**Лабораторная работа № 3**

**Дизассемблирование и отладка программ**

**Цель работы и содержание:** дизассемблирование и отладка с помощью IDA Pro простейшей программы.

**Продолжительность занятия:** 2 академических часа

# Теоретическая часть

**Отладчик** или **дебаггер** (англ.debugger)является модулем среды разработки или отдельным приложением, предназначенным для поиска ошибок в программе. Отладчик позволяет выполнять пошаговую трассировку, отслеживать, устанавливать или изменять значения переменных в процессе выполнения программы, устанавливать и удалять контрольные точки или условия остановки и т. д.

**Дизассемблер** –транслятор,преобразующий машинный код,объектный файл или библиотечные модули в текст программы на языке ассемблера. По режиму работы с пользователем дизассемблеры делятся на автоматические и интерактивные.

Примером автоматических дизассемблеров может служить Sourcer. Такие дизассемблеры генерируют готовый листинг, который можно затем править в текстовом редакторе. Примером интерактивного дизассемблера является IDA. Он позволяет изменять правила дизассемблирования и является весьма удобным инструментом для исследования программ.

***IDA*** Pro Disassembler (англ.*Interactive DisAssembler*) –интерактивный дизассемблер, который широко используется для реверс-инжиниринга. Он отличается исключительной гибкостью, наличием встроенного командного языка, поддерживает множество форматов исполняемых файлов для большого числа процессоров и операционных систем. Позволяет строить блок-схемы, изменять названия меток, просматривать локальные процедуры в стеке и много другое.

В последних версиях имеет встроенный отладчик, поддерживающий микроархитектуры x86 (IA-32) и ARM.

IDA, до определенной степени, умеет автоматически выполнять анализ кода, используя перекрестные ссылки, знание параметров вызовов функций стандартных библиотек, и другую информацию. Однако вся сила его проявляется в интерактивном взаимодействии с пользователем. В начале исследования дизассемблер выполняет автоматический анализ программы, а затем пользователь с помощью интерактивных средств IDA начинает давать осмысленные имена, комментировать, создавать сложные структуры данных и другим образом добавлять информацию в листинг, генерируемый дизассемблером, пока не станет ясно, что именно и как делает исследуемая программа.

Дизассемблер имеет консольную и графическую версии. Поддерживает большое количество форматов исполняемых файлов. Одной из отличительных особенностей IDA Pro является возможность дизассемблирования байт-кода виртуальных машин Java и.NET. Также поддерживает макросы, плагины и скрипты, а последние версии содержат интегрированный отладчик.

Существует несколько версий IDA Pro – бесплатная (freeware), стандартная (standard) и расширенная (advanced). Бесплатная версия обладает ограниченными возможностями по сравнению со стандартной и расширенной версиями – поддерживается только архитектура x86 и отсутствует поддержка подключаемых модулей.

IDA Pro является самым мощным и самым развитым интерактивным дизассемблером, доступным на сегодняшний день. Основные пользователи этого дизассемблера:

* антивирусные компании;
* специалисты по информационной безопасности;
* эксперты по программному обеспечению;
* разработчики программных защит.

Основная задача IDA Pro – превращение бинарного кода в читаемый текст программы – дополнена многими уникальными возможностями:

* распознавание стандартных библиотечных функций (технология FLIRT);
* интерактивность работы;
* развитая система навигации;
* система типов и параметров функций;
* встроенный язык программирования IDC;
* открытая и модульная архитектура;
* возможность работы практически со всеми популярными процессорами;
* возможность работы практически со всеми популярными форматами файлов;
* работа со структурами данных высокого уровня: массивами, структурами, перечисляемыми типами;
* встроенный отладчик для Win32.

Типичные примеры задач, решаемые с помощью дизассемблера:

* анализ вирусов, троянов и других вредоносных программ;
* поиск ошибок в программах;
* изучение полученного кода;
* валидация программ;
* оптимизация программ;
* разработка защит и поиск дыр в защите.

Остановимся более подробно на технологии FLIRT (англ. Fast Library Identification and Recognition Technology).

Во время дизассемблирования современных программ, написанных на языках высокого уровня, приходится тратить ощутимое время на разбор библиотечных функций. Это время можно считать просто потерянным непродуктивно, так как. при этом мы не получаем новые знания, а всего лишь облегчаем дальнейший анализ программы и алгоритмов, заложенных в нее. Тем более разбор приходится делать для каждой вновь дизассемблируемой программы.

При этом стоит отметить, что разбор алгоритма дизассемблируемоей программы зачастую резко облегчается даже от выяснения вопроса к какому «классу» библиотечных функций относится та или другая подпрограмма. Так, например, работа со потоками в языке C приводит к многократным вызовам различных процедур, ни один из которых не влияет на алгоритм функции…

С другой стороны, каждая программа на языке высокого уровня использует достаточно большое количество стандартных библиотечных функций, процент которых в телепрограммы может достигать до 95%. Из-за этого пользователю дизассемблера приходится тратить больше половины времени на разбор библиотечных функций.

Широкое применение готовых библиотек, таких как VCL, MFC и подобные, еще больше увеличивают долю стандартных функций в программе. Средняя программа, например для Win32, написанная на C++ с применением современных технологий использует 1000 – 2500 библиотечных функций одновременно.

Для повышения производительности труда пользователя дизассемблера IDA был построен алгоритм распознавания стандартных библиотечных функций.

Для улучшения работы алгоритма распознавания библиотечных функций было решено для каждого производителя компиляторов создавать свой сигнатурный файл – это уменьшает вероятность коллизий и удаляет из рассмотрения явно ненужные функции.

Для автоматического определения компилятора, использованного при написании программы и соответственно нужного сигнатурного файла сделаны специальные сигнатурные файлы, применяемые к точке входа в программу. Эти сигнатурные файлы называются startup-сигнатурами. Существуют отдельные startup-сигнатуры для каждого формата дизассемблируемого файла. Для программ под управлением MS DOS используется сигнатура exe.sig, под OS/2 – lx.sig или ne.sig и т.д.

Для уменьшения вероятности ошибки распознания коротких функций для них в обязательном порядке запоминается ссылка на внешнее имя, если она существует. Это в некоторой степени понижает вероятность распознания функции вообще, но это сделано сознательно – лучше не распознать вообще, чем распознать неправильно. Слишком короткие функции (длиной меньше 4 байт), не содержащие ссылок на внешние имена, не участвуют в создании сигнатурного файла и не будут распознаваться.

Коллизии разрешаются человеком – составителем сигнатурного файла. Он выбирает, какие функции надо включать в сигнатурный файл, какие – выбросить. Надо заметить, что этот процесс облегчен до максимума и заключается в расстановке знаков в текстовом файле.

Создание сигнатурного файла происходит в 2 этапа: обработка библиотек и формирование сигнатурного файла. В первом этапе используется программа parselib – она обрабатывает \*.OBJ и \*.LIB файлы. При этом форми-руются pattern-файлы, в которых содержатся шаблоны функций, имена, CRC16 и вся остальная информация, необходимая для построения сигнатурного файла. Hа втором этапе утилита sigmake создает сигнатурный файл из pattern-файлов.

Покажем на примере работу интерактивного дизассемблера IDA Pro для чего выполним дизассемблирование и отладку ассемблированной до исполняемого модуля программы

После инсталляции осуществим запуск дизассемблера IDA Pro и в появившемся диалоге приветствия, показанном на рисунке 8, нажмем кнопку «New», указывая таким образом дизассемблеру, что необходимо дизассемблировать новый исполняемый модуль. Если требуется продолжить анализ или выполнить отладку ранее открытого исполняемого модуля, то его необходимо выбрать в списке диалога приветствия двойным щелчком левой клавиши мыши. Кнопка «Previous» также позволяет открыть последний дизассемблированный исполняемый модуль.

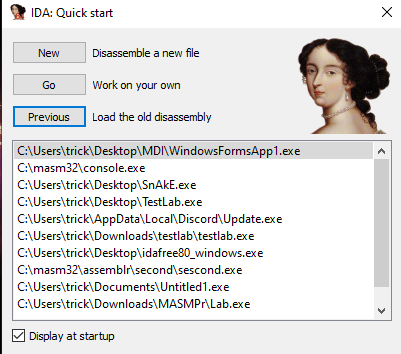


Рисунок 1 – Диалог приветствия интерактивного дизассемблера IDA Pro

После откроется стандартный диалог выбора файла, в котором необходимо выбрать файл с расширением EXE (console.exe). Далее дизассемблер запустит мастер загрузки выбранного исполняемого файла формата PE, в первом окне которого (рисунок 2), он попросит уточнить, нужны ли дополнительные опции анализа машинного кода. Выбираем все галочки контекстного меню и нажимаем далее.

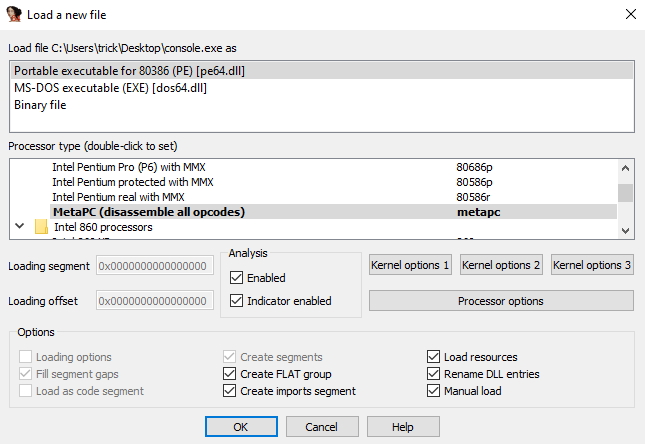


Рисунок 2. Окно подтверждения

Если сравнить исходный код программы, полученный в результате дизассемблирования, с текстом, приведенным в исходном коде преподавателя, можно отметить, что первый значительно больше и менее понятнее по сравнению со вторым. С чем это связано? Это связано в первую очередь с тем, что нами были использованы макросы стандартной библиотеки MASM32, которые в свою очередь, осуществляют вызов процедур этой библиотеки. Процедуры стандартной библиотеки «хранятся» в специальных файлах – файлах статических библиотек с расширением LIB, которые поставляются совместно с ассемблером MASM32.

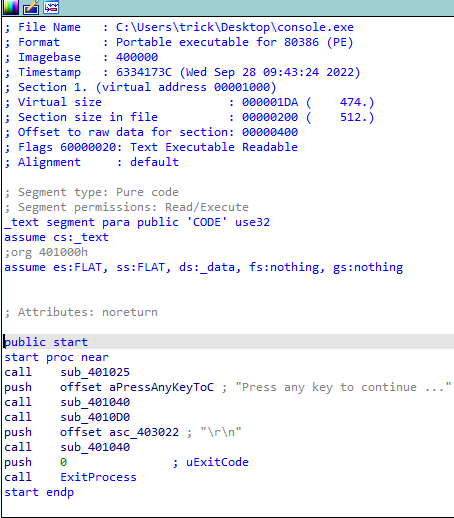


Рисунок 3 – Результаты анализа и дизассемблирования простейшей программы

Как известно, процесс «получения» исполняемого файла с расширением EXE из исходного файла на языке ассемблера с расширением ASM состоит из двух этапов: ассемблирования и редактирования связей. По завершению этапа ассемблирования создаются так называемые объектные файлы, файлы с расширением OBJ, содержащие результаты трансляции программы с языка ассемблера на машинный код. На следующем этапе редактор связей строит из объектных и библиотечных файлов один исполняемый файл. При этом он «добавляет» в исполняемый модуль из файлов статических библиотек машинный код тех процедур, к которым было обращение из всех объектных модулей программы, что обеспечивает автономность исполняемого файла и независимость его от дополнительных библиотек. Поэтому исходный код, полученный в результате работы дизассемблера, содержит в себе не только код процедур из исходного кода, но и код процедур стандартной библиотеки MASM32.

В вершинах графа вызова процедур стоят имена вызываемых процедур, при этом цвет вершины определяет тип вызываемой процедуры. В стандартных настройках IDA Pro для обозначения точки входа на графе используется зеленый цвет, для обозначения процедур программы и стандартной библиотеки используется черный цвет, а для обозначения функций, входящих в состав API Windows, используется пурпурный цвет.

Из анализа графа, можно сделать вывод о том, что точка входа в программу носит имя start. Найдем эту метку в исходном коде программы.

В списке имен, необходимо выбрать имя точки входа start как это показано на рисунке 4 и нажать на клавишу Enter. В результате дизассемблер автоматически перейдет к фрагменту исходного кода, содержащему точку входа.

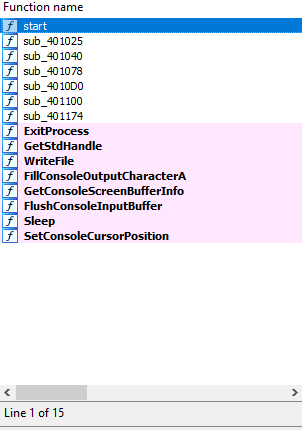


Рисунок 4 – Список имен простейшей программы

Проанализируем первую команду, следующую сразу за меткой, определяющей точку входа, в данном случае это инструкция

call sub\_401025

Первой инструкцией, которая следует сразу же за меткой start, определяющей точку входа, является инструкция call main

Таким образом, дизассемблер распознал процедуру main и присвоил ей новое имя sub\_401025. Поскольку весь основной код программы записан в теле этой процедуры, то нам необходимо осуществить ее поиск. Следует отметить, что для функции main в каждом конкретном случае дизассемблер может назначить свое уникальное имя, отличное от приведенного в этом пособии.

Для упрощения поиска, отобразим список процедур дизассемблированной программы. Для этого перейдем на вкладку Functions главного окна IDA Pro, а если она недоступна, для этого необходимо выбрать пункт меню

Анализируя тело процедуры sub\_401025, можно без труда соотнести его с телом процедуры main. Единственным существенным отличием будет вызов процедур стандартной библиотеки вместо макросов и представление всех целых чисел в шестнадцатеричном виде.

Исследуем предоставляемые функции отладки дизассемблера IDA Pro. Для этого сначала установим точку останова в исходном тексте программы так, как это показано на рисунке 5. Точка останова установлена на первой команде, занимающейся непосредственной обработкой арифметических данных. Чтобы установить точку останова необходимо установить курсор ввода на интересующую команду процессора и выбрать пункт меню Debbuger - Add breakepoin.

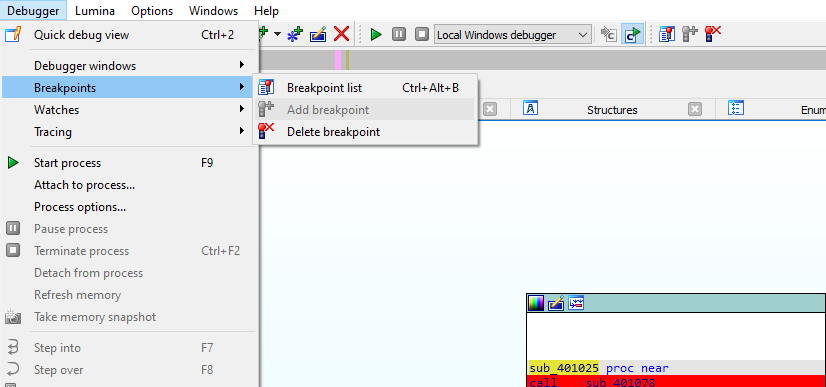


Рисунок 5 – Установка точки останова

После определения точки останова, необходимо запустить отладчик. Для этого необходимо выбрать пункт меню Debbuger - Start process или нажать клавишу F2. Вначале своей работы отладчик выдаст предупреждающее сообщение (рисунок 6) о возможной угрозе в случае отладки вредоносного программного обеспечения. Для продолжения работы отладчика необходимо нажать кнопку «Yes» диалога.

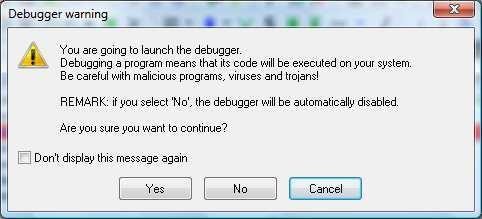


Рисунок 6 – Диалог с предупреждением об опасности со стороны вредоносного программного обеспечения

Отладчик начинает выполнение программы с точки входа и прерывает работу программы перед командой, отмеченной точкой останова. Текущая выполняемая команда выделяется синим цветом в специальном окне, показанном на рисунке 7.

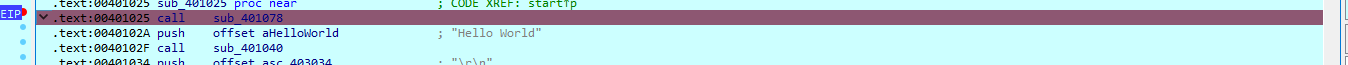


Рисунок 7 – Окно с исполняемыми командами программы

Содержимое программного стека и регистров процессора отображается в окне, показанный на рисунке 8 соответственно.

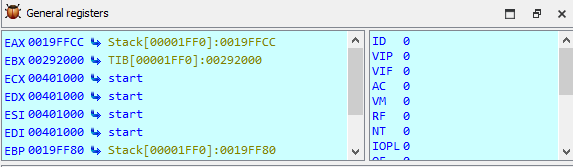


Рисунок 8 – Окно отображающее содержимое регистров общего

назначения, состояния и управления

Продолжить пошаговое выполнение программы можно с помощью команд меню Debbuger - Step into (клавиша F7) или Debbuger - Step over (клавиша F8). В первом случае при вызове процедуры отладчик будет переходить к исполнению команд, составляющих тело процедуры, во втором случае отладчик выполнит процедуру и перейдет к следующей за командой call инструкции. Завершить работу отладчика можно с помощью команды меню

Debbuger - Terminate Process.

# Методика и порядок выполнения работы

Перед выполнением лабораторной работы изучить теоретический материал лабораторной работы. Необходимо придерживаться следующего порядка выполнения лабораторной работы:

Отработать пример дизассемблирования и отладки, приведенный в тексте теоретического материала лабораторной работы.

Выполнить дизассемблирование и отладку каждого исполняемого модуля, полученного в результате ассемблирования. Сравнить исходные (можно получить у преподавателя) и дизассемблированные тексты программ. Отладку каждой из программ производить до тех пор, пока в регистре EAX не будет сформирован окончательный результат. При отладке зафиксировать состояния регистров процессора после выполнения каждой команды. Сравнить значения в регистрах процессора после выполнения каждой арифметической операции с результатами, полученными вручную.

# Вопросы к лабораторной работе

1. В чем отличие между дизассемблером и отладчиком?
2. Каковы основные и дополнительные возможности интерактивного дизассемблера IDA Pro?
3. Каким образом выполняется дизассемблирование исполняемых модулей формата PE в IDA Pro?
4. Для чего используется граф вызова процедур и как он может быть построен в IDA Pro?
5. Как найти точку входа программы в IDA Pro?
6. Как осуществляется отладка программы в IDA Pro?