Любая компьютерная программа, в конечном счете, должна быть представлена в виде инструкций для процессора (т.н. opcodes – операционных кодов). Причем если это язык такой, как C или Pascal, то исходный код компилируется уже в бинарный код – набор инструкций в двоичном (бинарном) виде – который и хранится в виде файла, а если интерпретируемый, как Python, Bash или Ruby, то программа хранится в текстовом виде и выполняется специальной программой (интерпретатором) по тексту. При этом есть и смешанные варианты – такие технологии, например, как Java или .NET предлагают исходный код на каком-то языке транслировать в некоторой код, не являющийся машинным, и затем выполнять уже этот код специальной программой.

Нас сегодня интересует программы, скомпилированные в бинарные файлы. Чаще всего в этих файлах не остается информация в текстовом виде о названиях функций и переменных, поэтому нам придется работать с абстрактными адресами в памяти.

Для начала немного теории. Упрощая, можно сказать, что программа храниться в файле в виде процессорных инструкций и при запуске записывается в оперативную память, затем на этот набор инструкций передается управление. В процессоре есть особые именованные ячейки памяти, которые используются в процессе выполнения программы как необходимый контекст для выполнения команд. Среди них есть такие как: ip (instruction pointer) – регистр, в котором хранится адрес следующей инструкции, sp (stack pointer) – регистр, в котором хранится адрес вершины стека (про него будет написано ниже), ax, bx, cx, dx и другие – регистры данных, используемые для промежуточных вычислений и так далее. Список регистров и больше информации про них можно найти в нижеуказанном руководстве, либо в других источниках энциклопедического характера.

Ассемблер – язык программирования, в котором команды ассоциированы с процессорными инструкциями, таким образом, одна команда – одна процессорная инструкция. Полный список инструкций для процессоров семейства Intel и другую информацию на эту тему можно посмотреть в книге [“Intel Software Developer Manual”](http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html). При этом программы разделены на функции (процедуры), каждая функция – набор инструкций, при этом эти инструкции тоже должны быть расположены где-то в памяти, поэтому у каждой функции есть адрес начала.

Также существует такое понятие, как стек – структура, которая присутствует у каждой программы, которая умеет осуществлять две команды – push и pop – соответственно положить на вершину стека (туда, куда указывает esp, при этом сам указатель смещается) и взять с вершины стека. При этом все локальные переменные в функции хранятся на стеке, и каждый раз при заходе в функцию из вершины стека вычитается суммарный размер переменных, так как стек растет вниз – когда мы делаем push, из esp вычитается соответственное количество байт. Обращение к этим переменным идет через регистр ebp (base pointer), в котором хранится адрес основания стека для данной функции, то есть значение esp, если бы мы не выделяли память под переменные.

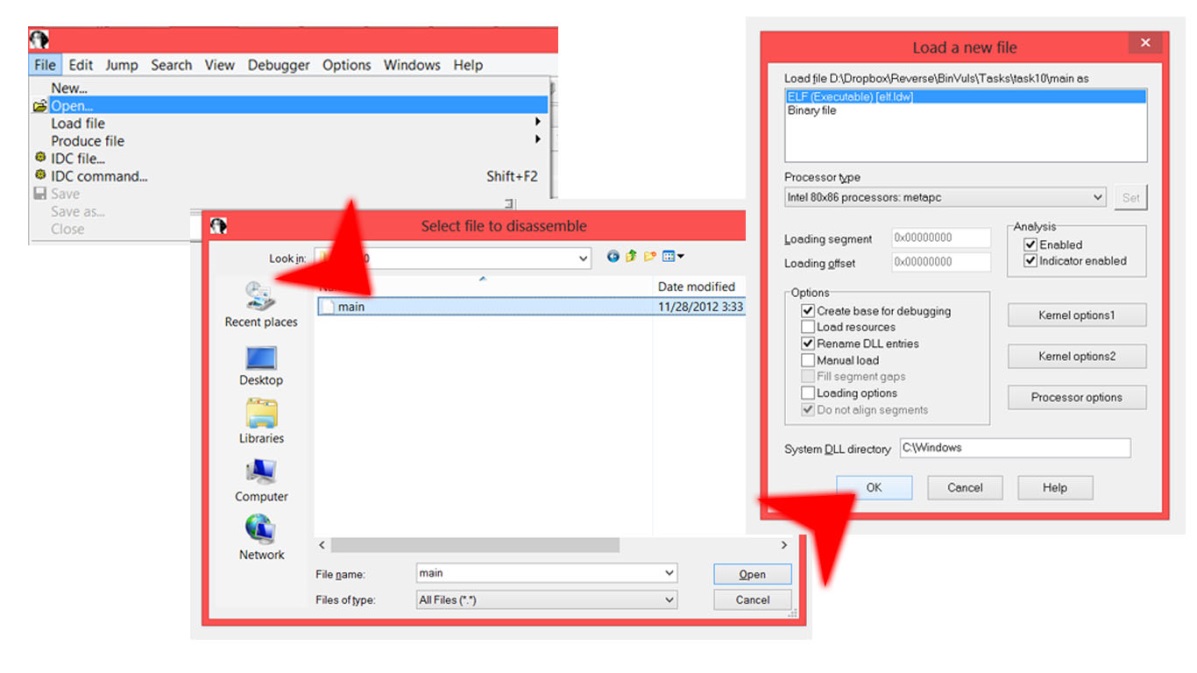
Для reverse engineering’а нам понадобятся следующие инструменты:

1. Дизассемблер для просмотра ассемблерного кода программы, например [IDA (Interactive Disassembler)](https://www.hex-rays.com/products/ida/support/download_freeware.shtml)
2. Hex-editor, например [HxD](http://mh-nexus.de/en/downloads.php?product=HxD)

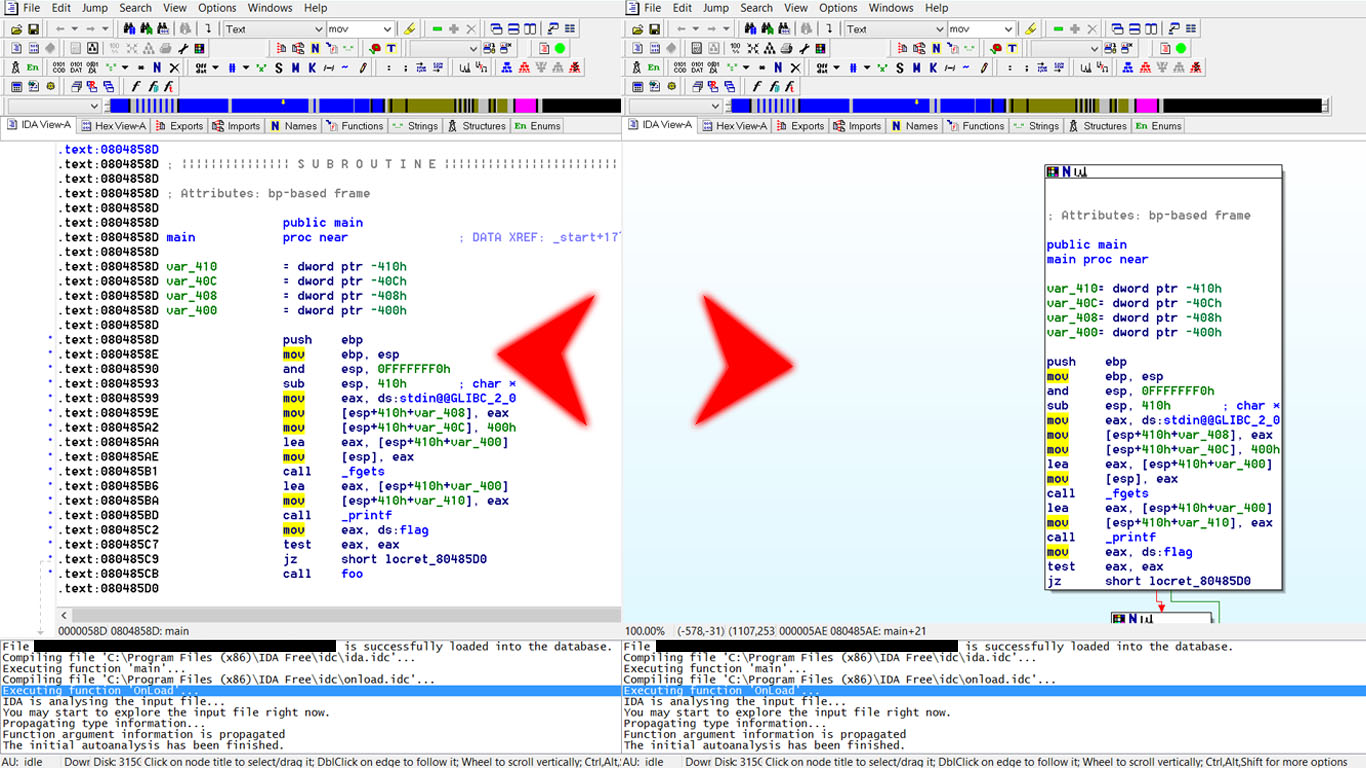
Также в процессе reverse engineering’а могут потребоваться такие вещи как обозреватель ресурсов (например PE Explorer) или дебаггер (например OllyDbg), но сегодня нам они не нужны.

Рассмотрим наши инструменты чуть более подробно

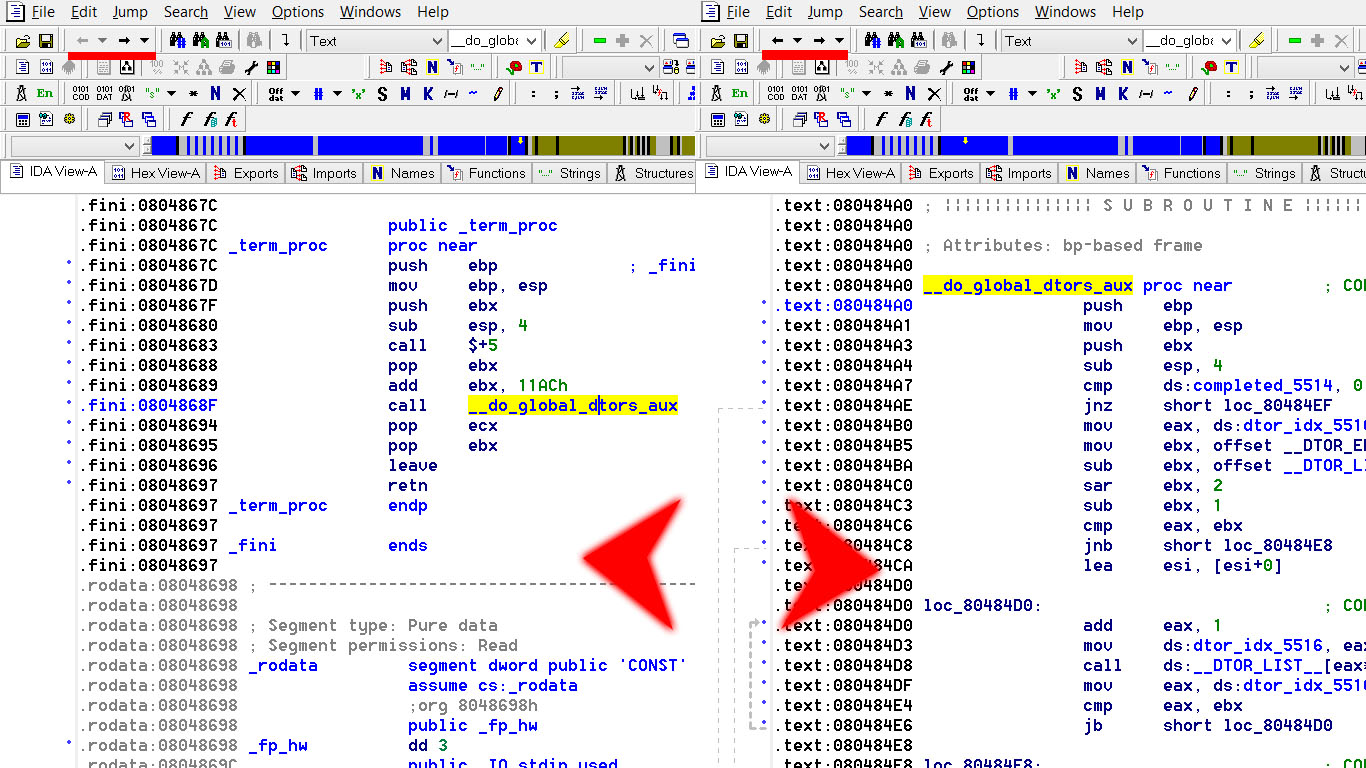
**Interactive Disassembler (IDA)**

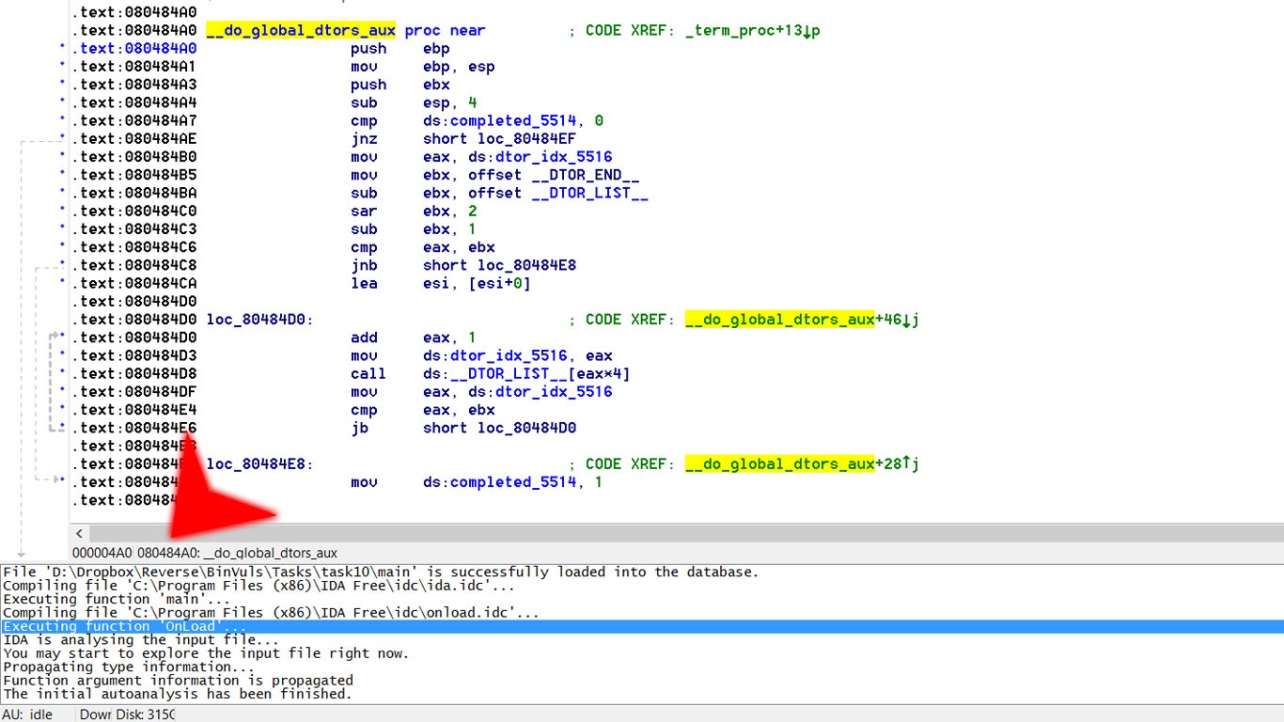
Открыть файл можно с помощью опции в меню File -> Open… (Ctrl+O). При этом в большинстве случаем все остальное определиться автоматически и требуется лишь подтвердить выбор. 

Когда мы открываем исполняемый файл, IDA автоматически генерирует соответствующий код на ассемблере, составляя списки функций, анализирует деревья их вызовов и выдает результат в удобном для нас виде. Для переключения между линейным листингом и представлением в виде графа необходимо нажать пробел.

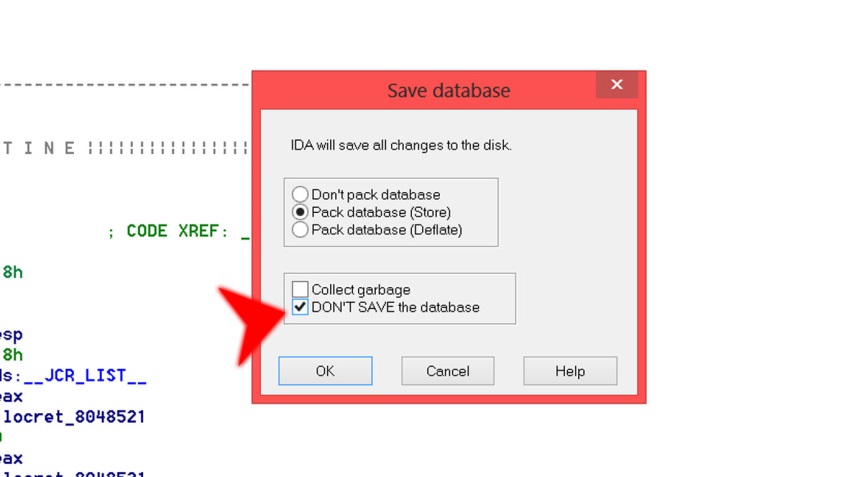


Мы можем перейти на определение функции или участка памяти, просто дважды нажав на них, при этом сохраняется позиция. Перемещаться по сохраненным позициям можно либо с помощью панели инструментов, либо с помощью сочетаний клавиш Esc (назад) и Ctrl-Esc (вперед).



Смещение текущей инструкции относительно начала программы указано в шестнадцатеричном виде в строке состояния. Сначала указано смещение в файле, потом смещение в памяти. 

При закрытии, если мы не хотим сохранять текущую сессию работы в IDA, необходимо указать опцию Don’t Save The Database



**HxD**

Открывать файлы можно с помощью File -> Open… (Ctrl+O) или Drag’n’Drop. Для сохранения файла использовать File -> Save (Ctrl+S). Слева указано смещение относительно начала файла в шестнадцатеричном виде. Изменять можно либо шестнадцатеричное значение, либо символьное представление данных.

