на рис. 9.27.

Рис. 9.27. Результат выполнения программы

Функция SetLayeredWindowAttributes устанавливает светопроницаемость и прозрачность окраски клавиши многослойного окна.

Параметры:

Hwnd – дескриптор многослойного окна;

Alpha – Alpha-значение используется для описания непрозрачности многослойного окна;

dwFlags – устанавливает предпринимаемое действие:

LWA_COLORKEY – цвет прозрачности;

LWA_ALPHA – непрозрачность многослойного окна.

9.4. Лабораторная работа «Внедрение кода в секции данных»

Цель занятия: приобрести практические навыки внедрения кода в различные секции;

создание новой функциональности, используя неявное в эксплуатации программное обеспечение для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с комментариями к каждой строке программы;

– результат выполнения программы (скриншоты упрощенного окна (MessageBox) и отладчика);

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Постановка задачи

1. Написать программу в среде masm64 с использованием макросов Msgboxl с неповторяющимися собственными иконками. Программу и результаты привести в отчете.

2. Код программы разместить в 3-х секциях.

Требования к интерфейсу пользователя:

Создать диалоговое окно с тремя диалоговыми окнами:

– собственными иконками с указанием фамилии разработчика программы на диалоговых окнах;

– кнопками;

– текстом на окне;

– статическими рамками и др.

Задание

В соответствии с номером студента в группе выбрать вариант задания и написать в среде masm64 на ассемблере программу вычисления одного из выражений:

1. $4ac - 32c/b$;
2. $8a/7 - d/32b$;
3. $d/4a + 64d/c$;
4. $2b - e/4b$;
5. $16d/4a - 4cd$;
6. $e/8b + 16ac$;
7. $d/4a + 16d/c$;
8. $32ab/2a - 4c$;
9. $2b - e/8b$;
10. $2d/a - 16bc$;
11. $4d/b + 8e$;
12. $8d/b - 32d/c$;
13. $2ab - 16c/d$;
14. $16d/2c - 8ab$;
15. $d/b - d/16c$;
16. $2a/b - c/16e$;
17. $2ab - 16c/b$;
18. $4e/b + 32ac$;
19. $8d/2b - 4ac$;
20. $32b - 8ac/d$;
21. $cb/2a - 8d$;
22. $e/4b - d/8c$;
23. $d/4c - 32ab$;
24. $32d/2c - 8cd$;
25. $32a/8c - d/2e$;
26. $4d/a - 16bd$;

где *a, b, c, d ,e, f* – числа.

Программа 9.11. Выполнение уравнения без интерфейса окна с выводом результата и адреса нахождения его в ячейке памяти:

```
        include \masm64\include64\masm64rt.inc
.data
        ; a/b - cd/e + f/g
        ; 10/2-(3*4)/6+8/2= 5-2+4 = 7
data0 dq 10,2,3,4,6,8,2
              ; a,b,c,d,e ,f, g
        res0 dq ?
        title0 db "Выполнение уравнения", 0
fmt0 db "Уравнение: a/b - cd/e + f/g",10,10,"Результат: %d",10,"Address:
%p",10,10,
               "http://blogs.kpi.kharkov.ua/v2/asm",0
        buf0 db ?

.code
entry_point proc
            ; Передача параметров в count через регистры
            mov r8, qword ptr [data0]          ; a
            mov r9, qword ptr [data0 + 8h]     ; b
            mov r10, qword ptr [data0 + 10h] ; c
            mov r11, qword ptr [data0 + 18h] ; d
            mov r12, qword ptr [data0 + 20h] ; e
            mov r13, qword ptr [data0 + 28h] ; f
            mov r14, qword ptr [data0 + 30h] ; g
        call count              ; процедура выполнения уравнения
            mov res0, rax
invoke wsprintf,addr buf0,addr fmt0,res0,addr res0 ; преобразование
invoke MessageBox,0,addr buf0,addr title0,MB_ICONINFORMATION
```

```
 invoke ExitProcess, 0
      entry_point endp

count proc
                          ; a/b - cd/e + f/g
        mov rax,r8
        xor rdx,rdx
        idiv r9          ; a/b rax
        mov rbx,rax  ; a/b rbx

        mov rax,r10  ; c
        imul r11        ; cd
        xor rdx,rdx
        idiv r12         ; cd/e rax
        sub rbx,rax    ; a/b - cd/e rbx

        mov rax,r13   ; f
        xor rdx,rdx
        idiv r14         ; f/g rax
        add rax,rbx    ; a/b - cd/e + f/g rax
        ret
      count endp
      end
```

Результат выполнения программы приведен на рис. 9.27.

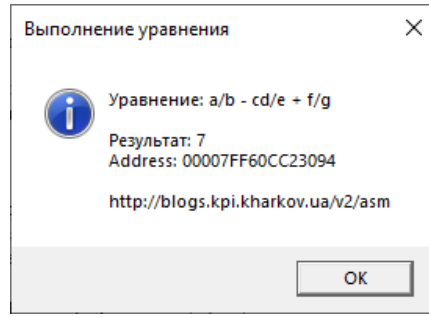


Рис. 9.27. Результат выполнения программы

Все числа, которые используются в уравнении записаны в один массив с именем data0 и выбираются из массива при помощи смещения, равного размерности элементов этого массива.

10. ВНЕДРЕНИЕ КОДА. ИЗМЕНЕНИЕ ТОЧКИ ВХОДА В ПРОГРАММУ

10.1. Алгоритм изменения точки входа в программу

Точка входа (англ. Entry Point (EP) – точка входа) – это адрес в оперативной памяти, с которого начинается выполнение программы.

Необходимость изменения точки входа может возникнуть в случае, если после внедрения кода необходимо сначала запустить внедряемый (вредоносный) код, а уже потом передать управление на продолжение основной программы. Так в основном поступают программы-вирусы, которые выполняют враждебные действия.

Внедряемый код наиболее просто можно разместить как после основной программы в той же секции кода, так и в других секциям, если выставить им соответствующие разрешения на запись и выполнение в этих секциях. Но такие секции теоретически должны отслеживать антивирусы. Но самым интересным способом внедрения кода можно считать внедрение его в заголовок PE-формата.

Общий алгоритм изменения точки входа:

– открыть подопытный exe-файл в отладчике;

– в свободные ячейки программы вставить свой код;

– последней командой вставленного кода вставляем команду передачи управления с адресом начала выполнения основной программы;

– открыть программу-анализатор PE-файла, например программу LordPE и загрузить в нее подопытный exe-файл;

– в программе-анализаторе PE-файла изменяем точку входа на адрес выполнения первой строки вставленного кода. Этот адрес определяется в отладчике как смещение относительно первоначальной точки входа до начала нового адреса. Значение новой точки входа в программе LordPE равняется сумме первоначального значения точки входа и смещения, по которому располагается новый адрес начала программы.

Пример 10.1. Рассмотрим механизм изменения точки входа на примере программы 10.1. После получения exe-файла в отладчике в свободные ячейки памяти в конце секции кода вставим команду передачи управления на начало программы. И, естественно, на нее поставим точку входа.

Программа 10.1. Вызов функции MessageBox с передачей параметров отдельными командами:

```
    include \masm64\include64\masm64rt.inc
.data
    titl db "Просто MessageBox",0
    inf1 db "Передача параметров отдельными командами.",0
.code
entry_point proc
      ;invoke MessageBox,0,addr inf1,addr titl,MB_ICONINFORMATION
    xor rcx,rcx
    lea rdx,inf1
    lea r8,titl
    mov r9,64
    call MessageBox

  invoke ExitProcess,0
    entry_point endp
    end
```

Компилируем эту программу и получаем exe файл, который открываем в отладчике x64Dbg (рис. 10.1).

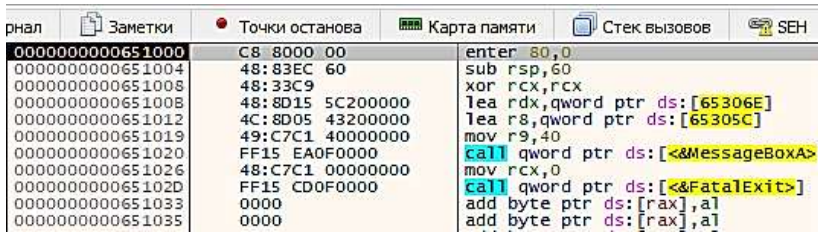


Рис. 10.1. Результат выполнения программы в отладчике

Исходный адрес точки входа программы 0x651000, а свободные ячейки памяти начинаются с адреса 0x651033.

В соответствии с первоначальным замыслом вставляем в свободную ячейку памяти команду передачи управления jmp. Но эта команда используется с адресом перехода.

Внесем адрес в буфер памяти. Для этого на первой строке кода нажимаем правую кнопку мышки и выбираем Копировать/ Адрес.

Ставим курсор и выделяем строку кода со свободной ячейкой. Нажимаем Пробел и в появившемся окно пишем код операции jmp и

вставляем адрес – Ctrl+V (рис. 10.2). Если адрес из буфера вставляется не правильный, то прописываем его самостоятельно.

После произведенных действий окно отладчика с кодом будет иметь вид, представленный на рис. 10.3.

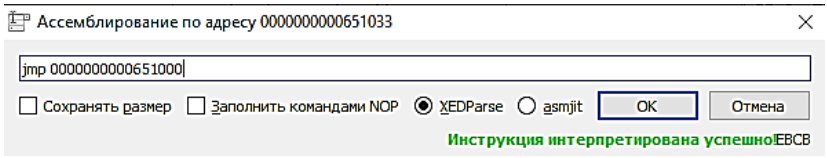


Рис. 10.2. Окно изменения кода ассемблирования

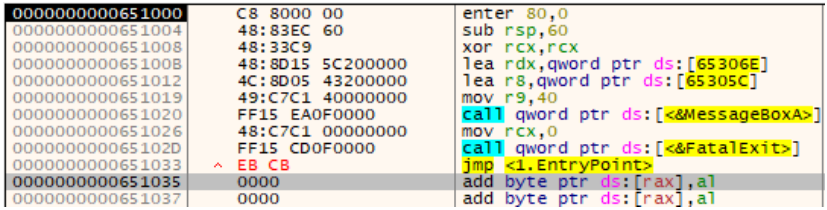


Рис. 10.3. Окно отладчика с изменениями кода программы

Для сохранения изменений в exe-файл выберем Файл/ Исправить файл или еще проще – в дополнительной строке меню выбрать пиктограмму Исправления (изображение резинки). В окне Исправление выбрать Исправить файл. В новом окне ввести новое имя и расширение exe. Подтвердить свои действия.

Приступаем к изменению точки входа.

Открываем программу LordPE (рис. 10.4).

Выбираем кнопку PE Ediror и загружаем измененный exe-файл. В окне с именем EntryPoind и первоначальным адресом 00001000 меняем адрес смещения от точки входа в отладчике до адреса расположения команды jmp. Это смещение равняется 33. Поэтому, к первоначальному адресу прибавляется смещение, по которому располагается новый адрес начала программы – 00001033 (рис. 10.5). Вводим этот адрес, нажимаем кнопку Save и подтверждаем Ok.

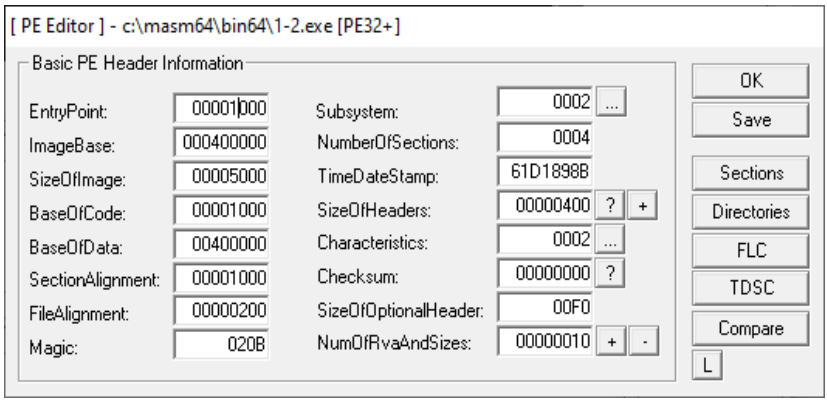


Рис. 10.4. Окно программы LordPE с загруженной программой

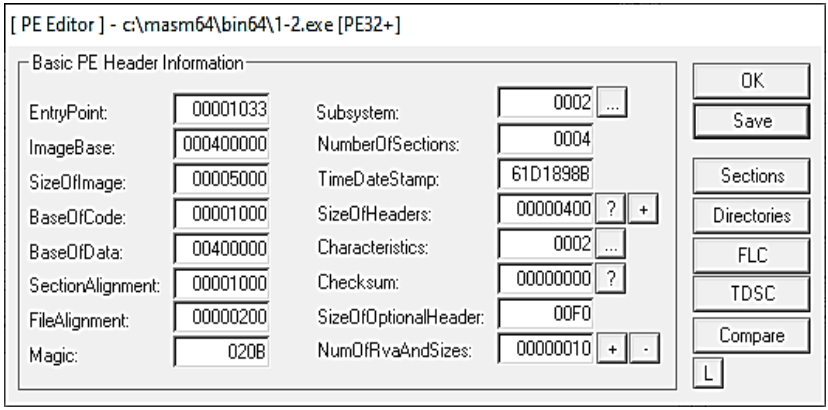


Рис. 10.5. Окно программы LordPE с измененным адресом точки входа

Проверяем последовательность выполнения строк кода нового exe-файла в отладчике x64Dbg (рис. 10.6).

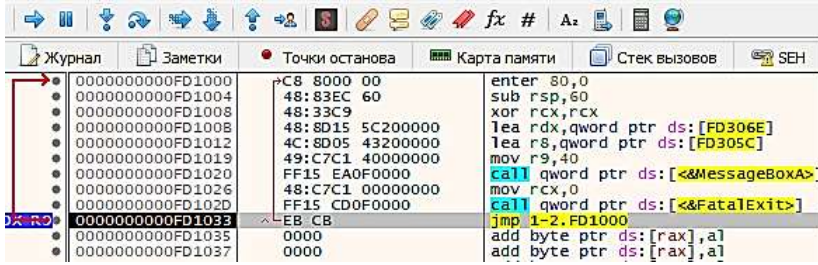


Рис. 10.6. Окно отладчика с измененным адресом точки входа

Т.к. точка входа в программе изменена, то первой выполняемой строкой кода будет строка

```
jmp 1-2.FD1033
```

Эта команда передает управление на начало выполнения основной программы. Правильным началом программы являются первые две строки, которые вставляет среда masm64

```
enter 80,0
sub rsp,60
```

Эти строки предназначены для выравнивания стека и без этих строк новые функции будут выполняться с ошибками.

Поэтому, если предполагается внедрение функций с параметрами, то при смене точки входа удобнее переносить и эти две начальных строки кода. А возврат управления на адрес перехода надо осуществлять уже с учетом этих изменений.

Пример 10.2. Внедрим в программу 10.1 в конец секции кода **новую функцию MessageBox** и поставим на нее точку входа. Т.е. сделаем так, чтобы программа начиналась со внедренной функции.

Алгоритм решения:

– получаем exe файл и открываем его в отладчике;

– пишем новую программу с используемой функцией или, если она уже применяется в подопытной программе, используем строки кода передачи параметров из нее;

– в отладчике переносим (или копируем – в зависимости от организации начала программы) две первых строки кода на новое место, которые используются для выравнивания стека;

– копируем и переносим строки кода передачи параметров в функции со старыми адресами;

– меняем старые адреса в параметрах функции на новые;

– последней командой вставляем команду jmp на адрес продолжения выполнения кода;

– в программе-анализаторе меняем точку входа.

Рассмотрим более детельно этот алгоритм.

Т.к в рассматриваемой программе уже используется требуемая функция, которую будем внедрять в качестве образца, то и писать новую программу для получения exe-файла с этой функцией в этом случае смысла нет.

Открываем exe-файл в отладчике.

Переносим первые две строки кода, которые отвечают за выравнивание стека. Для этого, на строке, которую необходимо перенести, нажимаем Пробел. После появления окна ассемблирования сохраняем строку в буфере, нажимая комбинацию Ctrl+C.

Выбираем в конце кода свободную ячейку, нажимаем Пробел и вставляем строку из буфера, нажимая комбинацию Ctrl+V.

Затем также переносим не только вторую строку, но и строки кода, которые отвечают за передачу параметров и вызов функции MessageBox пока со старыми адресами.

Последней командой вставляем команду возврата на адрес продолжения кода (рис. 10.7).

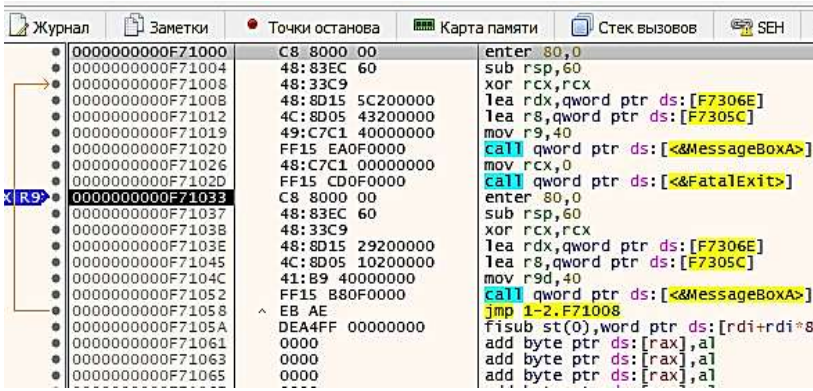


Рис. 10.7. Первоначально добавленные в отладчик строки кода

Адресом возврата является адрес расположения третьей строки кода

```
0000000000F71008 48:33C9 xor rcx,rcx
```

т.к. первые две строки кода так же перенесены.

Сохраняем внесенные промежуточные изменения. Для сохранения изменений в exe-файл выберем Файл/ Исправить файл или еще проще – в дополнительной строке меню выбрать пиктограмму Исправления (изображение резинки). В окне Исправление выбрать Исправить файл. В новом окне ввести новое имя и расширение exe. Подтвердить свои действия.

Но запускать такой exe-файл пока нельзя, т.к. не изменена точка входа.

Вычисляем смещение адреса расположения новой точки входа.

Первоначальный адрес точки входа равнялся 0xF71000. Новый адрес точки входа – 0xF71033. Следовательно, смещение равняется – 33.

Открываем программу LordPE. В ней открываем наш новый exe-файл и к первоначальной точке входа прибавляем значение 33. Вводим этот адрес, нажимаем кнопку Save и подтверждаем Ok.

Вот теперь можно проверить работоспособность программы.

Осталось поменять текст, который выводит новая функция MessageBox. В этой функции в регистре rdx передается сообщение, а название титула упрощенного окна передается в регистр r8.

Выделяем строку

```
0000000000F7103E 48:8D15 29200000    lea rdx,qword ptr ds:[F7306E]
```

новой функции.

Переходим на дамп памяти по этому адресу: нажимаем правую кнопку мышки и выбираем Перейти к дампу / Константа (рис. 10.8).

Адрес	Шестнадцатеричное																ASCII
0000000000F7306E	CF	E5	F0	E5	E4	E0	F7	E0	20	EF	E0	F0	E0	EC	E5	F2	Ïåðåäà÷à ïàðàìåò
0000000000F7307E	F0	EE	E2	20	EE	F2	E4	E5	EB	FC	ED	FB	EC	E8	20	EA	ðîâ îòäåëüíûì ê
0000000000F7308E	EE	EC	E0	ED	E4	E0	EC	E8	2E	00	00	00	00	00	00	00	îìàíäàìè.....
0000000000F7309E	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000000F730AE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
0000000000F730BE	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....

Рис. 10.8. Дамп памяти

Ставим курсор на свободную ячейку, но пропустив хотя бы одну с нулевым значением. Это связано с тем, что текстовая строка функции заканчивается нулем на конце.

Необходимо этот адрес вставить в параметр функции.

Для этого, выделяем эту ячейку (нажимаем на этой ячейке правую клавишу мышки).

Нажимаем правую клавишу мышки и выбираем Копировать / Адрес.

Выделяем строку с названием регистра rdx, нажимаем Пробел и вставляем новый адрес из буфера.

Подводим курсор к выбранной ячейке дампа памяти, нажимаем левую клавишу мышки. Затем нажимаем правую клавишу мышки и выбираем Двоичные операции /Редактировать. Вводим сообщение (рис. 10.9).

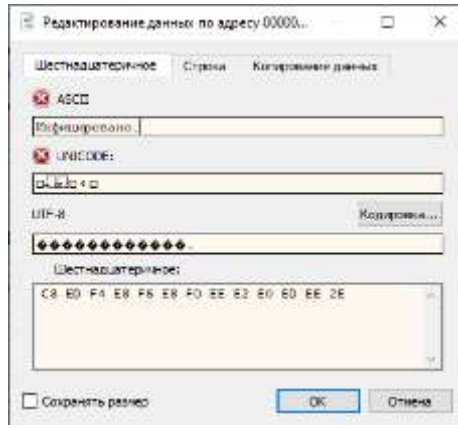


Рис. 10.9. Окно редактирования

Аналогично меняем текст и в титуле окна на текст Окно инфицирования.

Делаем последнее сохранение изменений с заменой имени файла на другое. Для этого выбираем пиктограмму с названием Исправления, а далее новое имя и расширение exe.

Запускаем exe-файл. После запуска последовательно выводятся 2 сообщения (рис. 10.10).

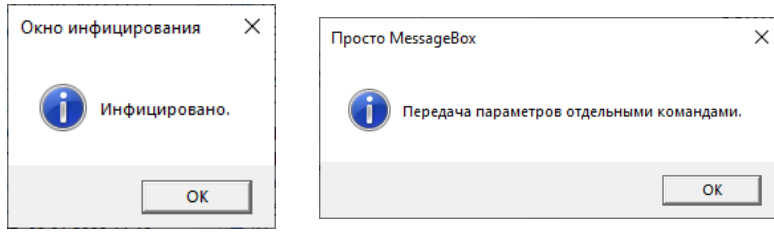


Рис. 10.10. Окна с полученными результатами внедрения

Окно отладчика с последними изменениями представлено на рис. 10.11.

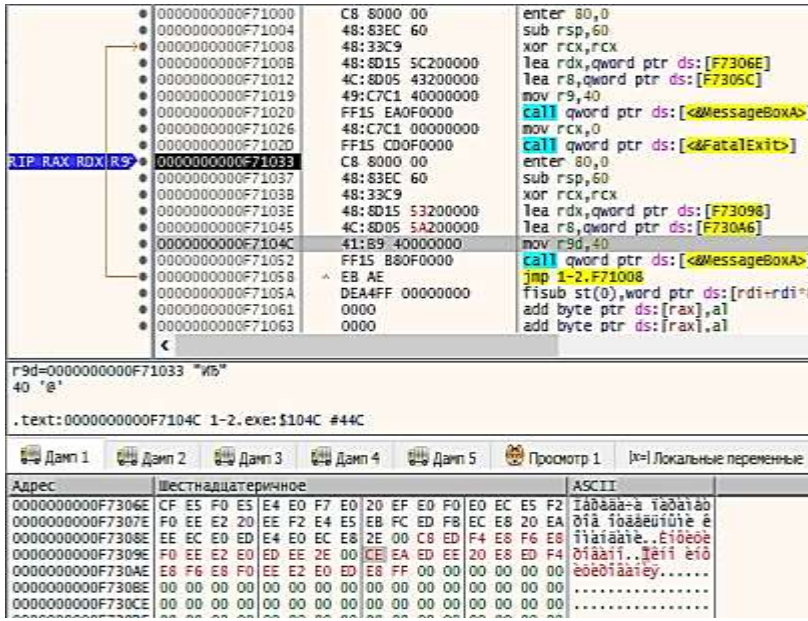


Рис. 10.11. Окна отладчика с последними изменениями

10.2. Лабораторная работа «Изменение точки входа»

Цель занятия: приобрести практические навыки внедрения кода и изменения точки входа в программу;

создание новой функциональности, используя неявное в эксплуатации программное обеспечение для платформы x64 в среде masm64.

В отчете представить:

– номер и название лабораторной работы, фамилию исполнителя;

– номер варианта и условие задания;

– текст исходной программы с указанием в комментариях номера варианта и условия задания;

– результат выполнения программы (скриншоты упрощенного окна (макрос **MsgboxI**) и отладчика);

– особенности выполнения, где описать общий алгоритм исправления вашей программы: какие клавиши в отладчике использовались для получения исправленной программы.

Постановка задачи

Написать программу в среде masm64 с использованием макросов MsgboxI с неповторяющимися собственными иконками в двух вариантах: одну – без интерфейса окна; другую – с диалоговыми окнами с собственными оригинальными иконками, состоящими из не менее чем из двух графических объектов, а также фамилии студента. Использовать статические объекты и элементы управления (кнопки). Программы и результаты привести в отчете.

В exe-файл программы внедрить функцию MessageBox и поставить на нее новую точку входа. Этой функцией вывести фамилию автора программы и номер учебной группы.

Задание

В соответствии с номером студента в группе выбрать вариант задания и написать в среде masm64 на ассемблере программу подсчета количества комбинаций в битовом представлении символов и количество букв в соответствии с заданием.

В exe-файл программы внедрить функцию MessageBox и поставить на нее новую точку входа. Этой функцией вывести фамилию автора программы и номер учебной группы.

Задан текстовый файл со стихотворением «Письмо к женщине», автор С.Есенин (найти в интернете). В кодировке двоичной последовательности посчитать количество:

1. Комбинаций: 0; 00 и букв: а, б.
2. Комбинаций: 00; 000 и букв: о, б, с.
3. Комбинаций: 000; 0000 и букв: е, с, д.
4. Комбинаций: 0000; 1; 01 и букв: и, д, о.
5. Комбинаций: 01; 001 и букв: а, о, е.
6. Комбинаций: 001; 0001 и букв: е, а, и.
7. Комбинаций: 0001; 11 и букв: е, и, н, т.
8. Комбинаций: 11; 111 и букв: о, т, с.
9. Комбинаций: 111; 1111 и букв: е, с, р, в.
10. Комбинаций: 1111; 10 и букв: а, р, в, л.
11. Комбинаций: 10; 101 и букв: и, л, к, м.
12. Комбинаций: 101; 1010 и букв: н, м, д, и.
13. Комбинаций: 1010; 00000000 и букв: о, и, т, д.
14. Комбинаций: 00000000; 20h и букв: о, е, а, и.
15. Комбинаций: 0001; 0010 и букв: м, о, ь, ч.
16. Комбинаций: 0010; 0011 и букв: д, о, г, щ.
17. Комбинаций: 0011; 0100 и букв: п, е, з, ш.
18. Комбинаций: 0100; 0101 и букв: у, а, б, э.
19. Комбинаций: 0101; 0110 и букв: я, и, с, ю.
20. Комбинаций: 0110; 0111 и букв: ы, т, с, я.
21. Комбинаций: 0111; 1000 и букв: к, ы, с, л.
22. Комбинаций: 1000; 1001 и букв: и, ы, т, я.
23. Комбинаций: 1001; 0001 и букв: е, ы, т, с.
24. Комбинаций: 0001; 0101 и букв: а, ы, т, с, я.
25. Комбинаций: 0100; 1000 и букв: о, ы, т, с, я.

Вывести полностью условие задания, количество символов в тексте и результаты.

При выполнении этого задания можно использовать фрагменты кода, рассмотренные при сканировани строк символов для поиска конкретных букв (тема «Строковые команды» в учебной дисциплине «Системное программирование»), а также фрагменты кода поиска битовых представлений, приведенного в программе 12.1.

11. ДИНАМИЧЕСКИЕ БИБЛИОТЕКИ

11.1. Общие сведения о динамических библиотеках

Динамические подключаемые библиотеки (Dynamic Link Libraries, DLLs) являются хранилищем общедоступных процедур. Механизм DLL-библиотек возник вместе с операционной системой Windows и является ее неотъемлемой частью. Суть этого механизма в том, что в процессе компоновки выполняемого модуля с использованием внешних процедур в него размещаются не сами процедуры, а только их названия (номера) вместе с названиями DLL-библиотек, в которых они содержатся.

Windows не загружает несколько копий DLL в память при одновременном выполнении нескольких экземпляров приложения. В реальности, у всех процессов, использующих одну и ту же dll, есть своя копия этой библиотеки, однако Windows делает так, чтобы все процессы разделяли один и тот же код этой dll. Тем не менее, секция данных копируется для каждого процесса.

Загрузка DLL отнимает время и увеличивает расход потребляемой приложением памяти. Поэтому бездумное дробление одного приложения на множество DLL не приводит к улучшению приложения.

Другое дело – если какие-то функции используются несколькими приложениями. Тогда, поместив их в одну DLL, можно избавиться от дублирования кода и увеличения объема приложений.

При разработке DLL предполагалось, что использование DLL позволит эффективно организовать память и дисковое пространство, используя только один экземпляр библиотечного модуля для различных применений.

Полных преимуществ от внедрения DLL получить не удалось в результате явления, называемого DLL hell («от DLL»). DLL hell возникает, когда несколько приложений требуют одновременно не полностью совместимые версии библиотек. Это приводит к сбоям в этих приложениях и к конфликтам типа DLL hell, резко снижая общую надежность операционных систем. В поздних версиях Microsoft Windows стали применять параллельное использование различных версий DLL (технология SxS), что снизило достоинство применения DLL.

Таким образом, DLL-библиотеки используются **для экономии памяти**, поскольку все запущенные приложения могут использовать один модуль DLL-библиотеки, не включая стандартные функции в состав своих модулей.

С помощью DLL-библиотек можно организовать коллективное использование ресурсов или данных, расположенных в сегменте данных библиотеки.

Использование DLL-библиотек повышает модульность приложений и самой операционной системы Windows.

Применение динамических библиотек позволяет добиться ряда преимуществ:

– уменьшается размер исполняемого файла приложения и занимаемые им ресурсы;

– функции DLL могут использовать несколько процессов одновременно;

– управление динамическими библиотеками возлагается на операционную систему;

– внесение изменений в DLL не требует перекомпиляции всего проекта;

– одни DLL могут использовать программы, написанные на разных языках.

11.2. Типы динамических библиотек

Есть 2 типа библиотек:

– статические – расширение lib;

– динамические – расширение dll.

Статическая библиотека состоит из подпрограмм, которые непосредственно компилируются и линкуются с программой. В Windows статические библиотеки имеют расширение .lib (сокр. от «library»)

Динамическая библиотека (или «общая библиотека») состоит из подпрограмм, которые подгружаются в программу во время её выполнения. При компиляции эта библиотека не становится частью вашего исполняемого файла. В Windows динамические библиотеки имеют расширение .dll (сокр. от «dynamic link library»). Одним из преимуществ динамических библиотек является то, что разные программы могут совместно использовать одну копию динамической библиотеки, что значительно экономит используемое пространство.

DLL – это библиотека функций, которые совместно используются другими исполняемыми программами. Lib – это библиотека функций, которые статически связаны с программой.

11.3. Создание динамического файла

С практической точки зрения имена файлов проще составлять вместе с расширением и разделительной точкой не больше 8 символов. Особенно это касается случая внедрения динамической библиотеки.

Для создания своей динамической библиотеки с файлом по имени myio.dll без конкретной точки входа необходимо выполнить определенные шаги.

Шаг 1. В текстовом редакторе создадим основную программу с именем myio.asm с меткой начала программы (программа 11.1). В этой программе необходимо подключить динамическую библиотеку со своим именем (mydll.lib), которую предстоит создать.

Программа 11.1. Файл с именем myio.asm:

```
; вызов собственной DLL
include /masm64/include64/masm64rt.inc
includelib mydll.lib
AddDD proto :QWORD,:QWORD ; прототип процедуры
.data
title1 db " Решение уравнения. masm64",0
txt1 db "Уравнение  a+b ",10, "Результат: %d",10,
"Автор: Рысованый А.Н., http://blogs.kpi.kharkov.ua/v2/asm, НТУ ХПИ",0
buf1 dq 0,0
.code
entry_point proc
invoke AddDD,5,4 ; вызов процедуры AddDD с параметрами
invoke wsprintf,ADDR buf1,ADDR txt1,rax; функция преобразования
invoke MessageBox,0,ADDR buf1,ADDR title1,MB_ICONINFORMATION
invoke ExitProcess,0
entry_point endp
end
```

Шаг 2. Создадим программу с именем mydll.asm без метки начала программы и без функции ExitProcess. Этот файл – основа создания dll-файла.

Программа 11.2. Файл с именем mydll.asm:

```
; основа для создания DLL
include /masm64/include64/masm64rt.inc
```

```
      AddDD proto :QWORD,:QWORD ; прототип процедуры
.code
      AddDD proc arg1:QWORD,arg2:QWORD
        mov rax,arg1 ;
        add rax,arg2 ;
        ret        ;
      AddDD endp   ;
      end ;
```

Шаг 3. Создадим файл с именем **mydll.def**, в котором указываются имя программы, которое включается в библиотеку и имя (имена) вызываемой процедуры, которое передается (экспортируется) в другую программу.

Программа 11.3. Файл с именем mydll.def:

```
LIBRARY  myio ; имя программы, которая включается в библиотеку
EXPORTS  AddDD    ; имя процедуры, которое передается
```

В соответствии с содержимым этого def-файла и будет создан файл с этим же именем, но с расширением dll.

Шаг 4. Создадим командный **bat-файл** для получения dll-файла и откомпилируем им файл **mydll.asm**

Программа 11.4. Файл с именем makeit.bat:

```
      :: командный bat-файл для получения dll-файла
      :: вывод выполняющихся строк на экран
@echo off
      :: установка переменной
set appname=%1
\masm64\bin64\ml64.exe /nologo /Cp /c %appname%.asm
      :: nologo - подавление авторских прав при запуске компилятора и
отображения сообщений при компиляции

\masm64\bin64\link.exe  /SUBSYSTEM:WINDOWS /ENTRY:entry_point
/DLL /DEF:mydll.def /NOENTRY  %appname%.obj

      ::NOENTRY требуется для создания ресурсной библиотеки DLL
      :: вывод списка файлов
dir %appname%.*
      pause
```

Выполним компиляцию. Результат компиляции с именами всех полученных файлов представлен на рис. 11.1.

Рис. 11.1. Результат компиляции с именами всех файлов

Шаг 5. Подготовим bat-файл с именем **makeit.bat** для создания исполняющего exe-файла. Это – обычный bat-файл для создания exe-файла, но без использования ресурсов, т.к. в этой программе их нет (программа 11.5).

Программа 11.5. Следующий файл с именем makeit.bat:

```
    :: вывод выполняющихся строк на экран
@echo off
    :: установка переменной
set appname=%1

\masm64\bin64\ml64.exe /c %appname%.asm

\masm64\bin64\link.exe /SUBSYSTEM:WINDOWS /ENTRY:entry_point
/nologo %appname%.obj
    :: nologo - подавление авторских прав при запуске компилятора и
отображения сообщений при компиляции
    :: вывод списка файлов
dir %appname%.*
    :: удаление файла *.obj
del %appname%.obj
    pause
```

Шаг 6. Откомпилируем основной файл **myio.asm.**

После выполнения полученного exe-файла программа выводит сообщение (рис. 11.2).

Код программы с вызовом динамической библиотеки с именем AddDD в отладчике приведен на рис. 11.3.

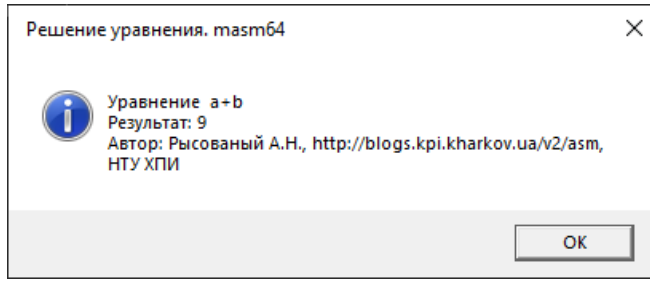


Рис. 11.2. Результат выполнения программы

```
00007FF63F8F1000    C8 8000 00          enter 80,0
00007FF63F8F1004    48:83EC 60          sub rsp,60
00007FF63F8F1008    48:C7C1 05000000    mov rcx,5
00007FF63F8F100F    48:C7C2 04000000    mov rdx,4
00007FF63F8F1016    E8 47000000         call <JMP.&AddDD>
00007FF63F8F101B    48:8D0D B6200000    lea rcx,qword ptr ds:[7FF63F8F30D8]
00007FF63F8F1022    48:8D15 4E200000    lea rdx,qword ptr ds:[7FF63F8F3077]
00007FF63F8F1029    4C:8BC0             mov r8,rax
00007FF63F8F102C    FF15 EE0F0000       call qword ptr ds:[<&wsprintfA>]
00007FF63F8F1032    48:C7C1 00000000    mov rcx,0
00007FF63F8F1039    48:8D15 98200000    lea rdx,qword ptr ds:[7FF63F8F30D8]
00007FF63F8F1040    4C:8D05 15200000    lea r8,qword ptr ds:[7FF63F8F305C]
00007FF63F8F1047    49:C7C1 40000000    mov r9,40
00007FF63F8F104E    FF15 D40F0000       call qword ptr ds:[<&MessageBoxA>]
00007FF63F8F1054    48:C7C1 00000000    mov rcx,0
00007FF63F8F105B    FF15 9F0F0000       call qword ptr ds:[<&FatalExit>]
00007FF63F8F1061    CC                  int3
00007FF63F8F1062    FF25 A80F0000       jmp qword ptr ds:[<&AddDD>]
00007FF63F8F1068    0000                add byte ptr ds:[rax],al
00007FF63F8F106A    0000                add byte ptr ds:[rax],al
00007FF63F8F106C    0000                add byte ptr ds:[rax],al
```

Рис. 11.3. Код программы в отладчике

А код динамического файла с именем AddDD приведен на рис. 11.4.

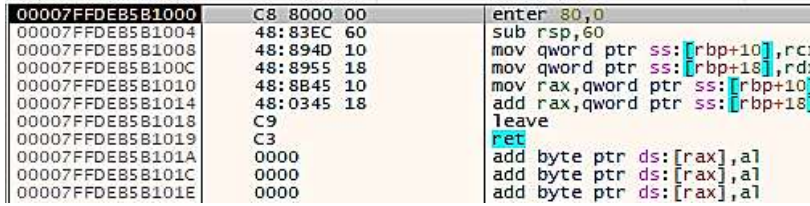


Рис. 11.4. Код динамического файла с именем AddDD

11.4. Точка входа DLL

В Windows все библиотеки DLL могут содержать необязательную функцию точки входа, обычно называемую DllMain, которая вызывается как для инициализации, так и для завершения. Это дает возможность выделять или освобождать дополнительные ресурсы по мере необходимости.

Когда библиотека DLL загружается в адресное пространство процесса либо при загрузке ее использующего приложения, либо при запросе приложением библиотеки DLL во время выполнения, операционная система создает отдельную копию данных библиотеки DLL. Это называется присоединением процесса. Присоединение потока происходит, когда процесс, в который загружена библиотека DLL, создает новый поток.

Операционная система выполняет отдельный вызов точки входа библиотеки DLL для каждого из этих событий, передавая аргумент reason для каждого типа события. Например, операционная система отправляет в DLL_PROCESS_ATTACH качестве аргумента DLL_PROCESS_ATTACH в сигнальное подключение процесса.

Функция Dllmain – дополнительная точка входа в динамично подключаемую библиотеку (http://www.i2r.ru/static/565/ out_17611.shtml).

DllMain может вызываться в четырёх случаях:

- при проецировании библиотеки в адресное пространство **процесса** (DLL_PROCESS_ATTACH) – инициализировать один раз для каждого нового процесса;
- при первом вызове библиотеки из **потока** (DLL_THREAD_ATTACH), например, с помощью функции LoadLibrary – инициализация в зависимости от потока;

- при выгрузке библиотеки **потоком** (DLL_THREAD_DETACH) – очистка для конкретного потока;
- при выгрузке библиотеки из адресного пространства **процесса** (DLL_PROCESS_DETACH) – очистка процесса.

Если функция используется, то она вызывается системой тогда, когда процессы и потоки инициализированы и завершили работу или при вызове функции LoadLibrary и FreeLibrary. Точка входа чаще всего имеет имя DllMain и синтаксис:

```
  invoke DllMain,\ ;
    HINSTANCE hinstDLL,\          ; дескриптор модуля DLL
     DWORD fdwReason,\            ; причина вызова функции
      LPVOID lpvReserved          ; зарезервировано
```

Базовый стиль параметра hinstDLL – это HINSTANCE.

Параметр fdwReason указывает на причину вызова DLL. Когда fdwReason равен DLL_PROCESS_ATTACH, то DLL загружается в виртуальное адресное пространство текущего процесса в результате операции запуска или в результате вызова LoadLibrary.

При значении DLL_THREAD_ATTACH текущий процесс создает новый поток. При значении DLL_THREAD_DETACH поток высвобождает память. А при DLL_PROCESS_DETACH DLL выгружается из виртуального адресного пространства занятого процесса в результате неудачной загрузки DLL, завершения работы процесса или вызова функции FreeLibrary.

Программа 11.6. Файл-основа с именем Dll64-3-dll.asm для создания dll **с точкой входа**:

```
      ; создание DLL
      include /masm64/include64/masm64rt.inc
      AddDD proto :QWORD,:QWORD ; прототип процедуры
.data
      hInst dq 0
.code
      ;   точка входа:
DllMain proc instance:HINSTANCE, reason:QWORD, unused:QWORD
        mov rax, reason          ; проверка причины вызова
        cmp rax, DLL_PROCESS_ATTACH
      ; Если причина отличается от подсоединяемого процесса - выход
        jnz _exit
        push instance  ;
        pop hInst      ; обязательное сохранение хендла программы
      _exit:
```

```
    and rax,1
    ret
DllEntry endp   ; окончание процедуры точки входа

  AddDD proc arg1:QWORD,arg2:QWORD
fn MessageBox,0,"Запущена динамическая библиотека","Dll",
MB_ICONINFORMATION
    mov rax,arg1 ;
    add rax,arg2 ;
    ret        ;
    AddDD endp   ;

    end ;
```

Рассмотрим программу: имеется два массива по 8 чисел. Необходимо создать 3-й массив по правилу: если элемент 1-го массива больше второго, то из элемента 1-го массива вычитанием элемент 2-го массива, а если элемент 1-го массива меньше второго, то выполнить сложение элементов массива с одинаковыми индексами. Создать динамическую библиотеку с выводом сообщения с использованием функции MessageBox. А также из файла динамической библиотеки запустить калькулятор Windows.

Программа 11.7, которая использует динамическую библиотеку имеет имя Dll64-3.asm.

Программа 11.7. Файл с именем Dll64-3.asm:

```
title програма, в которой используется собственная DLL
include \masm64\include64\masm64rt.inc
      comment * ----------------------------
       Два массива. Сформировать  3-й массив:
       если элемент 1-го массива больше второго, то - вычитание
       если элемент 1-го массива меньше второго, то - сложение
       -------------------------------------*
includelib Dll64-3-dll.lib
msgAuthor proto

.data
    _A dq 8,7,6,5,4,3,2,1 ; Массив с исходными данными
    _B dq 1,2,3,4,5,6,7,8 ; Массив с исходными данными

    buf1 dq 8 dup(0),0
    res2 dq 8 dup(0),0
    _title_rez db "Результат вычисления массива:",0 ;
    frmt db "Ответ =", 8 dup("   %d "),10,10,
```

```
        "Автор:",9,"Рысованый А.Н., НТУ ХПИ",10,
        9,"Сайт:  http://blogs.kpi.kharkov.ua/v2/asm/",0

.code                   ; директива начала сегмента кода
entry_point proc

        invoke msgAuthor      ; Вывод сведений об авторе

           lea rsi,_A
           lea rdi,_B
           lea rdx,res2
           mov rcx,8    ; кол. элементов массива - 8
m2:
           mov rax,QWORD ptr[rsi]
           mov rbx,QWORD ptr[rdi]
           .if rax{= rbx
           add rax,rbx
           .else
           sub rax,rbx
           .endif
           mov QWORD ptr[rdx],rax ; результат сохранения в памяти
        ; переход до следующего шага итерации
           add rsi,8
           add rdi,8
           add rdx,8
           loop m2
               ; перезапись в регистры для преобразования в wsprintf
        mov rax,res2[0]
        mov rbx,res2[8]
        mov rsi,res2[16]
        mov rdi,res2[24]
        mov r10,res2[32]
        mov r11,res2[40]
        mov r12,res2[48]
        mov r13,res2[56]
invoke wsprintf,ADDR buf1,ADDR frmt,rax,rbx,rsi,rdi,r10,r11,r12,r13;
invoke MessageBox,0,addr buf1,addr _title_rez,MB_ICONINFORMATION
         invoke ExitProcess, 0     ; выход из программы
        entry_point endp
        end
```

Файл динамической библиотеки приведен в программе 11.8.

Программа 11.8. Файл динамической библиотеки с именем Dll64-3-dll.asm:

```
        include \masm64\include64\masm64rt.inc
            comment * ----------------------------
            Применяется точка входа DllEntry.
            Осуществляется вывод смс с помощью MessageBox
            -----------------------------------------*
        msgAuthor proto ; функция вывода инф. про автора
.data
        hInst dq 0
        strInf db "Вывод смс из Dll",0
        titleInf db "Dll",0
        RunCalc db "c:\Windows\System32\calc.exe",0
.code
        ;   точка входа:
        DllEntry proc instance:HINSTANCE, reason:QWORD,
unused:QWORD
          mov rax, reason           ; проверка причины вызова
          cmp rax, DLL_PROCESS_ATTACH
        ; Если причина отличается от подсоединяемого процесса - выход
          jnz _exit
          push instance  ;
          pop hInst      ; обязательное сохранение хендла программы
        _exit:
            and rax,1
            ret
        DllEntry endp   ; окончание процедуры точки входа

msgAuthor proc
  ;fn MessageBox,0,"Запущена динамическая библиотека","Dll",
        MB_ICONINFORMATION
fn MessageBox,0,addr strInf,addr titleInf,MB_ICONINFORMATION

invoke WinExec,ADDR RunCalc,SW_HIDE; запуск внешнего файла
          ret
msgAuthor endp
        end
```

Файл Dll64-3-dll.def, в котором указываются имя программы, включаемое в Dll-библиотеку и имя экспортируемой функций представлен в программе 11.9.

Программа 11.9. Файл с именем Dll64-3-dll.def:

```
LIBRARY Dll64-3 ; имя программы, которая включается в библиотеку
EXPORTS msgAuthor ; имя процедуры, которое передается
```

При последующей компиляции asm-файла, имя которое указано в строке с ключевым словом LIBRARY, будет создан файл с этим же именем, но с расширением dll. Но для этого сначала надо подготовить файл **makeit.bat.**

В файле с именем **makeit.bat** строка кода для линкования должна иметь вид:

```
\masm64\bin64\link.exe /SUBSYSTEM:WINDOWS /ENTRY:entry_point
/DLL /DEF:Dll64-3-dll.def /NOENTRY %appname%.obj
```

Открываем в редакторе masm64 файл с именем Dll64-3-dll.asm. Компилируем эту программу и получаем файлы, которые представлены на рис. 11.5.

На этом этапе получается и файл с именем **Dll64-3.dll,** имя которого указано в файле с расширением def.

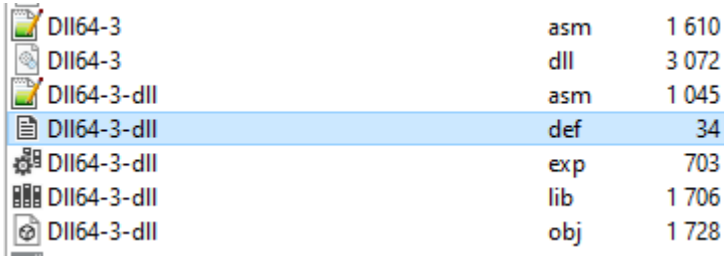


Рис. 11.5. Список фапйлов после компиляции программы с именем Dll64-3-dll.asm

Теперь проверяем использование полученной Dll в программе с именем Dll64-3.asm. А в этой программе есть строка кода

includelib Dll64-3-dll.lib

lib -- это принятый формат статической библиотеки, то есть, библиотеки, которая будет вкомпилирована компоновщиком в исполняемый модуль.

Теперь закружаем в редактор среды masm64 программу с именем Dll64-3.asm.

Перед компиляцией этого файла снова корректируем файл **makeit.bat.**

После компиляции программы получим два сообщения программы (рис. 11.6 и рис. 11.7), а также запуск калькулятора.

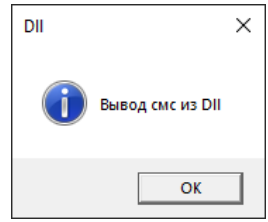


Рис. 11.6. Первое сообщение – сообщение от динамической библиотеки

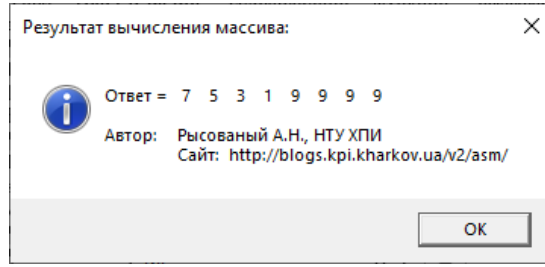


Рис. 11.7. Второе сообщение – сообщение от основной программы

Если в дальнейшем планируется внедрение своего динамического файла в exe-файл, то желательно из практической точки зрения создавать короткие имена файлов.

11.5. Создание нового импорта в DLL

Просто расширить таблицу импорта для собственного DLL нельзя – в PE-заголовке нет свободного места. Однако есть возможность создать новый раздел, перестроить таблицу импорта и записать эту таблицу в этот новый раздел. А на месте старой таблицы необходимо изменить смещение таблицы импорта, чтобы оно указывало на новую таблицу импорта. В главе 4 рассматривался этот процесс, который был реализован при помощи программы **Hiew 8.43.**

Но перестроить таблицу импорта можно и при помощи программы CFF Explorer. Для этого открываем в этой программе exe-файл, который будет вызывать DLL и переходим на вкладку Import Adder.

Нажимаем на кнопку Add и выбираем внедряемую DLL. Внизу появится список функций, которые экспортируются.

Выбираем имя внедряемой DLL, ставим отметку в поле Создать новый раздел / Create New Section» и нажимаем Import By Name (рис. 11.8).

Дальнейшие действия аналогичны рассмотренным в главе 4.

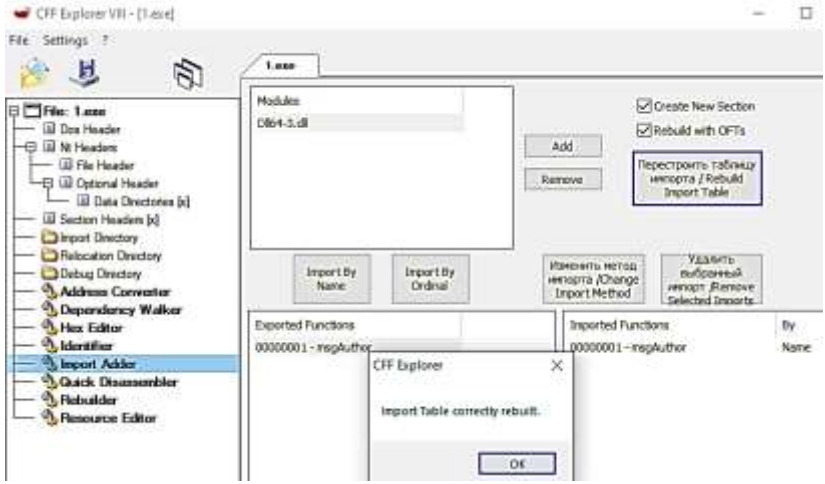


Рис. 11.8. Создание новой секции с таблицей импорта

Список секций можно увидеть в разных программах. Например, в отладчике список секций представлен на рис. 11.9.

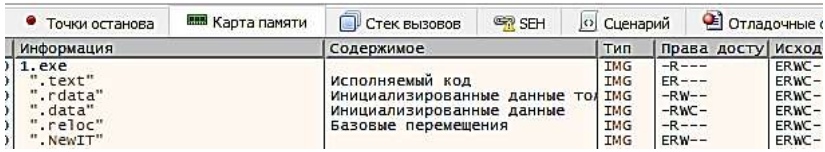


Рис. 11.9. Список секции в отладчике

12. УПАКОВЩИКИ И ПРОТЕКТОРЫ

12.1. Общие сведения

В настоящее время разработано достаточно много упаковщиков и протекторов под архитектуру системы команд x32. Для современной архитектуры x64 пока разработано не так много таких программ. Полностью отсутствуют такие программы с небольшим объемом и простотой использования.

Большое количество программистов пытаются приобщиться к написанию таких программ, т.к. написание упаковщиков и протекторов можно считать серьезной проверкой уровня профессионализма.

Программы, которые используются для сжатия, защиты кода путем усложнения, запутывания и кодирования кода называются **обфускаторами**. Обфускация делает код не читабельным для программиста, но при этом исполняемым.

Существуют следующие типы программ, предназначенные для изменения вида кода:

– архиваторы (Archivers);
– упаковщики (Packers);
– протекторы (Protectors, Crypters).

Архиватор – программа, предназначенная для сжатия без потерь одного и более файлов в единый файл-архив или в серию архивов для удобства переноса и/или хранения данных. Распаковка архивов выполняется с помощью того же архиватора либо посредством сторонних совместимых утилит. Представители – ZIP, RAR, CAB и другие.

Упаковщик – это программа, предназначенная для упаковки (изменения) файла ***со встраиванием упаковщика и распаковщика*** в изменяемый файл. Упаковщики работают так же, как и архиваторы, за одним исключением – они помещают процедуру распаковки в начало программы, которую только что сжали. Распаковка и исполнение файла происходит минуя операции с жестким диском, как это происходит с использованием архиватора. Наиболее популярные сегодня упаковщики – UPX, ASPack, FSG, PeShield, VMProtect, **но не все они успешно обрабатывают 64-разрядный код**. Например, упаковщик upx-3.96-

win64 (2020 год) не всегда успешно обрабатывает даже “чистый” код, тем более с API-функциями в masm64.

Протектор – это программа, которая не только сжимает файл, но и дополнительно шифрует. Загрузчик протектора оснащен большим количеством защит против взлома и анализа кода исходной программы. Поэтому, почти всегда, упакованный протектором файл больше по объему.

Антивирусы, как правило, реагируют на защиту, а в некоторых случаях, такие программы могут не запускаться.

12.2. Сжатие данных без потерь

Сжатие данных без потерь (англ. lossless data compression) – класс алгоритмов сжатия данных (видео, аудио, графики, документов, представленных в цифровом виде), при использовании которых закодированные данные могут быть полностью восстановлены.

В общих чертах смысл сжатия без потерь таков: в исходных данных находят какую-либо закономерность и с учётом этой закономерности генерируют вторую последовательность, которая полностью описывает исходную.

Большинство алгоритмов сжатия без потерь работают **в две стадии:** ***на первой*** генерируется статистическая модель для входных данных, ***на второй*** производится замена битовых представлений на новые, используя модель для получения «вероятностных» (то есть часто встречаемых) данных, которые используются чаще, чем «невероятностные».

Например, для кодирования двоичных последовательностей, в которых **много нулей** и мало единиц, можно использовать такую замену:

00 → 0
01 → 10
10 → 110
11 → 111

В таком случае шестнадцать битов
00 01 00 00 11 10 00 00
будут преобразованы в тринадцать битов
0 10 0 0 111 110 0 0

Приведенный алгоритм преобразования битового представления (который относится ко второй стадии сжатия) реализован в программе 12.1. При вводе произвольного текста в программе не

происходит сжатия объема. Напротив – объем кода после сжатия увеличивается, т.к. количество комбинаций 00 намного меньше, чем других комбинаций. Однако такая программа осуществляет шифрование сообщений методом замены битовых представлений.

Программа 12.1. Шифрование сообщений методом замены битовых представлений:

```
        include /masm64/include64/masm64rt.inc
.const
        ; Filenames
        input_file   db  "codingIn.dat", 0
        output_file  db  "codingDe.dat", 0
.data?
        hInput dq ?
        hOutput dq ?
        read_count dq ? ; Количество байтов, прочитанных файлом ReadFile

.data
        bRead db 0              ; Байт, который был прочитан
        bToWrite db 0           ; Байт для записи
        bRemainder db 0         ; Оставшиеся биты для записи
        nRemainderSize db 0 ; Количество битов в остатке
        nEncodedSize dq 0       ; Количество бит в закодированных данных

.code
entry_point proc
          ; Открытие входного файла
invoke CreateFileA, addr input_file, GENERIC_READ, FILE_SHARE_READ,0,\
                OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL
          cmp rax, INVALID_HANDLE_VALUE
          je exit                 ; If can't open file - exit
          mov hInput, rax         ; Save input file handle

          ; Создание выходного файла
invoke CreateFileA, addr output_file, GENERIC_WRITE, FILE_SHARE_READ,0,\
                CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL
          cmp rax, INVALID_HANDLE_VALUE
          je exit ; If can't open file - exit
          mov hOutput, rax ; Save input file handle

        ; AX - encoded result
        ; CL - shifting
        ; BX - loops
        ; DX - Read data (During data processing)
```

```
        ; Data segments:
        ; - Bits count in the last byte
        ; - Encoded data

; Резервный байт для сохранения количества оставшихся битов
            mov bToWrite, 0
            invoke WriteFile, hOutput, addr bToWrite, 1,0,0

        ; Основной цикл кодирования - Main encoding loop
        m0:
; Read byte from the file
            invoke ReadFile, hInput, addr bRead, 1, addr read_count, NULL
            mov al, bRemainder ; Load reminder into memory
            mov bl, nRemainderSize ; Set start bits count
            mov nEncodedSize, rbx

; Processing of the last byte
            cmp read_count, 1
            je m1 ; Byte was read. Handling

; Else - Prepare the remainder, write and exit
            mov cl, 8
            sub cl, nRemainderSize ; 8 - Rem. size = Count of bits to shift
            shl al, cl ; Move to the left
            mov bToWrite, al
        ; Write and exit
            invoke WriteFile, hOutput, addr bToWrite, 1, NULL, NULL
        ; Save remainder's bits count
            invoke SetFilePointer, hOutput, 0, NULL, FILE_BEGIN
        ; Remainder size is 0 or the size itself (from the last iteration)
            invoke WriteFile, hOutput, addr nRemainderSize, 1, NULL, NULL
            jmp exit

        m1:
        ; Обработка данных - Processing data
        ; Prepare metadata
            mov cl, 6 ; shift size (>> 6, >> 4, >> 2, >> 0)
            mov bl, 4 ; Repeat 4 times (For each pair of bits)

        ; Цикл кодирования - Encoding loop
        m2:
            cmp bl, 0
            je m3 ; End of encoding

            mov dl, bRead ; Данные, которые необходимо обработать
```

```
            shr dl, cl ; Перемещение битов в начало, чтобы их замаскировать
            and dl, 3 ; Маскированные биты для обработки

; Определение битовой последовательности и запись кодированных битов
        ; 00 -> 0
            cmp dl, 0
            jne seq1
            shl rax, 1 ; add 0 to the encoded sequence
            add nEncodedSize, 1 ; Increase size by encoded bits count
            jmp seq_end
        ; 01 -> 10
        seq1:
            cmp dl, 1
            jne seq2
            shl rax, 2 ; add 10 to the encoded sequence
            or rax, 2
            add nEncodedSize, 2 ; Increase size by encoded bits count
            jmp seq_end
        ; 10 -> 110
        seq2:
            cmp dl, 2
            jne seq3
            shl rax, 3 ; add 110 to the encoded sequence
            or rax, 6
            add nEncodedSize, 3 ; Increase size by encoded bits count
            jmp seq_end
        ; 11 -> 111
        seq3:
        ; Don't need to compare. This is the last possible case
            shl rax, 3 ; add 111 to the encoded sequence
            or rax, 7
            add nEncodedSize, 3 ; Increase size by encoded bits count

; end of the condifion and loop
        seq_end:
            dec bl    ; --repeat_count
            sub cl, 2 ; shift size -= 2 (Move to the next pair)
            jmp m2

; End of encoding
        m3:
            xor rbx, rbx ; Writing loops count (0)
            mov rcx, nEncodedSize

; Can't write if size < 8. Moving to the remainder and repeating loop
```

```
            cmp rcx, 8
            jge @f
            mov bRemainder, al
            mov nRemainderSize, cl
            jmp m0

; Get write loops count and remainder size
        @@:
            cmp rcx, 8
            jl @f
        ; Size >= 8
            inc rbx    ; ++write_loops
            sub rcx, 8 ; -1 writable byte from size
            jmp @b

        ; Size < 8
        @@:
            mov nRemainderSize, cl ; Save remainder size for the next reading loop
        ; Preparing mask for the remainder
            xor dl, dl ; 0000 0000
            not dl     ; 1111 1111
            shl dl, cl ; 1111 1000
            not dl     ; 0000 0111

; Save the remainder for the next reading loop
            and dl, al
            mov bRemainder, dl

            shr rax, cl ; Remove remainder from the encoded data
mov r12, rax ; Swap rax, r12. Now r12 contains encoded data (Because of invoke
will rewrite rcx-r11)

        ; Write each byte in loop
            ; rax - encoded data for processing
            ; rbx - loops count
            ; rcl - shifts count
            ; r11 - encoded data
        @@:
            cmp rbx, 0
            je m0 ; End. Perfecto! Next byte, please!

            mov rax, r12
            dec rbx ; -1 loop / shifts count (in bytes)
            mov cl, bl
            imul rcx, 8 ; shifts count (in bits)
```

```
    shr rax, cl ; Move the byte to the left
    and rax, 255 ; Mask exacly 1 byte

    mov bToWrite, al ; Final step
    invoke WriteFile, hOutput, addr bToWrite, 1, NULL, NULL
    jmp @b
exit:
    invoke CloseHandle, hInput
    invoke CloseHandle, hOutput
    invoke ExitProcess, 0
entry_point endp
end
```

12.3. Упаковщики

Упаковка исполняемых файлов заключается в сжатии исполняемого файла и прикреплении к нему кода, необходимого для распаковки и дальнейшего выполнения.

Упаковка применяется по ряду причин:

– упакованный файл занимает меньше места, что ускоряет его загрузку в память;

– некоторые виды упаковки используют шифрование для предотвращения отладки программы;

– упаковка с шифрованием может использоваться при создании вирусов, чтобы зашифровать и видоизменить код вируса для затруднения его обнаружения системами, основанными на сигнатурах (например, антивирусами).

После запуска сжатого файла распаковщик извлекает код программы из архива (обычно в память, хотя существуют и упаковщики, записывающие файл на диск) и передаёт ему управление.

Упакованный исполняемый файл – это разновидность самораспаковывающегося архива (SFX), в котором сжатая информация и программа для распаковки находятся в одном файле.

Существует большое количество упаковщиков, особенно под архитектуру x32. Наиболее известными представителями упаковщиков для x32 являются:

– UPX;

– ASPack;

– eXPressor;

– Mpress;

– PECompact;
– PELock;
– PESpin.

При исследовании упакованного файла его сначала надо распаковать, то есть снять все навесные защиты, восстановить оригинальную точку входа OEP и, как минимум, таблицу импорта.

Например, универсальный принцип действия распаковки, используемый в Quick Unpack, следующий:

1. Поиск оригинальной точки входа (OEP);
2. Запуск программы и остановка на точке входа (EP);
3. Установка контрольной точки (брейкпоинта) на OEP;
4. Запуск программы на исполнение;
5. Срабатывание брейкпоинта, происходит остановка процесса на OEP;
6. Снятие дампа;
7. Восстановление таблицы импорта;
8. Восстановление таблицы настроек адресов (FixUp Table или Relocations) (принципиально только для DLL);
9. Восстановление локальной области данных цепочек (TLS), если она повреждена;
10. Корректировка ресурсов (перемещение, склеивание);
11. Структурирование и модификация дампа с учетом восстановленных данных.

В настоящее время достойных внимания упаковщиков для платформы x64 не очень много. Поэтому в качестве примеров рассмотрим несколько упаковщиков для платформы x32.

12.3.1. Упаковщик UPX

UPX (the Ultimate Packer for eXecutables) – упаковщик/распаковщик исполняемых **32-разрядных файлов**, поддерживающий несколько различных платформ и форматов файлов. Некоторые программы неспособны работать в сжатом виде, так как используют малодокументированные (или недокументированные) возможности операционной системы по работе с исполняемыми файлами.

Сжатый исполняемый файл состоит из модуля распаковки/инициализации и блока данных, содержащего в себе исходный файл в сжатом виде. При запуске модуль распаковки выделяет память и распаковывает туда содержимое блока данных. Для некоторых

платформ распаковка в память невозможна, в этом случае используется распаковка во временный файл.

Рассмотрим применение программы upx-3.96-win64 (последняя версия на момент рассмотрения). Эта версия программы не всегда правильно (после упаковки программы не запускаются) упаковывает даже элементарные файлы под x64 программирование.

Для получения справочной информации об этой программе необходимо выполнить:

1. Переписать на диск C папку UPX со всем содержимым;
2. Вызвать командную строку: Поиск/ cmd/Командная строка;
3. Перетянуть мышкой на окно командной строки программу upx.exe и нажать Enter. В результате содержимое командной строки будет иметь вид, как показано на рис. 12.1.

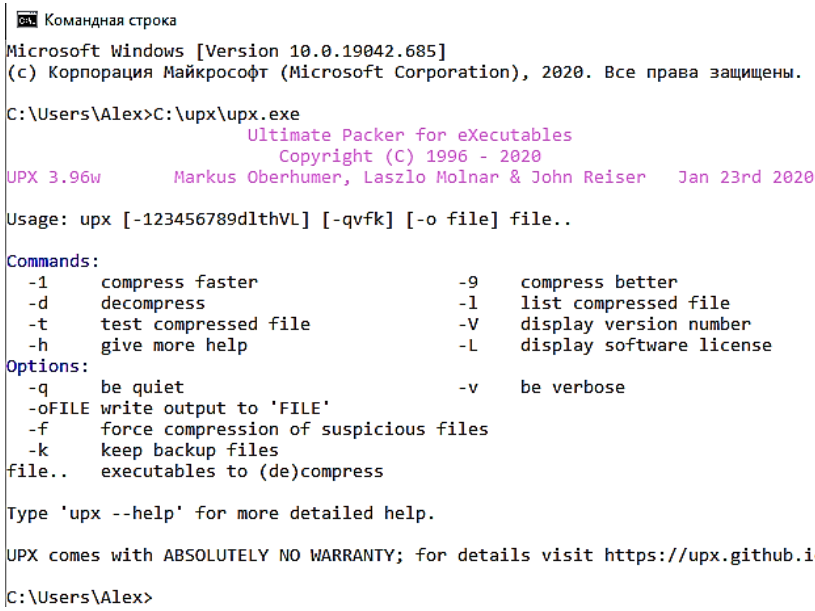


Рис. 12.1. Содержимое командной строки при вызове программы upx

Рассмотрим процесс применения протектора upx. Для этого напишем программу 12.2 без использования функций (т.к. upx не обрабатывает корректно программы с функциями под x64). В программе

осуществляется передача параметров через стек и их последующее сложение. В этой программе специально не применяются другие API-функции, кроме ExitProcess.

Программа 12.2. Передача параметров через стек и их последующее сложение:

```
    include \masm64\include64\masm64rt.inc
.data
    mas1 dq 1,2,3,4
.code
entry_point proc
    push mas1
    push mas1+1
    pop r8
    pop r9
    add r8,r9
invoke ExitProcess,0
    entry_point endp
    end
```

Присвоим этому файлу имя 111.asm. В среде masm64 получим exe-файл с именем 111.exe объемом 3072 байта.

Для применения упаковщика необходимо выполнить:

1. Переписать на диск C папку UPX со всем содержимым;

2. Переписать в эту папку свою программу с именем 111.exe, над которой будут осуществляться соответствующие действия;

3. Вызвать командную строку: Поиск/ cmd/Командная строка;

4. Перетянуть мышкой на окно командной строки программу upx.exe.

5. В командной строке после записи upx.exe сделать один пробел и для компрессии вставить команду -1 C:\upx\111.exe:

```
C:\Users\Alex>C:\upx\upx.exe -1 C:\upx\111.exe
```

Нажать Enter.

В результате содержимое командной строки будет иметь вид, как показано на рис. 12.2.

После упаковки размер файла стал 2560 байтов. Если открыть в отладчике x64Dbg первоначальный файл с именем 111.exe, который хранится по адресу c:\masm64\bin64\ и упакованный файл в другом окне отладчика x64Dbg, который хранится по адресу c:\upx\, то можно сравнить коды этих двух файлов.

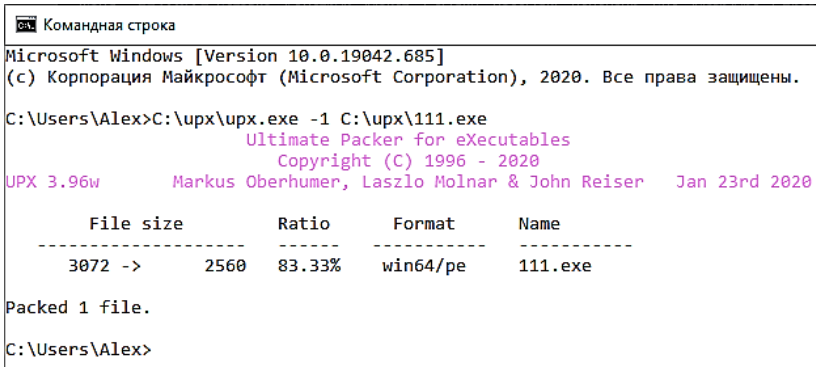


Рис. 12.2. Содержимое командной строки после успешной упаковки

Для распаковки упакованного файла необходимо ввести команду –d C:\upx\111.exe.

В результате содержимое командной строки будет иметь вид, как показано на рис. 12.3.

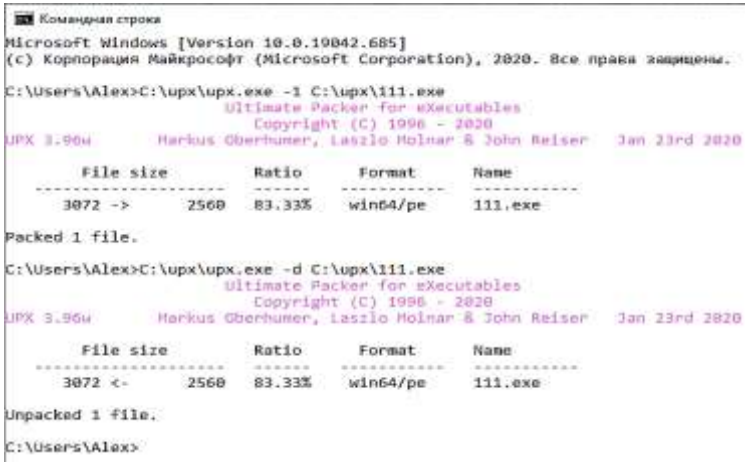


Рис. 12.3. Содержимое командной строки после успешной распаковки

В результате распаковки размер файла стал таким же, как до упаковки.

Программа Upx с оконным интерфейсом называется UpxVis.

12.3.2. Упаковщик ASPack

ASPack – программа для сжатия исполняемых файлов для Windows **платформы x32**, имеет внутренний архиватор. Автор – Алексей Солодовников.

Преимущества программы-упаковщика ASPack:

– наличие версии для архитектуры x64 (http://aspack.com, ASProtect 64 Protection Studio 5.50.28.108);

– уменьшение занимаемого места (кроме ассемблерных файлов);

– применяемое сжатие защищает программы от непрофессионального реверс-инжиниринга;

– совместимость с программами, которые созданы с помощью Microsoft Visual C++, Visual Basic, Delphi, C++ Builder;

– создание резервной копии файла (BAK файл) перед началом упаковки.

Пример интерфейса программы ASPack v.2.12 приведен на рис. 12.4.

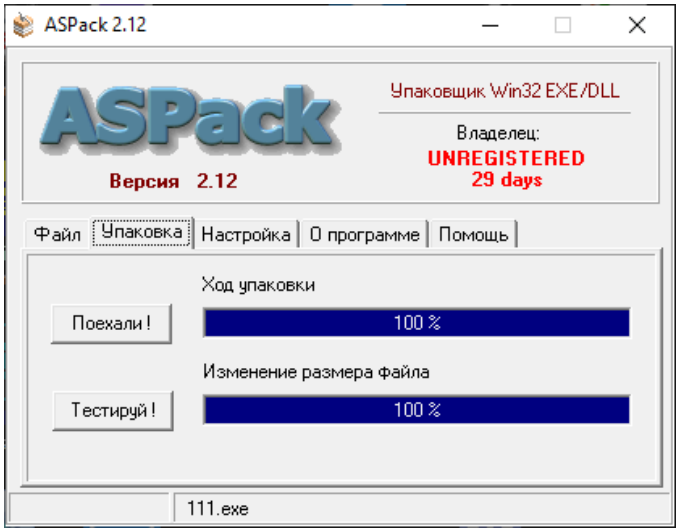


Рис. 12.4. Интерфейс программы ASPack v.2.12

Настройки программы приведены на рис. 12.5.

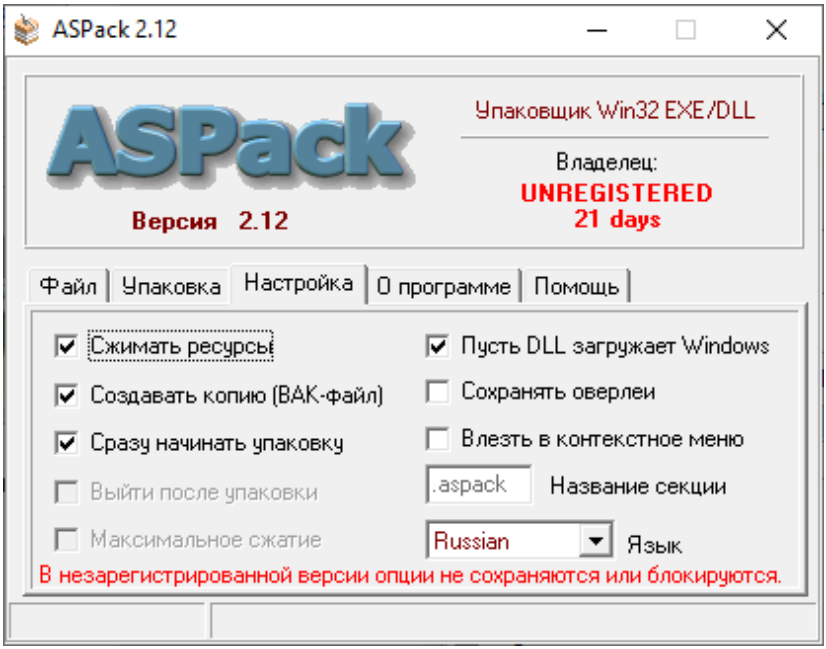


Рис. 12.5. Настройки программы ASPack v.2.12

Пункт меню «Настройки» позволяет проанализировать возможности программы по управлению защитой программы.

Недостаток версии ASProtect 64 – сложность процесса настройки интерфейса, неоднократное обращение к серверу с загрузкой дополнительных файлов, неоднозначная обработка exe-файлов среды masm64 (работа с этой программой будет рассмотрена в другом разделе). **А если программа обращается к серверу, то это свидетельствует о том, что автор такой разработки не уверен в защите своей программы от взлома и не хочет постоянно усовершенствовать ее защиту. Кроме того, если одну из версий удастся взломать, то станет понятен алгоритм ее работы и потребность в новых версиях будет неочевидна.**

12.3.3. Упаковщик PECompact

PECompact – программа для сжатия исполняемых файлов для Windows **платформы x32**. В программе используется оригинальный алгоритм сжатия JCALG1, позволяющий значительно уменьшать размер исполняемого файла. Допускается оставлять несжатыми некоторые из компонентов файла - иконки, информацию о версии или любые другие выбранные ресурсы. Кроме сжатия, PECompact позволяет оптимизировать структуру файла и тем самым уменьшить его размер без сжатия (опция "Trim Only").

Интерфейс главного окна приведен на рис. 12.6.

Рис. 12.6. Интерфейс главного окна

Окно параметров программы приведено на рис. 12.7.

12.4. Программы-протекторы

12.4.1. Назначение программ-протекторов

Основным инструментом, применяемым для защиты приложений от взлома, являются программы-протекторы. Защита, реализуемая большинством протекторов, заключается в том, что они упаковывают и/или шифруют исходный исполняемый файл, уделяя основное внимание применению методов защиты и к процедуре распаковки/расшифровки файла, и к самой программе.

При рассмотрении программ, которые затрудняют исследование кода нельзя не вспомнить программу **ORiEN**, которая применялась для

защиты кода, написанного для **платформы x32**. В этой программе интерес представляют ее настройки (рис. 12.8).

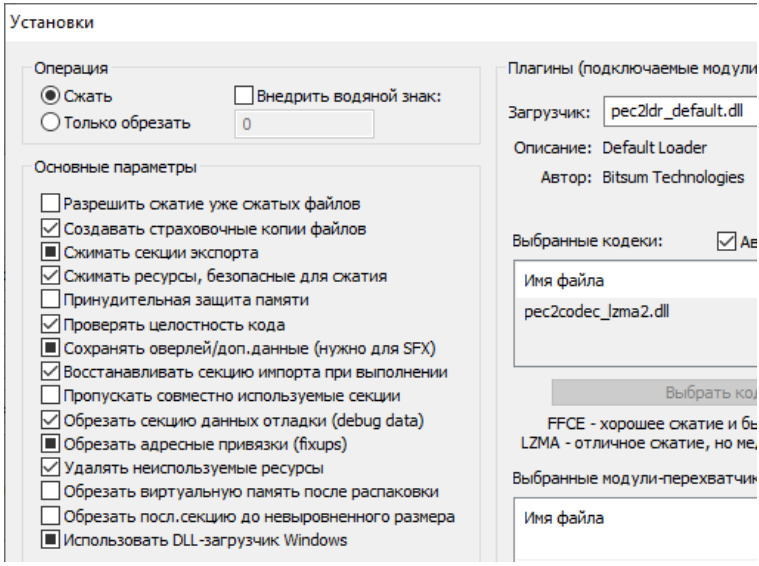


Рис. 12.7. Окно параметров программы

В зависимости от выбранных пунктов программа ORiEN осуществляет защиту исходного кода и привязку к конкретному компьютеру. И хотя выбор пунктов защиты не очень велик, эти пункты позволяют сформировать представление о некоторых возможностях протекторов.

12.4.2. Основные характеристики программ-протекторов

Существует достаточно большое количество характеристик программ-протекторов (https://sohabr.net/habr/post/215553/), к основным из которых можно отнести следующие:

– мутация кода – это замена ассемблерных инструкций на их полные аналоги. Мутация кода предназначена для усложнения поиска типичных сигнатур кода, которые содержатся в используемых библиотеках и типичны для конкретных компиляторов. Мутация (metamorf) предполагает переупорядочивание инструкций без изменения их результирующего действия;

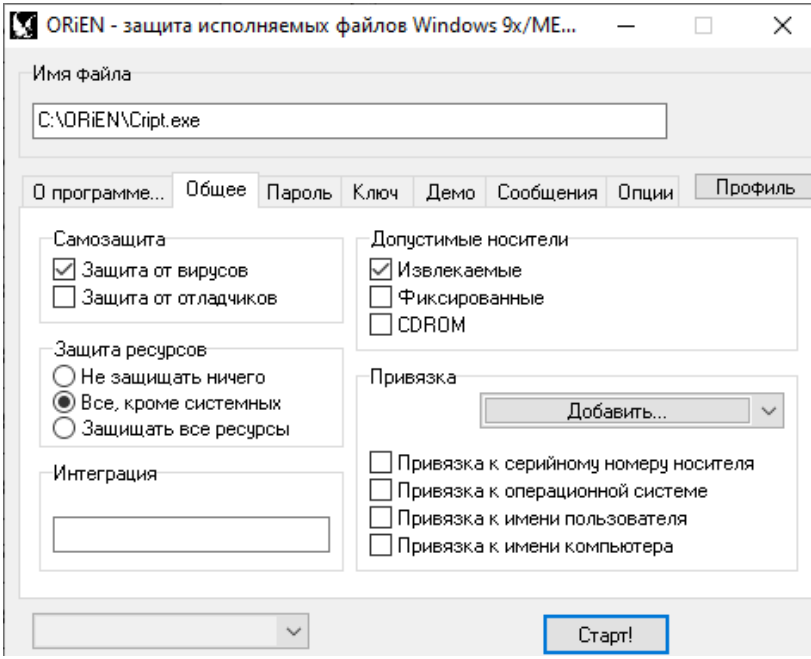


Рис. 12.8. Настройки программы ORiEN

– виртуализация кода – замена машинного кода на байт-код виртуальной машины, которая встраивается в исполняемый файл или динамическую библиотеку. Этот метод защиты обычно представляет серьезную проблему для взломщиков. Протекторы, использующие виртуализацию, обычно генерируют новую систему байт-кодов виртуальной машины каждый раз при обработке защищаемого приложения, что делает “кряки” и генераторы ключей для старых версий ПО несовместимыми с каждой новой версией;

– обнаружение отладчика – функция позволяет запрещать выполнение защищаемого приложения под различными отладчиками. Существует большое количество способов антиотладки;

– обнаружение VM – функция позволяет запрещать выполнение защищаемого приложения в операционной системе, которая установлена на виртуальной машине, такой, например, как VMWare или VirtualPC;

– защита памяти – функция позволяет защищать приложение от атаки путем снятия дампа памяти работающего приложения. Основной целью защиты памяти является запрет доступа процесса к той памяти, которая не выделена для этого процесса. Такие запреты повышают надёжность работы как программ, так и операционных систем, так как ошибка в одной программе не может повлиять непосредственно на память других приложений;

– проверка целостности – функция позволяет обнаруживать попытки модификации кода и данных приложения во время выполнения. Повреждение файлов может возникать из-за сбоев, из-за ошибок в программах или из-за преднамеренного внедрения чужого кода. Целостность файлов часто основывается на CRC (специальное число, так же именуемое контрольной суммой, рассчитываемой на основе данных файла);

– защита импорта – функция позволяет прятать информацию об используемых в программе API-функциях. Вызовы любых функций в процессе из сторонних DLL выполняется через таблицу импорта функций. Эта таблица заполняется при загрузке DLL в процесс и в ней прописываются адреса всех импортируемых функций, которые процессу могут понадобиться. Соответственно, для того, чтобы перехватить вызов API-функции, необходимо найти таблицу импорта, в ней – функцию, которую хотим перехватить, сохранить хранящийся там адрес (указатель на тело функции) в какую-нибудь переменную (чтобы иметь возможность самостоятельно вызвать оригинал), после чего поместить туда указатель на свою функцию;

– защита ресурсов – функция предотвращает извлечение и модификацию ресурсов приложения (в системах Win32/Win64);

– dll boxing – функция позволяет включать в защищаемое приложение библиотеки DLL, требуемые для его работы, при этом DLL входят в состав исполняемого файла и при выполнении приложения загружаются из памяти без промежуточной записи на диск.

В отличие от статических файлов .lib, библиотеки DLL не присоединены непосредственно к выполняемым файлам с помощью редактора связей. В выполняемый файл занесена только информация об их местонахождении. В момент выполнения программы загружается вся библиотека целиком. Благодаря этому разные процессы могут пользоваться совместно одними и теми же библиотеками, находящимися в памяти. Такой подход позволяет сократить объем памяти, необходимый для нескольких приложений, использующих много общих библиотек, а также контролировать размеры EXE-файлов;

– размер защищенных файлов – часто обработка исполняемых файлов сильно увеличивает размер защищаемого файла. В настоящее время объемы исполняемых файлов и дистрибутивов обычно не являются проблемой, но бывают случаи, когда размер защищенного файла является важным критерием. Данная характеристика может изменятся раз от раза при обработке приложения, но средняя величина обычно остается неизменной;

– упаковка защищаемых файлов – при защите приложений протектор может также производить упаковку защищаемого файла для уменьшения его размера. Эта возможность может быть важна, если использование протектора приводит к значительному увеличению размера защищаемого файла;

– поддержка x64 – 64-разрядная платформа разработки программ. Большая часть протекторов под x32 уже не развиваются. А те, которые функционируют под архитектуру x64, являются платными;

– поддержка .NET, Linux, Mac – дополнительные возможности по защите приложений .NET, а также приложений под операционные системы Linux и Mac;

– консольные утилиты – наличие у протектора консольных утилит (Win32), которые позволяют автоматизировать процесс защиты приложения;

– короткие серийные номера – поддержка протектором коротких серийных номеров (вида XXXX-XXXX-XXXX-XXXX) вместо длинных (размером в килобайт и более) ключевых файлов. Иногда короткие серийные номера называют кодами активации, чтобы отличать их от длинных ключей;

– деактивация ключей – функция деактивации позволяет пользователю перенести программу на другой компьютер и активировать ее заново;

– блокировка ключей – функция позволяет блокировать «утекшие» ключи. Особенно это полезно, если не используется привязка к оборудованию. Обычно заблокированные ключи (вернее, обычно используются их хэши) зашиваются в защищаемое приложение при его обработке протектором;

– данные о покупателе – хранение данных покупателя (имя и email в ключе);

– дополнительные данные – возможность добавления каких-либо данных в ключе для дальнейшего использования в защищаемом приложении;

– поддержка различных режимов активации –функция применяется для организации различных режимов работы защищаемого приложения.

К примеру, одно и то же приложение может быть активировано в режиме «Trial», в обычном режиме («Std») и в режиме «Pro»;

– привязка к компьютеру – часто требуется привязать программу к определенному компьютеру, чтобы ограничить ее нелегальное распространение. В то же время идентификатор оборудования не должен зависеть от установленной системы или ее версии, а также от аппаратных компонентов, которые наиболее подвержены возможности их замены;

– наличие сервера активации – сервер активации обычно предполагает, что для приложения требуется активация. Данная опция помогает ограничивать нелегальное распространение ПО. В случае, если ключевой файл активации формируется единожды для заданного имени пользователя и/или его email'a существует возможность утечки данного ключевого файла в широкий доступ. Необходимость активировать приложение через сервер активации каждый раз при его установке снижает опасность нелегального распространения ПО совместно со скомпрометированными ключевыми файлами;

– генератор ключей – дает возможность генерировать ключи для активации приложения в автоматическом режиме. Обычно это требуется для генерации ключей/серийников/кодов активации при продаже ПО через регистраторов. При этом генератор ключей располагается на вашем сайте и вызывается регистратором при осуществлении продажи ПО. Генератор ключей может быть приложением Win32, Linux программой, или может быть реализован на языках типа PHP. Последний вариант наиболее предпочтителен, так как не накладывает никаких ограничений на тип хостинга, на котором он будет располагаться;

– поддержка регистраторами – многие регистраторы предоставляют интегрированные в свою систему генераторы ключей. Такая возможность может пригодится, если по каки-либо причинам нет возможности или желания устанавливать генератор ключей на свой сервер;

– ложные срабатывания (false positivies) – эта характеристика протектора говорит о том, как часто защищенные протектором приложения могут быть определены антивирусным ПО как зловредные. Большинство разработчиков систем защиты взаимодействуют с производителями антивирусов, и стараются уменьшить количество ложных срабатываний при использовании протектора, но протекторы, разработка/поддержка которых приостановлена или заброшена, могут давать много ложных срабатываний;

– стабильность – определяет, насколько часто защищаемые приложения становятся неработоспособны, насколько долго нужно затратить времени над ключами и опциями защиты, чтобы получить защищенное работоспособное приложение;

– простота интеграции – данная характеристика отражает простоту встраивания защиты и системы активации в приложение. Поддержка системы активации включает в себя API для отображения диалогов активации/деактивации, API для сохранения ключевой информации в системе пользователя, и, по возможности, различные функции управления различными аспектами подсистемы безопасности Windows;

– поддержка инсталяторов – характеристика отражает простоту встраивания системы активации приложения в инсталятор продукта. В некоторых случаях бывает удобно для конечных пользователей встраивать активацию приложения в инсталятор продукта, особенно если лицензии на использование приложения продаются пачками на 10-50-100-1000000 допустимых активаций. В подобных случаях особенно необходима бывает возможность silent-установки и silent-активации по заданным в командной строке параметрам.

Приведенные параметры используются для сравнения возможностей различных программ-протекторов.

12.4.3. Протектор VMProtect

Основными методами защиты программного кода, применяемыми VMProtect, являются изменение кода, смешанный метод защиты, и последующая виртуализация. Виртуальная машина, на которой выполняются виртуализированные фрагменты кода, встраивается в результирующий код защищаемого приложения.

Виртуализация кода – следующий шаг в защите программного обеспечения. Большинство систем защиты шифруют код, а затем расшифровывают его при запуске приложения. VMProtect не расшифровывает код. Вместо этого зашифрованный код запускается на виртуальном процессоре, который заметно отличается от обычных процессоров x86 и x64, поскольку набор команд отличается для каждого защищенного файла.

В настоящее время **актуальной** является версия VMProtect Ultimate v 3.4.0, интерфейс которой приведен на рис. 12.9.

Эта версия программы успешно обрабатывает exe-файл, полученный в среде masm64.

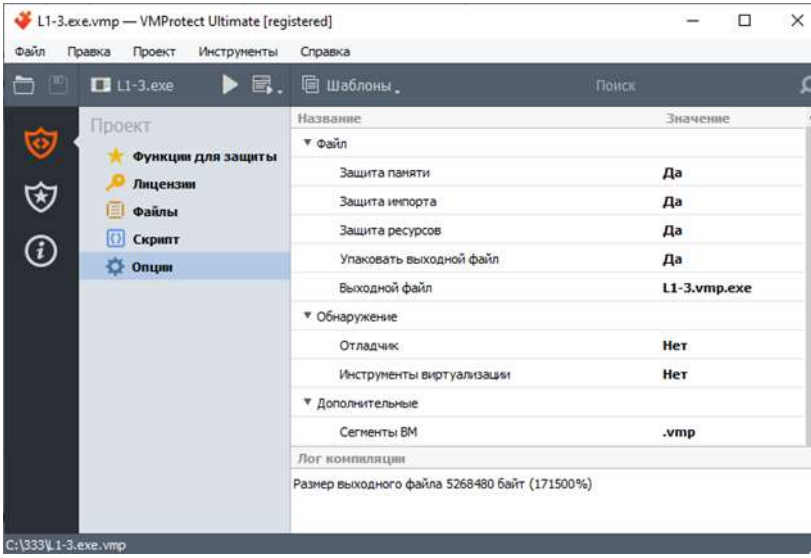


Рис. 12.9. Интерфейс программы VMProtect Ultimate v 3.4.0

Для получения обфускационного кода необходимо выполнить действия:

– загрузить exe-файл. Для этого выполнить: Открыть и выбрать файл;

– нажать на иконку треугольника.

Полученный exe-файл из-за добавления навесной защиты получается намного большего объема, чем исходный.

Содержимое полученного файла можно исследовать в отладчике.

Этот протектор на сегодняшнее время является наиболее доступным для использования.

12.4.4. Протектор ASProtect

ASProtect – это программа-протектор, т.е. программный инструмент, предназначенный для защиты других программ. ASProtect представляет собой коммерческую систему защиты программного обеспечения, использующую такие средства, как сжатие исполняемых

файлов, шифрование, установка ограничений по времени использования и количеству запусков. Механизм защиты взят из вируса Marburg.

Для того, чтобы защитить исполняемый файл, протектор преобразует его содержимое, применяя упаковщик. Основное назначение упаковщика – это уменьшение размера сжимаемой программы. Однако, так происходит не всегда.

С другой стороны – упаковку можно считать специфическим видом шифрования и сжатия файла. По этой причине ASProtect иногда называется «паковщиком файлов» (packer). При запуске защищенной программы управление сначала получает код протектора. Протектор выполняет предусмотренные проверки и расшифровывает в памяти необходимые области. После этого протектор передает управление на оригинальную точку входа (Original Entry Point), и начинается выполнение основной программы.

Протектор ASProtect обеспечивает выполнение следующих функций:

– ограничение периода работы пробной версии;

– ограничение функциональности пробной версии;

– динамическая расшифровка фрагментов кода при наличии правильного регистрационного кода;

– различные методы защиты от снятия протектора;

– различные методы противодействия исследованию под отладчиком.

Реализация конкретного варианта защиты определяется дополнительным кодом, внесенным в программу, и установленными опциями ASProtect при обработке исполняемого файла. Протектор реализовывает внутреннюю защиту с привязкой к оборудованию, т.е. к конкретному компьютеру пользователя. Для этого ASProtect использует специальный идентификатор оборудования Hardware_ID, который зависит от комплектации компьютера и номеров оборудования. Этот идентификатор вычисляется программно функциями, имеющимися в библиотеках dll, поставляемых вместе с ASProtect. Для работы пользователя с защищенной программой должна быть выполнена ее регистрация у разработчика. Защищенный файл при отсутствии регистрации должен всегда запускать форму регистрации. Пользователь вводит в форму регистрационное имя, и вместе с идентификатором оборудования Hardware_ID отсылает его разработчику. Разработчик запускает протектор, вводит полученный Hardware_ID и регистрационное имя, и генерирует регистрационный ключ. Этот ключ отсылается пользователю, который и вводит его для регистрации своей программы. Если регистрация прошла успешно, форма регистрации

больше не появляется, и программа работает в соответствии с основным кодом.

При попытке установить программу на другой компьютер регистрацию нужно проводить заново, поскольку Hardware_ID у каждого компьютера свой. При обработке протектором защищаемой программы всегда создается проект, который содержит все настройки опций защиты. Кроме того, проект защиты содержит ключи шифрования, которые используются для последующей генерации регистрационных ключей. Поэтому проект защиты должен быть обязательно сохранен. Каждый проект защиты имеет уникальные ключи шифрования. Генерация регистрационного ключа в ASProtect происходит на закладке Activation Keys. Для этого нужно ввести Hardware_ID и регистрационное имя (Name For Registration), присланное пользователем, после чего нажать кнопку Create. В поле Registration Key появится ключ для регистрации, который и отсылается пользователю.

ASProtect 64 предназначен для профессиональной защиты от несанкционированного использования, промышленного/домашнего копирования, профессионального взлома и анализа программных продуктов.

Для установки защиты на приложения служит программный комплекс Protection Studio, являющийся одним из компонентов системы защиты ASProtect 64. **Защита файлов выполняется на сервере StarForce**.

Защита ASProtect 64 предусматривает привязку к компьютеру и технологию активации защищённого продукта.

Пример интерфейса программы ASProtect 64 Protection Studio 5.50.28.108 приведен на рис. 12.10.

Обращение к серверу, естественно, резко увеличивает устойчивость программы от взлома, однако свидетельствует о неуверенности разработчиков в применяемых приемах самозащиты программы и нежелании постоянного ее совершенствования.

Из-за перечисленных особенностей этот протектор не находит широкого применения в повседневной деятельности обычных пользователей.

12.4.5. Протектор Enigma

Протектор Enigma содержит широкий набор уникальных функций и возможностей, основной целью которых является защита исполняемых

файлов от нелегального копирования, взлома, модификации и исследования для программ платформ x32/x64. Поддерживаются практически любые типы исполняемых файлов (exe, src, dll, ocx, bpl и другие) и .NET файлов.

Рис. 12.10. Интерфейс программы ASProtect 64 Protection Studio 5.50.28.108

В протекторе применяются криптостойкие алгоритмы создания регистрационных ключей, различные методы обфускации, мутации и виртуализации кода. Протектор реализован в двух версиях: demo и платной (рис. 12.11).

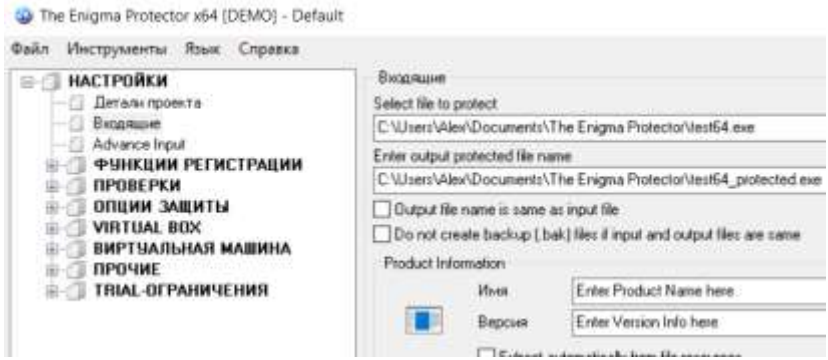


Рис. 12.11. Интерфейс протектора Enigma

В demo-версии при запуске обработанного файла появляется дополнительное окно с рекламой протектора Enigma.

Этот протектор является наиболее практичным в использовании.

СПИСОК ЛИТЕРАТУРЫ

1. Касперски Крис, Рокко Ева. Искусство дизассемблирования / Касперски Крис, Рокко Ева. – Санкт-Петербург, БХВ-Петербург, 2008 – 896 с.

2. Методические указания к выполнению лабораторных работ по курсу «Реверсное программирование», Ч.1 «Внедрение кода» для студентов специальностей: 123 – «Компьютерная инженерия», 125 – «Кибербезопасность» / составитель А.Н. Рысованый. – Х. : «Слово», 2019. – 84 с.

3. Методические указания для выполнения лабораторных работ по курсу «Реверсное программирование», Ч.2 «Крэкинг: практика взлома простых программ. Среда программирования masm64» для студентов специальностей: 123 – «Компьютерная инженерия», 125 – «Кибербезопасность» всех форм обучения/ составитель А.Н. Рысованый. – Х. : «Слово», 2020. – 92 с.

4. Панов А.С. Реверсинг и защита программ от взлома / А.С. Панов. – Санкт-Петербург, БХВ-Петербург, 2006 – 256 с.

5. Рысованый А.Н. Реверсное программирование. Антиотладочные приемы защиты от реверса. Среда программирования masm64 : учебное пособие для студентов специальностей 123 «Компьютерная инженерия», 125 «Кибербезопасность»/ А.Н. Рысованый. – Харьков : «Слово», 2020. – 112 с.

6. Рысованый А.Н. Системное программирование, Ч.1. Программирование в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 108 с.

7. Рысованый А.Н. Системное программирование, Ч.2. Расширенные возможности программирования в среде masm64 : учеб.-метод. пособие / А.Н. Рысованый. – Харьков : «Слово», 2017. – 140 с.

8. Рисований О.М. Системне програмування : підручник для студентів напряму “Комп’ютерна інженерія” вищих навчальних закладів / О.М. Рисований. – Харків : НТУ “ХПІ”, 2010. – 912 с.

9. Сикорски М., Хониг Э. Вскрытие покажет! Практический анализ вредоносного ПО. — СПб.: Питер, 2018. – 768 с.

10. Эриксон Д. Хакинг: искусство эксплойта. 2-е изд . – СПб.: Питер, 2018. – 496 с.

Навчальне видання

РИСОВАНИЙ Олександр Миколайович

«РЕВЕРСНЕ ПРОГРАМУВАННЯ»

Впровадження коду.
Середовище програмування masm64

Навчальний посібник

Російською мовою

Роботу до видання рекомендував *М.Й. Заполовський*
Відповідальний за випуск *С.Г. Семенов*

В авторській редакції

План 2021 р.

Формат 60x84 1/16. Папір офісний.
Riso-друк. Гарнітура Times. Ум. друк. арк. 15. Наклад 50 прим.
Зам. № Ціна договірна.

Видавничий центр НТУ «ХПІ»
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.
61002, Харків, вул. Кирпичова, 2

Віддруковано в друкарні «Слово»
Адреса: 61002, Харків, вул. Лермонтовська, 27.

РЫСОВАНЫЙ
Александр Николаевич –

Родился 22.09.1957, русский.
Закончил с отличием Харьковское высшее военное авиационное командное училище связи в 1977 г., Военно-воздушную инженерную орденов Ленина и Октябрьской революции Краснознаменную академию
им. проф. Н.Е. Жуковского в 1984 г., г. Москва.
Специальность "Автоматизированные системы управления", квалификация "военный инженер системотехник".

Служба в Вооруженных Силах – 31 год: ХВВАКУС; в/ч пп 81524, Wittstock/Dosse, Германия; ВВИА, г.Москва; ХВВАУРЭ, г.Харьков; ХВУ; ХУВС.
Воинское звание – полковник.

Служил в должностях: командир взвода связи с ЗАС, начальник смены группы оборудования радиолокационного поста с пунктом наведения 266 УАСУ 33 иап 16 иад 71 ИАК 16 ВА, начальник группы автоматизации 314 ОБАУ 33 иап 16 иад 71 ИАК 16 ВА, слушатель ВВИА, начальник лаборатории, начальник смены вычислительного центра, преподаватель, доцент. Последняя должность: доцент кафедры компьютерных систем факультета АСУ войск ПВО ХУВС.

Работает в НТУ "ХПИ" с 2006 г.
На должности проф. кафедры "Вычислительная техника и программирование" с 2013 г., профессор НТУ ХПИ с 2015 г.
Кандидат технических наук. Диссертацию защитил в 1995 г.
в специализированном совете Харьковского военного университета.
Доцент с 1998 г.
Имеет более 300 научных публикаций, из них 6 учебников и 3 учебных пособия с грифами министерства образования и науки Украины,
более 40 учебных пособий, более 100 изобретений, оформленных в виде авторских свидетельств, патентов, патентов на полезные модели и др.