МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра информационной безопасности

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«14» ноября 2017г.

ОБЗОР РЕ-ФОРМАТА ИСПОЛНЯЕМЫХ ФАЙЛОВ ПЛАТФОРМЫ WIN32

Методические указания по выполнению лабораторной работы по дисциплинам «Основы реверсинжиниринга программных средств», «Методы защиты программного обеспечения» для студентов специальности 10.03.01

Курск 2017

УДК 004.056.55

Составитель А.Л. Марухленко

Рецензент
Кандидат технических наук, доцент *И.В. Калуцкий*

Обзор ре-формата исполняемых файлов платформы win32 Studio: методические указания по выполнению лабораторных работ по дисциплине «Основы реверсинжениринга», «Методы защиты программного обеспечения» / Юго-Зап. гос. ун-т; сост. А.Л. Марухленко. Курск, 2017. 19с.

Рассматривается ре-формат исполняемых файлов платформы win32. Указывается порядок выполнения лабораторной работы и содержание отчета.

Методические указания соответствуют требованиям программы, утвержденной учебно-методическим объединением по образованию в области информационной безопасности (УМО ИБ).

Предназначены для студентов специальности 10.03.01.

Текст печатается в авторской редакции

Подписано в печать 01.11.2017. Формат 60х84 1/16.
Усл.печ. л. 1,1. Уч.-изд.л. 1,0. Тираж 30 экз. Заказ _____. Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

ОГЛАВЛЕНИЕ

1. ЦЕЛЬ РАБОТЫ

Получение представления о структуре исполняемых файлов Win32, ознакомление с PE-заголовком, секциями и таблицами импорта приложений, получение начальных сведений о работе загрузчика Windows NT (ntdll.dll), а также получение навыков работы с редакторами исполняемых файлов и дизассемблерами.

2. СВЕДЕНИЯ О СТРУКТУРЕ ИСПОЛНЕЯЕМЫХ НА ПЛАТФОРМЕ WIN32 ФАЙЛОВ

PE (*Portable Executable*) – это файловый формат исполняемых файлов Win32, спецификации которого происходят от Unix/COFF (*Common Object File Format*). Загрузчик PE распознает и использует этот файловый формат даже когда приложение запускается на не PC CPU платформе.

Общую структуру PE-файла можно представить следующим образом (рис. 1):

DOS MZ-заголовок
Заглушка DOS (stub)
Заголовок PE
Таблица секций
Секция 1
…
Секция n

Рисунок 1– Структура формата PE

Как видно из рисунка, все PE-файлы (в том числе 32-битные с расширением dll) должны начинаться с DOS-заголовка. Обычно он служит для того, чтобы в случае запуска программы из операционной системы, не знающей о PE-формате, такой, как DOS, запускалась DOS-stub. Большинство компиляторов используют Int 21h, сервис 9, чтобы напечатать строку “This program cannot run in DOS mode”. Но поскольку DOS-заглушка – это полноценное DOS-приложение, не исключено другое её применение.

После DOS-stub'a идет PE-заголовок. PE-заголовок – это общее название структуры под названием IMAGE_NT_HEADERS.

Эта структура содержит много основных полей, используемых PE-загрузчиком. В случае, если программа запускается операционной системой, которая знает о PE-формате, PE-загрузчик может найти смещение PE-заголовка в заголовке DOS-MZ. После этого он может пропустить DOS-stub и перейти напрямую к PE-заголовку, который является настоящим заголовком исполняемого файла.

Настоящее содержимое PE-файла разделено на блоки, называемые секциями. Секция – это блок данных с общими атрибутами, такими как код/данные, чтение/запись и т.д. Группирование данных производится на основе их атрибутов. Не играет роли, как используются код/данные, если данные/код в PE-файле имеют одинаковые атрибуты, они могут быть сгруппированы в секцию, т.е. секции могут содержать и данные и код одновременно, главное, чтобы те имели одинаковые атрибуты. Если у вас есть данные, и вы хотите, чтобы они были доступны только для чтения, вы можете поместить эти данные в секцию, помеченную соответствующим атрибутом. Таблица секций, соответственно, располагает данными о каждой секции PE-файла.

Остановимся подробнее на заголовке PE-файла. Адрес PE-заголовка загрузчик получает из DOS-MZ заголовка, который, в свою очередь тоже является структурой. Сама структура IMAGE_NT_HEADERS, представляющая PE-заголовок, определена следующим образом:

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

Поле Signature представляет собой двойное слово 50 45 00 00 (ASCII-символы “PE”, за которыми следуют два нулевых байта).

Далее следуют две структуры, IMAGE_FILE_HEADER и IMAGE_OPTIONAL_HEADER. Приводить их полное описание нецелесообразно, поскольку большинство из их полей не используются, либо используются для отладки. Поэтому ограничимся перечисленными полями в следующих таблицах:

Таблица 1. – Структура файлового заголовка

Имя поля	Описание
Machine	Целевая CPU платформа машины, на котором файл будет исполняться. Для Intel это значение равно константе IMAGE_FILE_MACHINE_I386. Это поле обычно заполняется компилятором, когда он встречает соответствующую директиву (например, .386 в masm).
NumberOfSections	Число секций PE-файла. Оно также определяет количество элементов в таблице секций.
TimeDataStamp	Дата и время создания файла.
SizeOfOptionalHeader	Размер опционального заголовка.
Characteristics	Содержит дополнительные флаги файла, например, является ли он exe или dll.

Таблица 2. – Структура опционального заголовка

Имя поля	Описание
AddressOfEntryPoint	RVA точки входа – первой инструкции, которая будет передана загрузчику после размещения файла в виртуальной памяти.
ImageBase	Предпочитаемый адрес загрузки PE-файла. Файл загружается в указанную область если она не занята никаким другим процессом.
SectionAlignment	Размер выравнивания секций в памяти. Например, если значение в этом поле равно 4096 (1000h), каждая секция должна начинаться по адресу, кратном этому значению. Если первая секция находится в 401000h и его адрес равен 10 байтам, следующая секция должна начинаться в 402000h, даже если адресное пространство между ними останется неиспользованным.

Имя поля	Описание
FileAlignment	Размер выравнивания секций в файле. Например, если значение в этом поле равно 512 (200h), каждая секция должна начинаться на расстоянии от начала файла кратном 512 байтам. Если первая секция в файле находится по смещению 200h и ее размер 10 байт, следующая секция должна быть расположена со смещением 400h: пространство между смещениями 522 и 1024 будет неиспользовано/неопределено.
Minor/Major Subsystem Versions	Версия подсистемы win32. Если PE-файл спроектирован для Win32, версия подсистемы должна быть 4.0, в противном случае диалоговое окно не будет иметь 3D-вида.
SizeOfImage	Общий размер образа PE в памяти. Это сумма всех заголовков и секций, выровненных по SectionAlignment.
SizeOfHeaders	Размер всех заголовков + таблицы секций. То есть это значение равно размеру файла минус комбинированный размер всех секций в файле.
Subsystem	Указывает для какой из подсистем NT предназначен этот PE-файл. Для большинства win32-программ используется только два значения: Windows GUI и Windows CUI (консоль).
DataDirectory	Массив структур IMAGE_DATA_DIRECTORY. Каждая структура дает RVA важной структуры данных в PE-файле, например таблицы адресов импорта (IAT).

Во второй таблице: RVA – *relative virtual address* – относительный виртуальный адрес, расстояние для ссылающейся точки в виртуальном адресном пространстве (т.е. RVA, в отличие от абсолютного VA, не зависит от адреса загрузки образа, который может отличаться от ImageBase).

Примечание: все указанные структуры, имена полей и констант определены в заголовочных файлах Windows.

3. ИССЛЕДОВАНИЕ ФОРМАТА ИСПОЛНЯЕМОГО ФАЙЛА

3.1 Анализ заголовка файла.

Рассмотрим простейшее приложение helloworld.exe, выводящее строку на экран консоли Win32:

Рисунок 2 – Окно простейшего приложения Win32

Несмотря на то, что программа работает в консоли, это полноценное Win32 приложение. Для наглядности ниже приведен его листинг:

```
.386
 .model flat,stdcall
option casemap:none

includelib kernel32.lib

SetConsoleTitleA PROTO :DWORD
GetStdHandle PROTO     :DWORD
WriteConsoleA PROTO
:DWORD,:DWORD,:DWORD,:DWORD,:DWORD
ExitProcess PROTO      :DWORD
```

```
Sleep PROTO              :DWORD
 .const
sConsoleTitle  db 'My Nth Console Application',0
sWriteText  db 'Hello World!!!!'
 .code
Main PROC
  LOCAL hStdout :DWORD
  push offset sConsoleTitle
  call SetConsoleTitleA
  push -11
  call GetStdHandle
  mov hStdout, EAX

  push 0
  push 0
  push 16d
  push offset sWriteText
  push hStdout
  call WriteConsoleA
  push 2000d
  call Sleep
  push 0
  call ExitProcess
Main ENDP
end Main
```

Как можно увидеть из листинга программы, её основная работа состоит в получении дескриптора окна консоли с помощью API функции GetStdHandle и вывода строки в консоль с помощью API WriteConsoleA. API-функция SetConsoleTitleA устанавливает заголовок окна консоли, Sleep – обеспечивает задержку для фиксации результата выполнения программы на экране. Все API вызовы, используемые этой программой, импортируются из системной библиотеки kernel32.dll.

Работу с файлом будем производить с помощью редактора двоичных файлов HIEW. HIEW совмещает функции шестнадцатеричного редактора и дизассемблера, имеет возможности анализа исполняемых файлов (рис. 3).

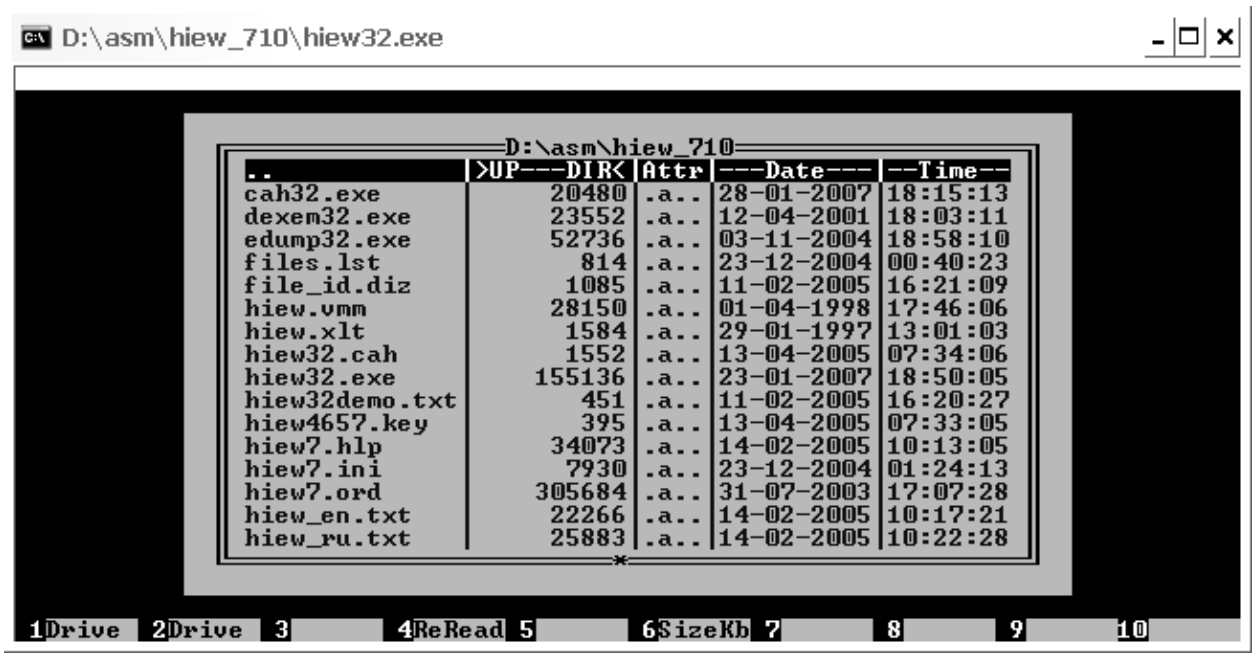

Рисунок 3 – Главное окно программы HIEW

С помощью клавиш навигации находим исследуемый файл, нажимаем Enter для его открытия (рис. 5).

Рисунок 4 – Текстовое представление файла helloworld.exe

По умолчанию файлы открываются для текстового просмотра/редактирования. Для перехода в шестнадцатеричный режим, необходимо нажать F4 (Mode) и выбирать Hex (рис. 5).

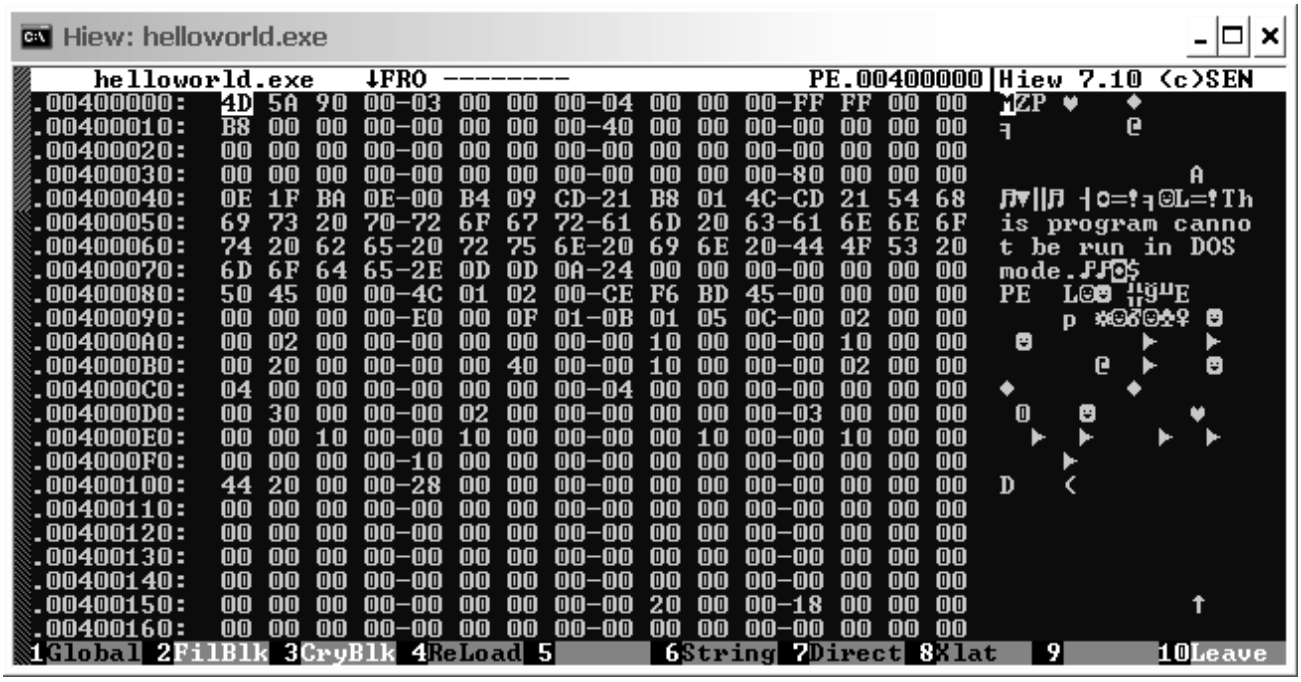

Рисунок 5 – Шестнадцатеричное представление файла helloworld.exe

HIEW автоматически показывает виртуальные смещения слева для всех исполняемых файлов, как будто образ уже загружен в память (виртуальные адреса отмечены точкой слева от смещения). Нажатие Alt+F1 переключает режим отображения смещений между виртуальными адресами и внутрифайловыми смещениями (рис. 6):

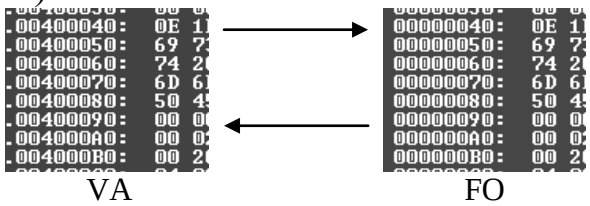

Рисунок 6 — Виртуальные и адресные смещения

Имея теоретические сведения о формате PE-файлов и заголовков, можно кратко проанализировать файл (обратите внимание на то, что для указания любых значений используется обратный порядок байтов):

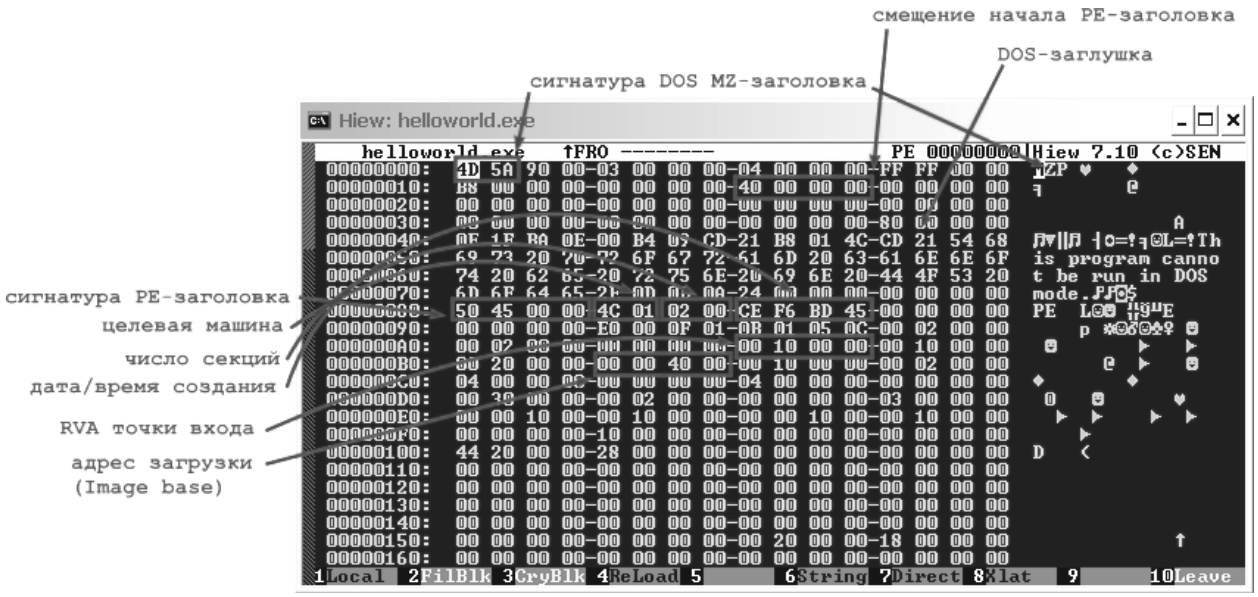

Рисунок 7 –Элементы PE-файла в окне дизассемблера HIEW

Встроенный в HIEW анализатор заголовков позволяет проверить правильность найденных вручную значений. Для этого необходимо нажать F8 (Header):

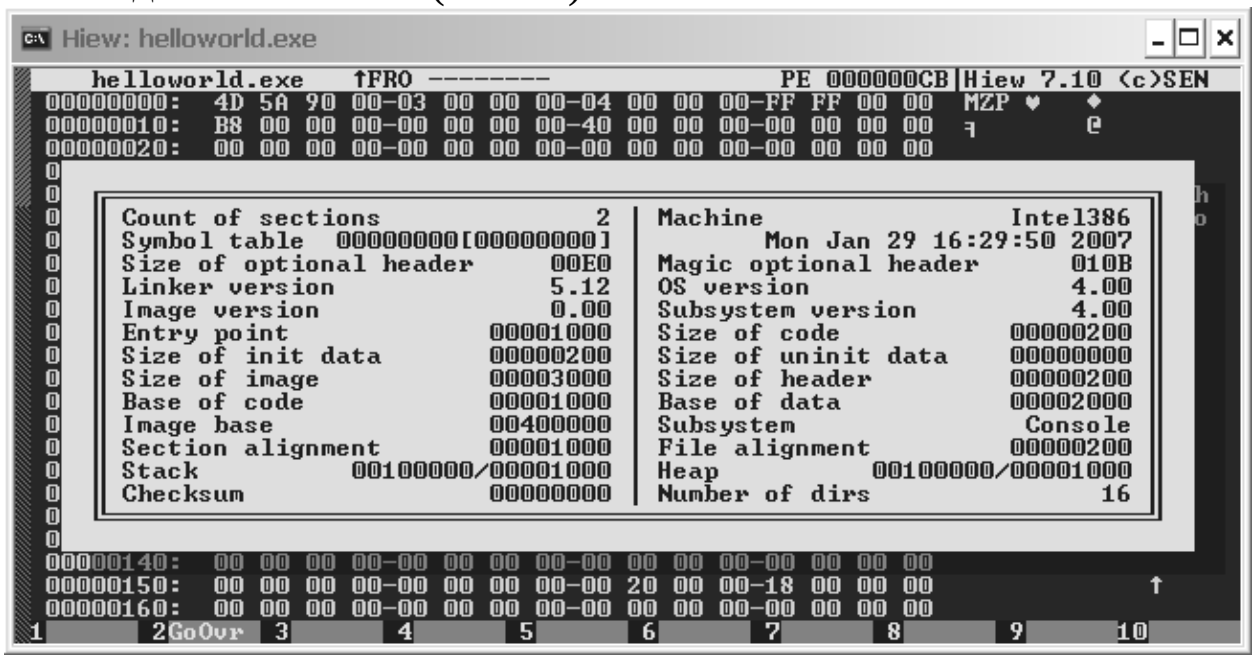

Рисунок 8 – Встроенный анализатор

Полученных сведений достаточно, чтобы определить следующие параметры исполняемого файла:

а) целевая машина – I386;

б) число секций – 2;

в) целевая подсистема – Windows CUI (Console);

г) виртуальный адрес загрузки образа – 0x00400000;

д) виртуальный адрес и абсолютное внутрифайловое смещение точки входа:

- RVA = 1000h,
- VA = ImageBase + RVA = 0x00400100;

Чтобы найти абсолютное внутрифайловое смещение точки входа, необходимо нажать F5 (Entry), после чего происходит переход на точку входа.

е) внутрифайловое смещение точки входа равно 0x0200.

При нахождении на точке входа, нажатие F4 (Mode) и выбор режима Decode отобразит на экране дизассемблированный код приложения (рис. 9).

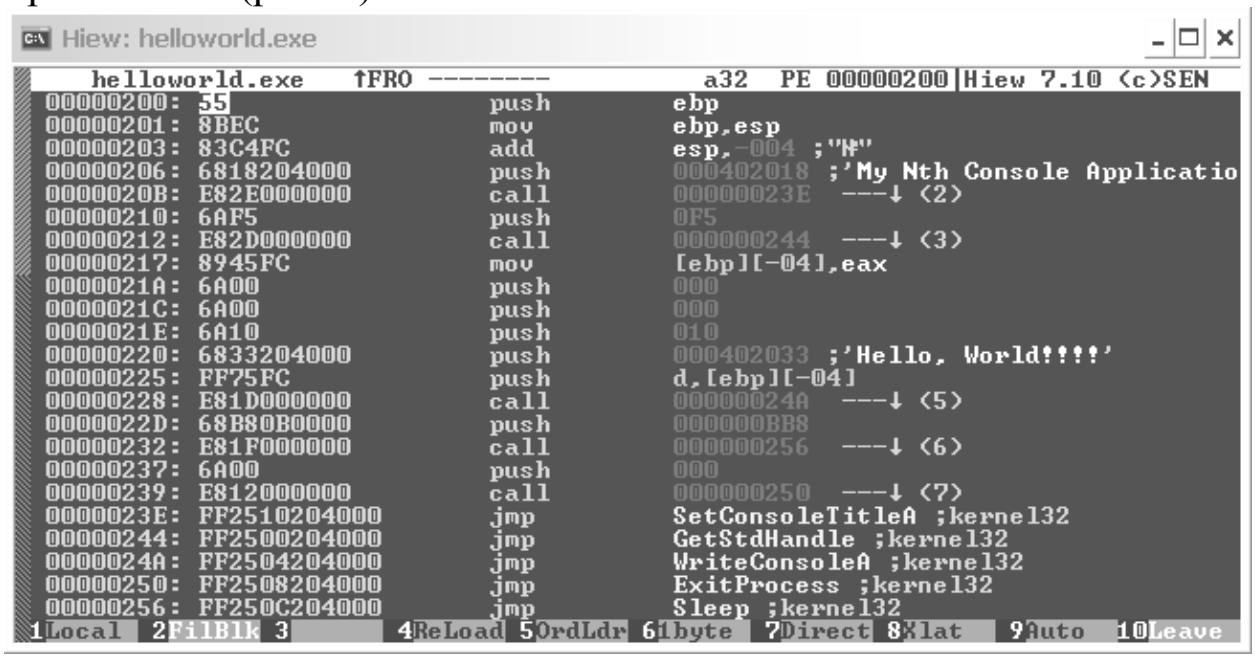

Рисунок 9 – Дизассемблированный код приложения

Поскольку по смещению 0x0200 находится точка входа нашего приложения, первой инструкцией является операция помещения в стек указателя базы кадра стека:

```
00000200: 55        push        ebp
```

Системе безразлична конечная цель применения тех или иных данных. Это означает, что если точка входа указывает, скажем, на секцию с данными, то процессор будет трактовать, например, строковые данные как код и пытаться его исполнить, что может привести к непредсказуемым последствиям. Так же и дизассемблер будет пытаться дизассемблировать любой код, начиная с указанного байта, и если инструкция имеет, например, 3-хбайтный

опкод, а дизассемблирование начинается со второго байта, то результаты дизассемблирования будут неверными.

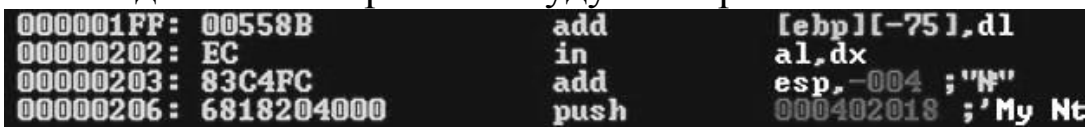

Рисунок 10 – Результат неверного дизассемблирования

На рисунке показан пример неверного дизассемблирования. Точка входа имеет адрес 0x200, а дизассемблирование начинается с адреса 0x1FF трёхбайтной инструкцией. Для того, чтобы получить верное отображение файла в декодированном виде, следует поставить указатель на правильный байт (в нашем случае, по адресу 0x200) и нажать клавишу «/» (current offset at top). В этом случае дизассемблирование начнётся с правильного места.

3.2 Секции.

Как уже было выяснено из анализа заголовка, исследуемое приложение содержит две секции. Их описание содержится в таблице секций, описываемой массивом структур IMAGE_SECTION_HEADER.

```
IMAGE_SECTION_HEADER STRUCT
      Name1 db IMAGE_SIZEOF_SHORT_NAME dup(?)
      union Misc
         PhysicalAddress dd ?
         VirtualSize dd ?
      ends
      VirtualAddress dd ?
      SizeOfRawData dd ?
      PointerToRawData dd ?
      PointerToRelocations dd ?
      PointerToLinenumbers dd ?
      NumberOfRelocations dw ?
      NumberOfLinenumbers dw ?
      Characteristics dd ?
   IMAGE_SECTION_HEADER ENDS
```

Нас интересуют имя секций, адреса их начала, а также их виртуальные и физические размеры. Иногда интерес также

представляет поле Characteristics, используемое для назначения атрибутов секций (например, возможность чтения или записи для секции).

Рассмотрим таблицу секций приложения. Таблица секций начинается по адресу .00400178 именем первой секции (.text). Вторая запись начинается по адресу .04001A0 именем второй секции (.rdata) (рис. 11)

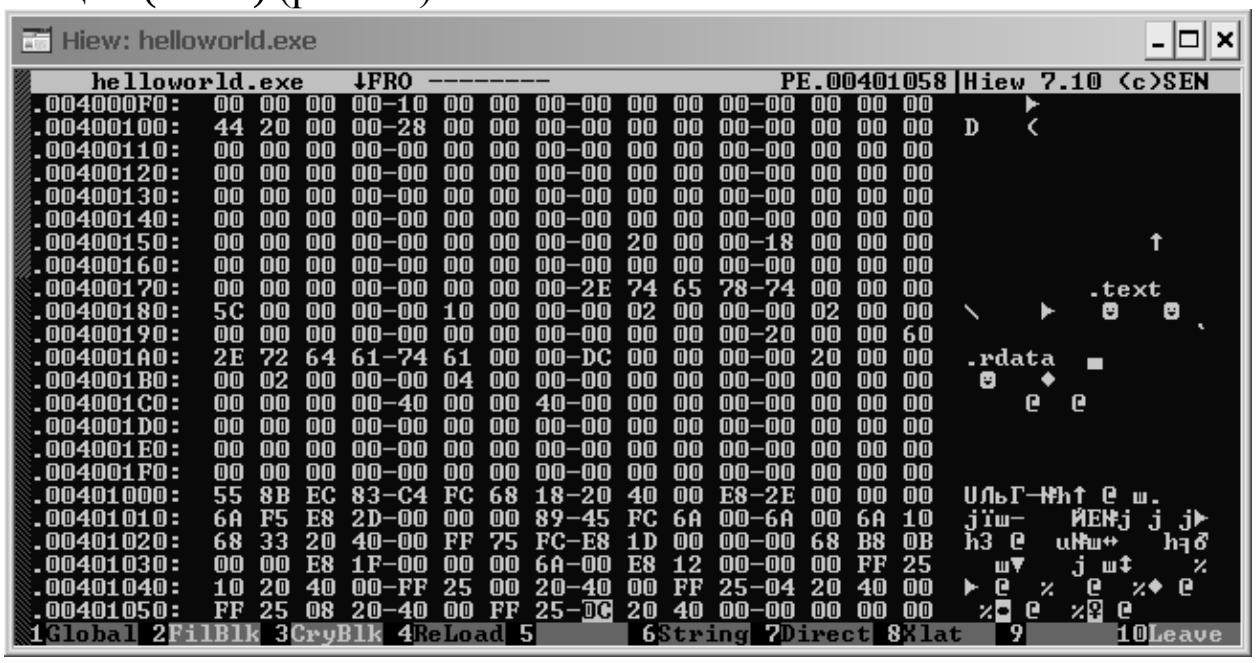

Рисунок 11– Таблицы секций приложения

Рассмотрим первую секцию. Следующие за её именем двойные слова представляют собой её виртуальный размер, относительный виртуальный адрес, физический адрес, внутрифайловое смещение и характеристики (как указано в структуре).

Выписываем необходимые данные:

- Имя секции: .text
- RVA начала: 0x00001000
- Внутрифайловое смещение начала: 0x00000200
- Виртуальный размер секции: 5Ch = 92 байт
- Физический размер секции: 200h = 512 байт

Аналогично записываем в отчет данные о второй секции:

- Имя секции: .rdata
- RVA начала: 0x00002000
- Внутрифайловое смещение начала: 0x00000400
- Виртуальный размер секции: DCh = 220 байт

- Физический размер секции: 200h = 512 байт

Встроенный в HIEW анализатор секций позволяет проверить правильность найденных данных. Для этого необходимо нажать F8 (Header) и F6 (ObjTbl) (рис. 12)

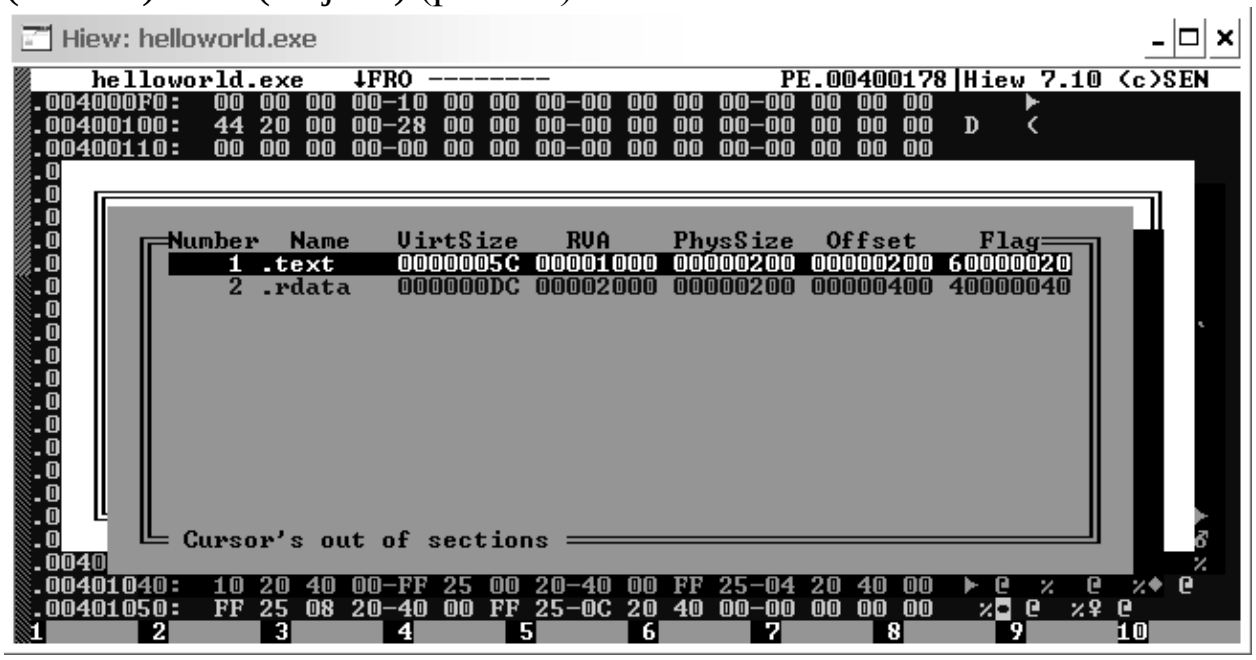

Рисунок 12– Результат работы встроенного в HIEW анализатора секций

3.3 Таблица импорта функций.

Импортируемая функция – это функция, которая находится не в модуле вызывающего приложения, но им вызывается. Функции импорта физически находятся в одной или более DLL. В модуле вызывающего находится только информация о функциях. Эта информация включает имена функций и имена DLL, в которых они находятся. На неё указывает член структуры опционального заголовка DataDirectory (см. описание опционального заголовка). Эти структуры содержат не только символы экспорта, но также символы импорта, ресурсы, исключения, безопасность и т.д. Каждый элемент массива директорий данных – структура IMAGE_DATA_DIRECTORY:

```
IMAGE_DATA_DIRECTORY STRUCT
     VirtualAddress dd ?
     isize dd ?
```

```
IMAGE_DATA_DIRECTORY ENDS
```

Это RVA начала структуры и её размер в байтах. Анализировать директории данных полностью нет необходимости. В нашем примере нам нужны только имена импортируемых функций. Их можно найти во второй секции (рис. 13)

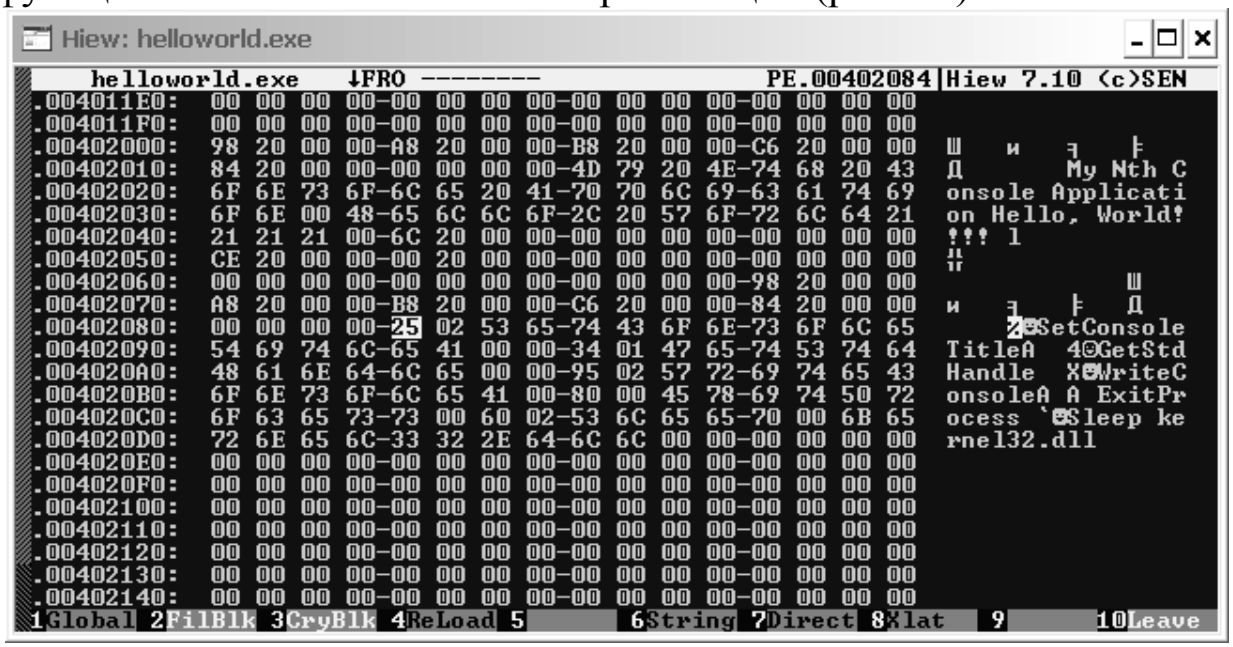

Рисунок 13– Название импортируемых функций в коде программы

Каждая импортируемая функция содержит слово-индекс для быстрого нахождения функции в IAT загрузчиком и её имя. Наше приложение использует 5 импортируемых функций: SetConsoleTitleA, GetStdHandle, WriteConsoleA, ExitProcess, Sleep. Все они импортируются из библиотеки kernel32.dll. Список импортируемых библиотек можно увидеть, нажав F8 (Header) и F7 (Import) в HIEW (рис. 14)

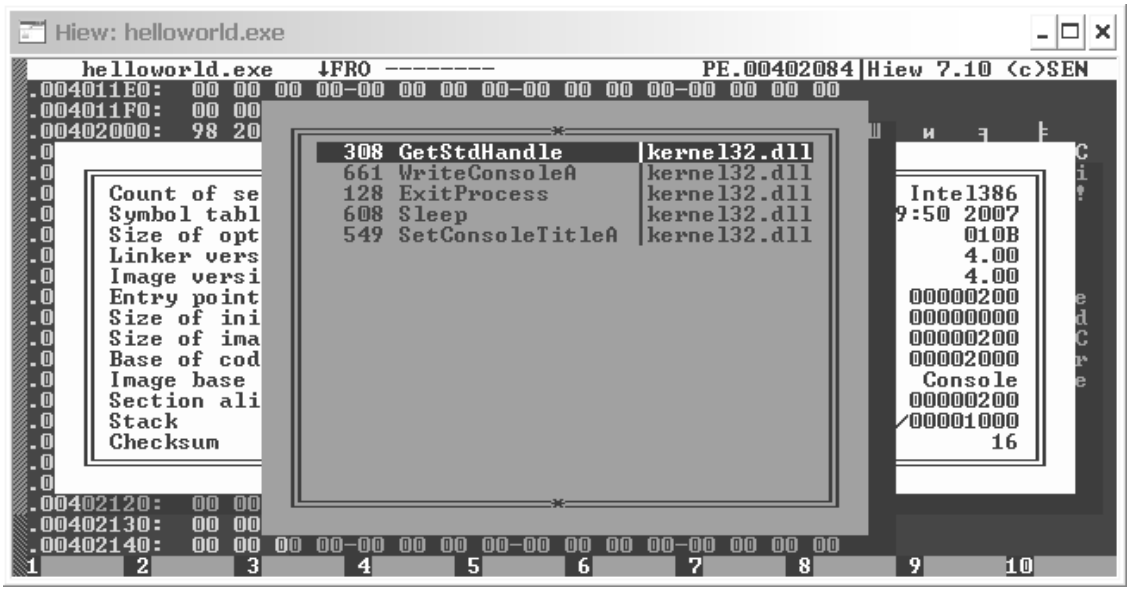

Рисунок 14– Список импортируемых функций.

4 ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Проанализировать любой исполняемый файл в редакторе двоичных файлов HIEW, найти следующие его характеристики:
   а) целевая машина;
   б) число секций;
   в) целевая подсистема (Subsystem);
   г) виртуальный адрес загрузки образа (Image Base);
   д) виртуальный адрес и абсолютное внутрифайловое смещение точки входа;
   е) первую инструкцию, исполняемую загруженным приложением.
2. Для каждой секции найти её имя, относительный виртуальный адрес и внутрифайловое смещение начала секции, её виртуальный и физический размеры.
3. Найти в файле структуру таблицы адресов импорта. Выписать все API функции, импортируемые приложением, а также соответствующие им библиотеки.

5 СОДЕРЖАНИЕ ОТЧЁТА.

1) титульный лист;
2) скриншот программы своего варианта;
3) скриншоты основных этапов выполнения работы;
4) результаты выполнения работы.

6 ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ.

1) Опишите структуру PE-файла.
2) Что такое DOS-заглушка?
3) Опишите формат заголовка PE-файла.
4) Что такое относительный виртуальный адрес?
5) Что такое точка входа? Почему при дизассемблировании необходимо правильно указывать её адрес?
6) Как определить количество секций программы и характеристики каждой из них?
7) Что такое импортируемая функция? Где они расположены?

7 БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1) Гордеев, А.В., Молчанов, А.Ю. Системное программное обеспечение – Спб.: Питер, 2001. – 736с. ил.
2) Рихтер, Дж. Windows для профессионалов: создание эффективных WIN32 приложений с учетом специфики 64-х разрядной версии Windows. – СПб.: Издательский дом «Питер», 2001.
3) Брой, М. Информатика. Основополагающее введение. Структуры систем и системное программирование.– М.: Диалог-МИФИ, 1996
4) Ганеев, Р.М., Проектирование интерфейса пользователя средствами Win32 API. – М.: Горячая линия–Телеком, 2001. – 336 с.