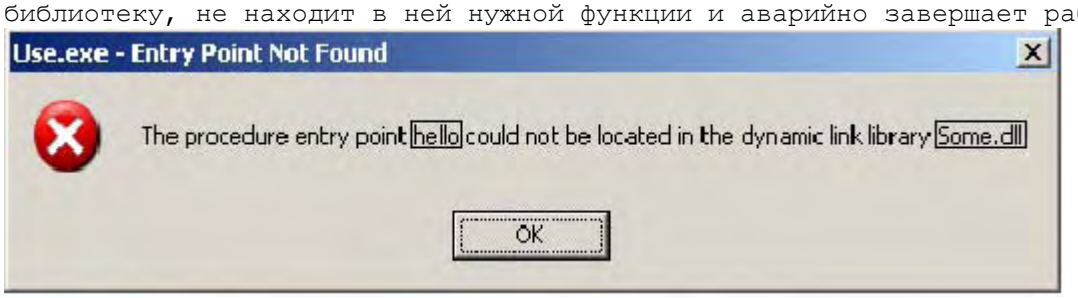
Эссе по курсу "Защита информации", кафедра радиотехники,
Московский физико-технический институт (ГУ МФТИ),
http://www.re.mipt.ru/infsec

Система электронной подписи DLL сборок в .NET
Strong Name Scenario in .NET DLL Assemblies

Выполнил студент 212 группы
ФРТК МФТИ
Быстров Григорий Леонидович.

27.04.2006г

С появления первых языков программирования перед программистами встала проблема разделения и повторного использования кода. Для решения этих проблем были придуманы такие подходы, как процедурное и модульное программирование. Основным недостатком было отсутствие единого стандарта модуля. Например, затруднительно было использовать pascal-модуль graph.tpu в программе на c++ и наоборот. Следующим этапом стало создание динамически подключаемых библиотек (dynamic link lybrary, dll). Dll-библиотеки имеют несомненные преимущества по сравнению со всеми предыдущими стандартами, так как могут подключаться не только статически на этапе компилирования приложения, но и динамически при его выполнении. Так же они позволяют использовать код и ресурсы, имеющиеся в библиотеке, в любом языке программирования. Кроме того, dll-библиотека загружается в память один раз и затем используется различными приложениями без повторной загрузки, что снижает потребности в памяти и увеличивает скорость работы всей системы. Среди недостатков dll-библиотек отсутствие единого стандарта вызова функций из библиотеки (всего их семь: stdcall, cdecl, fastcall, thiscall, PASCAL, FORTRAN, SYSCALL, причем при вызове функции нигде не проверяется формат передачи параметров в функцию, что приводит к ошибкам), отсутствие поддержки классов и самый главный - отсутствие информации о версии библиотеки в самой библиотеке. Это приводит к тому, что программа подключает библиотеку только по имени и остается только надеяться на то, что подгрузится именно нужная библиотека а не какая-нибудь другая с таким же именем. А поскольку Microsoft рекомендует хранить все библиотеки в папке system (или system32), а папка общедоступная, то нередко случается ситуация, когда одна динамическая библиотека заменяется другой с таким же именем (обычно это происходит при установке нового программного обеспечения). Либо будет подгружена "правильная" библиотека, но другой версии. Как правило, это приводит к тому, что программа, подключив библиотеку, не находит в ней нужной функции и аварийно завершает работу:



Эта проблема называется адом динамических библиотек (Dll Hell). Осознав серьезность ситуации, Microsoft порекомендовала хранить библиотеки в папке программы, а не в общем хранилище, а так же ввела поддержку версий dll-библиотек, введя Versioning API для управления версиями, функции этого API следующие:

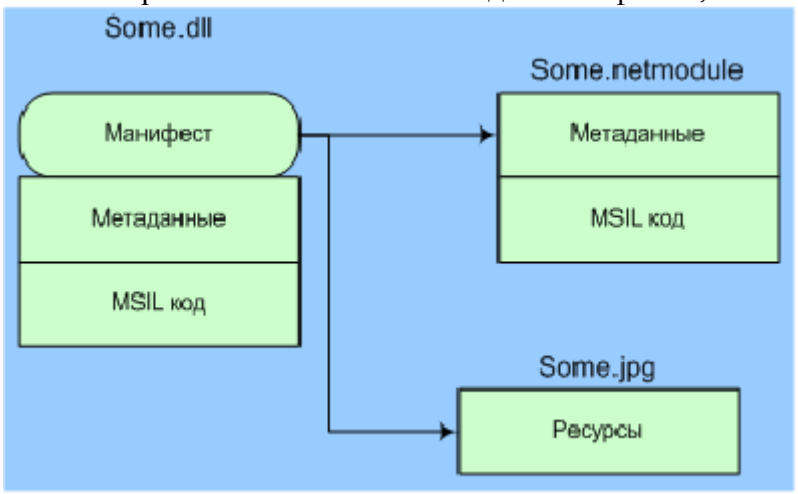
GetFileVersionInfo
GetFileVersionInfoSize
VerFindFile
VerInstallFile
VerLanguageName
VerQueryValue

Конечно, программист мог и сам организовать контроль версий своих библиотек (введя специальные функции в библиотеки), но это не спасало от несанкционированной замены библиотеки, к тому же большинство программистов не подозревали о проблеме ада динамических библиотек. Кстати, защищая свои собственные библиотеки, Microsoft ввела сервисы защиты системных файлов System File Protection и Windows File Protection, не позволяющие, например, совершить подмену dll-библиотек ядра Windows.

При создании платформы .NET Microsoft попыталась разом решить все накопившиеся проблемы, введя технологию сборок (assemblies). Сборка – это наименьший строительный блок приложения .NET, предназначенный для хранения кода приложений и системных библиотек .NET. Сборки разграничивают пространства имен (namespace), типы с одинаковыми именами в разных сборках считаются различными, при указании типа нужно указывать перед его именем имя его namespace или указывать

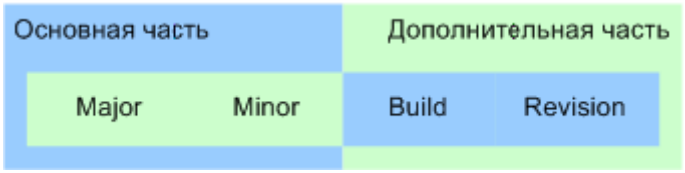
«using <somenamespace>;» в начале кода программы. Каждая сборка имеет версию, считается, что все сущности внутри сборки имеют ту же версию, что и сама сборка. Сборки лежат в глобальном хранилище сборок (GAC, Global Assembly Cache), которое отвечает за совместное использование сборок и политику версий. Две одинаковых сборки с разными версиями могут использоваться совместно (чтобы появление сборки более поздней версии не нарушило работоспособность программ, использующих текущие версии сборок).
Сборка состоит из манифеста, метаданных (которые описывают типы, входящие в сборку), MSIL кода (Microsoft intermediate language, промежуточный язык, в который компилируется код со всех языков .NET) и произвольных ресурсов:

Блоки сборки можно выносить в отдельные файлы, ссылки на них лежат в манифесте:



Манифест - единственный блок, являющийся обязательным. В нем хранится информация о сборке (имя, версия, контрольная сумма и открытый криптографический ключ), список зависимостей (список сборок, необходимых для работы данной сборки), а так же список экспортируемых типов (классов), их связь с внутренней реализацией.
Сборки бывают двух видов – со строгими именами (strongly named assemblies), и с нестрогими именами (в книге [3] названы weakly named assemblies). Сборки со строгими именами отличаются от сборок с нестрогими именами тем, что они подписаны при помощи пары ключей, идентифицирующих издателя сборки. Строгое имя это уникальный идентификатор сборки, поэтому компании, подписывающие свои сборки строгими именами, могут не опасаться конфликтов, вызванных совпадением имен сборок – при подключении сборки проверяется не только имя файла сборки, но так же ее строгое имя.
Строгое имя, записанное в манифесте сборки, состоит из четырех атрибутов:

- Имя (Name), совпадающее с именем файла сборки
- Версия (Version), состоящая из четырех цифр, разбитых на две группы:



Основная (Major) часть, второстепенная (Minor) часть, номер построения (Build) и номер ревизии (Revision). Причем при сравнении строгих имен двух сборок сравниваются только первые две цифры версии (основная часть), а остальные две (дополнительная часть) игнорируются. Дополнительная часть используются для нахождения последнего построения (build) текущей версии сборки при ее подключении к приложению.

- Идентификатор регионального стандарта (Culture)
- Маркер открытого ключа (Public Key Token)

При строгой подписи сборки используется пара из открытого ключа (Open Key) и закрытого ключа (private key). При описании списка зависимостей описываются необходимые для работы сборки и их открытые ключи, но размер открытого ключа велик (несколько килобайт), что сильно увеличивает размер манифеста сборки, поэтому на практике используется маркер открытого ключа (Public Key Token). При его генерировании рассчитывается хеш открытого ключа (по умолчанию для этого используется алгоритм SHA-1), а затем берутся последние 8 байт полученного хеша.

Для строгой подписи сборки сначала нужно получить пару из открытого и закрытого ключа. Для этого применяется утилита Strong Name, SN.exe, поставляемая с .NET framework SDK и с MS Visual Studio 2005. Чтобы сгенерировать пару ключей, нужно выполнить следующую команду:

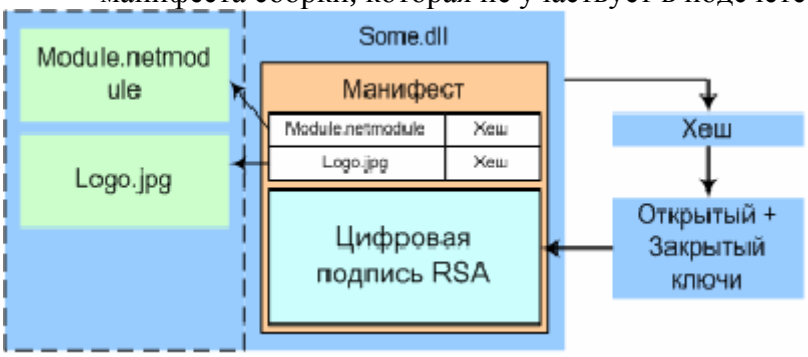
SN –k MyKeys.keys

после чего в файле MyKeys.keys будет лежать пара из закрытого и открытого ключей в двоичном формате. Для того, чтобы никто не мог подписать свою сборку чужим именем, нельзя допустить, чтобы закрытый ключ попал в чужие руки. В крупных компаниях типа Microsoft закрытые ключи хранят в специальных сейфах на смарт-картах, доступ к которым имеет ограниченный круг лиц. Закрытый ключ используется непосредственно со смарт-карты с помощью специальных драйверов, без копирования на диск.

Для подписи сборки строгим именем нужно указать компилятору (в настройках или в командной строке) путь к файлу с парой ключей. Обнаружив этот файл, компилятор подписывает сборку строгим именем и записывает открытый ключ в манифест сборки.

Алгоритм подписи следующий:

- Компилятор считает контрольную сумму файла сборки (контрольные суммы всех файлов, входящих в состав сборки, если сборка состоит не из одного файла)
- С помощью алгоритма SHA-1 рассчитывается хеш контрольной суммы
- Хеш контрольной суммы подписывается алгоритмом RSA парой ключей (закрытым и открытым) пользователя и записывается в специальную секцию манифеста сборки, которая не участвует в подсчете контрольной суммы



При установке сборки в глобальное хранилище сборок (GAC, Global Assembly Cache), система считает контрольную сумму, рассчитывает хеш и сравнивает с цифровой

подписью RSA, извлеченной с помощью открытого ключа. Идентичность значений гарантирует, что ни один бит сборки не был изменен, а так же тот факт, что Ваш открытый ключ соответствует закрытому ключу издателя сборки.
Таким образом, для каждой сборки, включенной в глобальное хранилище сборок (а туда можно включать только строго именованные сборки) при добавлении вычисляется маркер открытого ключа, который сравнивается с маркерами, записанными в манифестах сборок, использующих данную сборку, маркер вычисляется ровно один раз. Если же приложение или сборка хотят использовать стороннюю сборку, не включенную в глобальное хранилище сборок, то алгоритм вычисления маркера открытого ключа используемой сторонней сборки вычисляется каждый раз при запуске приложения.

В процессе разработки приложения, состоящего из нескольких сборок, приходится помногу раз перекомпилировать сборку и запускать приложение. При этом не удобно каждый раз запускать механизм строгой подписи компилируемой сборки, т.к. при этом придется каждый раз доставать закрытый ключ, который обычно хранится где-нибудь в безопасном месте, если речь идет о крупной компании (типа Microsoft), придется каждый раз доставать из сейфа смарт-карту с закрытым ключом. Для решения этой проблемы имеется механизм отложенного подписания (delayed signing), или частичного подписания (partial signing). Отложенное подписание позволяет подписывать сборку, используя только открытый ключ, корректно прописывать маркеры открытого ключа в другие сборки, а так же включать разрабатываемую сборку в глобальное хранилище сборок. Поскольку файл не подписан закрытым ключом, то он беззащитен от несанкционированного изменения. Однако беззащитен он только во время разработки, а перед выпуском приложения все сборки подписываются закрытым ключом.

Применение сборок решает большое количество проблем, которые были у обычных dll-библиотек: глобальное хранилище сборок следит за версиями библиотек, две компании могут создать разные библиотеки с одинаковым именем файла, разные приложения могут использовать различные версии одного и того же файла (Мало того, одно и то же приложение может использовать различные версии одной и той же сборки разными своими частями, прозрачно для программиста). Использование строгих имен гарантирует подлинность библиотеки (сборки), а так же гарантию того, что используется нужная версия файла. Поскольку конкретная сборка подключается к программе только в случае необходимости, т.е. если использован ресурс, класс или функция из этой сборки, то это дает преимущество сборок по сравнению с обычными dll-библиотеками. Даже если у нас есть не все необходимые сборки, есть шанс работоспособности приложения, при использовании только части его возможностей. Кроме того, динамическое подключение сборок экономит трафик при запуске приложений из Интернет: загружаются только необходимые сборки. Таким образом, сборки представляют собой удобный механизм безопасного разделения кода в приложениях .NET.

Изображения взяты из статьи «Немного о сборках» ресурса www.rsdn.ru
Ссылки:
[1] www.msdn.microsoft.com
[2] www.rsdn.ru -> Статьи -> .NET
[3] Книга Программирование на платформе Microsoft .NET Framework, издание третье, Джеффри Рихтер.