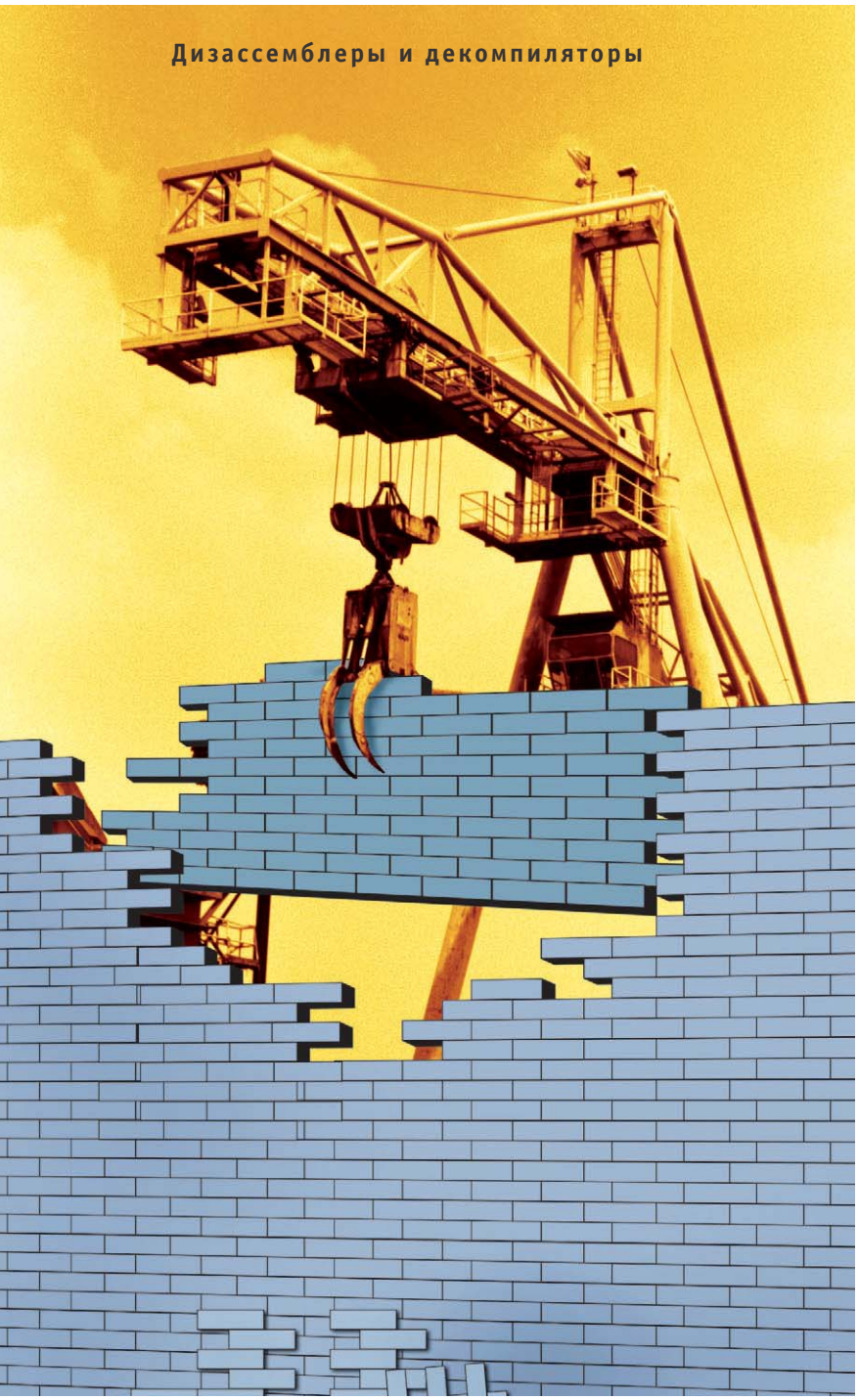
Демонтаж .NET

Платформа Microsoft .NET и спецификация .NET Framework предоставляют большие возможности для изучения структуры и исходного кода созданных в них программ. Поэтому широкое распространение получили специальные утилиты, предназначенные для этой цели.

Дизассемблеры и декомпиляторы



Очередная «инициатива» Microsoft, платформа .NET, сегодня очень популярна среди разработчиков. Одной из ее характеристик является наличие так называемой CLI (Common Language Infrastructure). Исполняемые файлы, предназначенные для работы в CLI (их обычно называют assembly — «сборки»), с одной стороны, являются обычными PE-файлами, с другой, резко отличаются от таковых. Отличие состоит в том, что .NET-файлы содержат в себе дополнительные данные (метаданные).

Метаданные можно представить как совокупность таблиц реляционной базы данных. В полях одних таблиц (назовем их основными) содержится какая-то информация, например анкетные данные студента. Для того чтобы не повторять эти данные многократно, в других таблицах используются ссылки на основные. Этими ссылками могут являться, например, номера строк основных таблиц. Таким же образом устроены и метаданные. Они полностью описывают сборку.

Эти метаданные позволяют, во-первых, определить, какие файлы необходимы, для того чтобы сборка могла работать, и, во-вторых, получить детальную информацию о том, какие типы данных присутствуют в сборке. Все эти сведения при известном навыке весьма несложно получить. Нужно только знать, в какой таблице перечисляются файлы, необходимые для работы сборки, в какой — типы данных, присутствующие в сборке, где можно найти описание полей и методов этих типов. Ну и, естественно, нужно понимать, как связать эти данные друг с другом.

К данным одних таблиц доступ осуществляется по номеру таблицы. При этом размер каждой строки таблицы фиксирован. В некоторых же случаях доступ к данным осуществляется по смещению относительно начала таблиц (например, последовательность строк, используемых программой). Совокупность таких метаданных называют хипом. Обычно в исполняемом файле присутствуют пять хипов данных, каждый из которых предназначен для хранения информации определенного вида и назначения.

Код, содержащийся в сборках, в большинстве случаев является так называемым managed-кодом, то есть после- »

» довательностью операторов на языке MS IL (Microsoft Intermediate Language). Дизассемблировать MS IL намного легче, чем обычный (native) ассемблер. Понять дизассемблированную программу, написанную на MS IL, также намного проще, чем native-код (то есть код на ассемблере процессоров семейства x86). Ну а то, что в состав метаданных включены имена классов, методов, полей и т. д., позволяет рассматривать сборку как модуль с включенной в нее отладочной информацией. Все это дает возможность получить из сборки более понятный (по сравнению с обычными EXE-файлами) дизассемблированный текст, а зачастую и воспроизвести исходный текст программы.

Программисты — люди любопытные, получить исходный код программы — большая радость для них, поэтому совершенно естественно, что сразу же после появления .NET и ее спецификаций были выпущены несколько утилит, предназначенных для просмотра метаданных.

В этой статье я постараюсь рассказать об утилитах, позволяющих осуществлять просмотр метаданных, дизассемблирование сборок и их декомпилирование.

ILDasm

Об этой утилите, входящей в состав Visual Studio .NET производства Microsoft, известно, вероятно, каждому, кто знаком с .NET. Более того, именно с ILDasm обычно начинается знакомство с «внутренностями» .NET.

Эта программа является самой настоящей «рабочей лошадкой» и честно выполняет свои обязанности. Я не знаю ни одного случая «падения» ILDasm, за исключением ситуаций разбора защищенных программ.

Сама программа позволяет осуществлять просмотр дерева типов на экране и сохранять его в файле. Утилита может работать как оконная программа и как утилита командной строки. Для получения полной справки о ее возможностях достаточно просто набрать в командной строке ildasm /?.

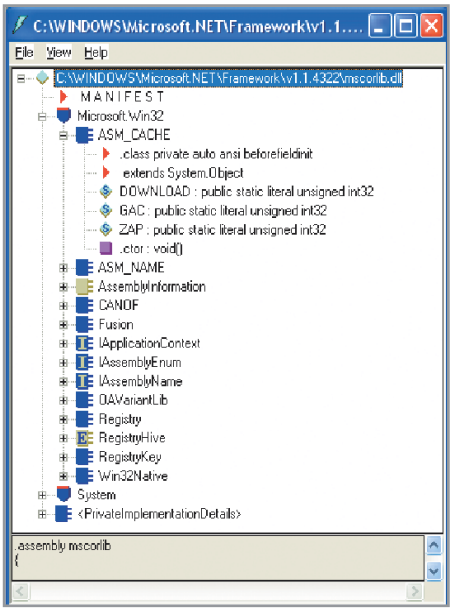
При работе в качестве оконной программы ILDasm позволяет посредством меню определять, какие данные будут отображаться (public, private, member types и т. д.) Что же касается непосредственно дизассемблера, то ILDasm позволяет осуществлять просмотр дизассемблированных методов на экране. При этом для каждого метода создается новое окно — стоит только дважды кликнуть мышкой на элементе дерева, соответствующем методу. В том случае, если пользователю требуется дизассемблировать всю программу, он может сохранить результаты работы ILDasm в файле. Приятной особенностью является то, что в файле может быть сохранено и дерево типов. Честно говоря, у других программ я такой возможности не встречал. Естественно, подобные вещи умеет делать и версия программы, работающая из командной строки. Думаю, каждый может самостоятельно определить, какую версию необходимо использовать в тех или иных случаях.

У ILDasm есть один недостаток. Утилита предназначена только для подготовленного пользователя. Она отображает результат своей работы, но не показывает того, как этот результат был достигнут. Исходные данные, то есть метаданные, о которых я говорил выше, результатом обработки которых и являются дерево типов и дизассемблерный листинг, остаются за пределами видимости пользователя. В том случае, если пользователю необходимо всего-навсего изменить пару байтов в программе, ему придется дизассемблировать всю программу, каким-то образом найти в дизассемблированном листинге необходимый метод, внести изменения, после чего вновь откомпилировать программу при помощи ILDasm. С одной стороны, это неудобно, а с другой — можно вносить любые изменения в программу и не быть ограниченным какими-то рамками.

Asmex

На сайте этой программы о ее возможностях заявляется достаточно громко:

- извлечение ресурсов из сборок;

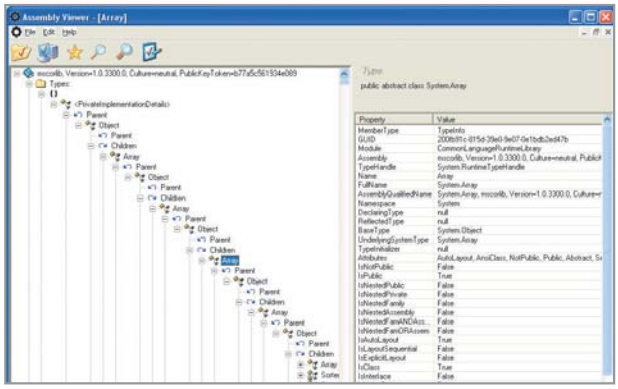


▲ «Дерево» типов, как его формирует ILDasm

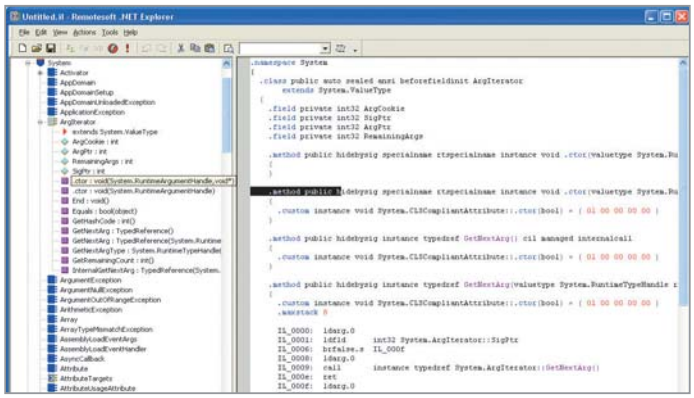
- просмотр таблицы строк метаданных;
- открытие сборки как файла и как элемента глобального кэша сборок;
- просмотр дизассемблированных модулей при помощи ILDasm;
- просмотр структуры PE-файла;
- просмотр типов, пространств имен, параметров методов и т. д.

Давайте рассмотрим ее возможности поподробнее. Ресурсы Asmex действительно умеет сохранять в файле. Но только те ресурсы, которые были добавлены в сборку как файлы (например, файлы картинок, иконок, текстовые файлы). С ресурсами типа .resources, которые были скомпилированы из .resx-файлов (основным типом ресурсов в .NET), Asmex не работает вовсе! Для нее они просто не существуют.

Просмотр метаданных Asmex действительно осуществляет. Однако она показывает только содержимое таблиц, не пытаясь хоть каким-то образом связать их друг с другом. Программа просто-напросто »



◂ Asmex — область его применения весьма ограничена



Внешний вид Remotesoft .NET Explorer

» выдает какой-то набор цифр, находящийся в определенном месте, не более. Другими словами, Asmex работает по известному принципу: «Что вижу, о том и пою». Тем не менее этот недостаток является одновременно и достоинством. Однажды мне пришлось анализировать «обфусцированную» сборку, при анализе которой «упали» абсолютно все упомянутые здесь программы. Researcher .NET показала мне то место, в котором произошло «падение», далее этого места просмотреть файл было невозможно. И только Asmex позволила в более-менее удобочитаемом виде определить, какие изменения были внесены обфускатором.

Итак, я для себя ограничил область применения этой программы просмотром метаданных в тех случаях, когда по каким-то причинам другими средствами это сделать невозможно.

Remotesoft .NET Explorer

Как и ILDasm, эта программа позволяет строить дерево типов и дизассемблировать модуль. При этом на экран выводятся все дизассемблированные методы того класса, на методе которого произведен щелчок мышкой. Искомый метод отмечен черной полосой. По каким-то непонятным причинам сохранить дизассемблированный текст в файле невозможно. Максимум, на что способна программа, — это сохранение в файле дизассемблированного класса. Но этим возможности Remotesoft .NET Explorer не ограничиваются. Она позволяет показывать метаданные и структуру PE-файла.

В частности, при щелчке мышкой по какому-то элементу курсор в окне шестнадцатеричного просмотра устанавливается в позицию, соответствующую началу этого элемента. К сожалению, разработчики не довели эту идею до конца, в результате можно получить смещение только тех элементов, которые включены в дерево, находящееся в левой части. Смещения отдельных строк таблиц метаданных придется рассчитывать вручную. И здесь есть еще одна проблема. Я не понял, чем руководствовались программисты-разработчики, но зачастую величины, занимающие два байта (естественно, речь о 32-битовых процессорах x86), в таблице просмотра метаданных (правая верхняя часть окна просмотра метаданных) отображаются как четырехбайтовые. Но все бы было ничего, если бы не одно но. Практически всегда в нестандартных ситуациях Remotesoft .NET Explorer просто-напросто отказывается что-то выполнять. Например, при анализе kernel32.dll она распознала, что в файле есть раздел Bound Import, но его смещения не показала. При анализе файла dw.exe, который поставляется вместе с Visual Studio .NET, она вообще неверно определила длину файла и не показала, в каком месте располагается раздел безопасности. Справедливости ради оба этих случая нельзя назвать стандартными (например, в kernel32.dll раздел Bound Import расположен после таблицы секций, но до начала первой секции), но эти ошибки разработчики могли бы и устранить.

8 июня 2003 года вышла новая версия Remotesoft .NET Explorer. В ней разработчики постарались добавить такую возможность, как просмотр managed-ресурсов. В том случае, когда я попытался просмотреть ресурс, который был добавлен как файл (не как .resources), программа просто завершила работу. Просто исчезла с экрана, не сказав ни слова. Так сказать, ушла по-английски, не прощаясь.

Кстати, если посетить сайт этой программы, то станет понятно, что разработчики делают ставку не на возможности Remo- »

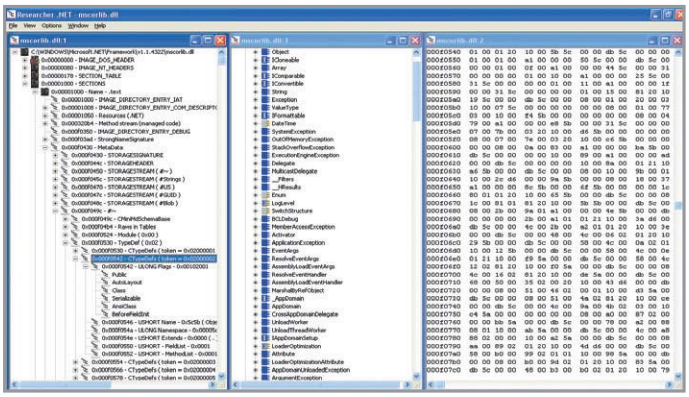
Важный момент

Что такое обфускация?

При работе с исполняемыми файлами, предназначенными для среды .NET, нельзя забывать о том, что эти файлы практически беззащитны. Их не нужно даже взламывать. При необходимости их можно представить практически в исходном виде, внести изменения и перекомпилировать. Ведь даже сама корпорация Microsoft предоставила для этого утилиту ILDasm. Однако далеко не всем авторам программ нравится то, что их код становится общедоступным. Поэтому зачастую используются программы, предназначенные для того, чтобы сделать код менее читаемым, запутать его. Эти программы называются обфускаторами. Они могут изменять имена (например, с Array на 2Fvjs67G), вносить изменения в метаданные, для того чтобы «падали» анализирующие программы, и т. д. Сборки, обработанные обфускаторами, называются обфусцированными. На деле обфусцированный код выглядит так, что совершенно невозможно разобраться в его структуре и зацепится глазом за что-либо. Перед вами практически ничего не значащий набор цифр и букв.

Итак, как же работают обфускаторы? Они анализируют таблицы метаданных и вносят изменения в них, основываясь на определенном алгоритме. А ведь что такое таблица метаданных? Обыкновенная реляционная БД. И зашифровать ее, сделав неудобочитаемой, не представляется сложным делом. После такого шифрования производится запись данных обратно в сборку или генерируется новая сборка (кстати говоря, многие обфускаторы еще и оптимизируют при этом код).

Интересно, что обфускаторы с успехом применяются не только в .NET, но и в других средах и языках разработки, к примеру Java и Perl.



Внешний вид утилиты Researcher .NET

» tesoft .NET Explorer, а на интегрируемые с ней декомпилятор и обфускатор. Remotesoft .NET Explorer рассматривается разработчиками только как среда, которая должна осуществлять доступ к возможностям обфускатора и декомпилятора.

Researcher .NET

Судя по всему, программа напрямую не предназначена для просмотра метаданных, ее целью является построение «карты» файла, то есть определение того, какая информация в каком месте файла находится. Тем не менее к метаданным она подходит детально, разбирая каждый флаг.

Программа показывает всю структуру исполняемых файлов. Для каждого из элементов отображается его смещение в исполняемом файле, тип, название, а также и его значение. При этом программа старается в максимальной степени сделать так, чтобы пользователю не пришлось прыгать по таблицам в поисках информации. Почти каждый элемент дерева сопровождается максимально доступной информацией о нем. В любой момент можно посмотреть содержимое хипа строк и блоков, а также структуру ресурсов (к сожалению, визуальный просмотр ресурсов не реализован).

Кроме того, в программе возможен шестнадцатеричный просмотр файла и дерева типов файла (в стиле ILDasm). При этом необходимо отметить один интересный момент. Все три окна (физическая структура файла, дерево типов и шестнадцатеричный вид) синхронизированы, то есть при изменении выделенного элемента в одном из деревьев меняется выделение и в другом дереве. Таким образом, логическая и физическая структуры данных оказываются связанными. Кроме того, смена элемента в любом из деревьев приводит к смене текущей позиции и в окне шестнадцатеричного просмотра. При этом в отличие, скажем, от Remotesoft .NET Explorer информация в окнах синхронизируется при нахождении курсора в любой позиции, а не только в строго фиксированных местах типа начала таблицы метаданных.

Отличием Researcher .NET от всех остальных программ является и то, что она показывает расположение в исполняемом файле не только непосредственно метаданных, но и исполняемого кода.

Ни в одной другой программе, включая ILDasm, я не нашел такой возможности. Да, необходимо отметить еще и то, что во время тестирования программа не «упала» ни разу. Более того, те нестандартные случаи, с которыми не справилась Remotesoft .NET Explorer, Researcher .NET распознала без труда.

При попытке просмотра «обфусцированного» файла Researcher .NET не смогла до конца разобрать его, но, что интересно, завершила разбор именно на том месте, которое и привело к «падению». После этого стоило только открыть «обфусцированный» файл с помощью Asmex и посмотреть, в каком месте необходимо произвести исправления. А остальное, как говорится, дело техники!

Reflector

Во-первых, эта программа отображает тип выделенного в настоящий момент в дереве элемента. Кроме того, она умеет не только декомпилировать, но и дизассемблировать методы. Но есть одна интересная деталь. Дело в том, что правое окно (в нем отображаются результаты декомпиляции или дизассемблирования) сделано на основе Internet Explorer. Типы, которые представлены в левом окне, появляются в правом как ссылки. Кликнув по ссылке, можно в левом окне перейти на описание того типа, по которому произведен щелчок мышкой. Еще одна приятная особенность — возможность возврата после перехода по ссылке на то место, в котором курсор находился до перехода. Ни одна другая программа этого, к сожалению, не позволяет.

Reflector может также просматривать ресурсы. В том случае, если ресурсы могут быть визуализированы, они показываются в виде картинок. Если визуализировать ресурсы нельзя, они показываются как шестнадцатеричный дамп.

Программы

Asmex 1.0

Разработчик ▸ Jbrowse

Сайт разработчика ▸ www.jbrowse.com

Условия распространения ▸ freeware

Remotesoft .NET Explorer 1.0

Разработчик ▸ Remotesoft

Сайт разработчика ▸ www.remotesoft.com

Условия распространения ▸ freeware

Researcher.NET

Разработчик ▸ Павел Румянцев

Адрес разработчика ▸ pavvel@yahoo.com

Условия распространения ▸ demo

Reflector 3.2.5

Разработчик ▸ Lutz Roeder

Сайт разработчика ▸ www.aisto.com/roeder/dotnet

Условия распространения ▸ freeware

Итоги

Итак, какие же выводы можно сделать? Researcher .NET я, например, использую в тех случаях, когда мне необходимо разобраться со структурой метаданных. Для проверки полученных ей данных пользуюсь Remotesoft .NET Explorer. Если же мне необходимо понять, что делает обфускатор, я использую Researcher .NET в паре с Asmex. В тех случаях, когда мне необходимо получить дизассемблерный вывод, я использую ILDasm. Думаю, грамотный читатель сам придет к выводу о полезности для себя той или иной программы.

■ ■ ■ **Павел Москвин**