Автозащита .NET-приложений

Порядок защиты

Для автоматической защиты **.NET-сборок** используются 2 консольные утилиты:

Утилита	Назначение
CodeObfuscator.exe	Символьная обфускация **MSIL-кода** и шифрование строк
CodeProtect.exe	Перенос части **MSIL-кода** исполняемых файлов и динамических библиотек в защищенное хранилище

Утилиты CodeObfuscator.exe и CodeProtect.exe можно применять как совместно, так и ограничиться использованием одной из них. Однако следует всегда соблюдать определенный порядок использования утилит – сначала приложение необходимо обфусцировать, и только потом провести защиту кода. Правильный порядок использования:

1. обфускация и шифрование строк .NET-сборки
2. защита кода .NET-сборки

Принцип защиты

Появление технологии **.Net** усложнило жизнь разработчиков систем защиты от пиратства. Технология хранения метаданных, используемая в .NET-приложениях, применяется не только для упрощения процесса разработки, но и для эффективного реинжиниринга полученных .Net-приложений. С помощью различных утилит легко восстановить исходные тексты приложений на языках высокого уровня, а анализируя исходный код приложения можно не только отключить систему защиты, но использовать чужие тексты кодов.

Для автоматической защиты **.NET-приложений** используется подход, основанный на двухуровневой защите, где каждый уровень выполняет свои задачи в общем процессе:

- символьная обфускация защищаемого **MSIL-кода**. Используется для затруднения анализа исходного кода программы. Но обфускаци не в состоянии полностью защитить код от изучения: код можно декомпилировать, а затем изучить, хотя и с большим трудом.
- перенос части **MSIL-кода** исполняемых файлов и динамических библиотек в защищенное хранилище. Для более качественной защиты .NET используются методы отложенной компиляции **MSIL-инструкций** посредством технологии **Reflection.Emit,** которая позволяет осуществлять компиляцию MSIL-кода в процессе выполнения. Это дает полное сокрытие исходного текста, даже во время выполнения

Такая концепция позволяет существенно повысить уровень защищенности приложения в целом, так как распространенный инструментарий обратного анализа приложений **.NET** (ildasm, reflector.net и т. п.) становится бессильным.

Это обусловлено тем, что большая часть кода приложения будет храниться в защищенном Native-контейнере, который в свою очередь защищен псевдокодом и использует функционал электронного ключа.

Когда происходит вызов защищенного кода, который зашифрован на аппаратном алгоритме, то сначала происходит обращение непосредственно к самому ключу, и только после этого начинается выполнение кода.

Ограничения

1. Автозащита должна выполняться на ключе той же модели, что будет поставляться с защищенной программой.
2. Для успешной установки и работы автозащиты в ключе, к которому привязывается приложение, должен содержаться алгоритм типа **GSII64** или **AES.**
3. Определитель аппаратного алгоритма в ключе, используемом при защите, должен быть идентичен определителю этого же алгоритма в ключе из комплекта поставки защищенного приложения.

- Не поддерживаются сборки со смешанным кодом и мультимодульные сборки.
- Не поддерживаются самораспаковывающиеся архивы **ZIP, RAR** и т. д.
- Не поддерживаются программы-мастера установки приложений, созданные в специализированных средах разработки: **Wise Installer, Install Shield** и других.
- Не гарантируется корректная защита или последующая работа приложения, которое перед защитой было упаковано специальным упаковщиком *EXE-файлов:* **UPX, ASPACK** и др.
- Не гарантируется корректная защита **EXE-**файлов, код которых был предварительно защищен от модификации или анализа.

Автоматическая защита **.NET** предназначена для защиты **.NET-сборок** (*.exe и *.dll) на платформах **x86/x64/Any CPU**. Режимы шифрования строк, обфускация графа потока управления, защита кода реализуются с использованием алгоритмов шифрования, которые вызываются из электронного ключа.

Различные модели электронных ключей Guardant помогут как реализовать режимы лицензирования по количеству запущенных копий в сети (Guardant Net), так и ограничить время работы (Guardant Time) либо приложения в целом, либо его отдельных сборок. С помощью технологии псевдокода сам Native-контейнер и методы доступа к нему надежно защищены от статического и динамического анализа. Он реализует интерфейс для дешифрования данных с помощью электронного ключа Guardant, осуществляет проверку целостности, дешифрование и загрузки VM.

Рекомендации

Пользуйтесь символьным обфускатором и утилитой защиты MSIL-кода. При этом целесообразно использовать следующие опции:

- **Символьная обфускация**. Позволяет производить запутывание кода приложения, что вместе с другими мерами серьезно повышает уровень защищенности приложения. Используйте обфускацию для всех *.exe и *.dll сборок в приложении, за исключением сторонних и подписанных цифровой подписью.
- **Шифрование строк**. Привязывает обфусцируемое приложение к ключу Guardant и шифрует строковые константы в защищаемых сборках. Рекомендуется использовать эту опцию, если в приложении нет логических элементов, сильно зависящих от быстродействия строковых констант.
- При использовании **опции обфускации публичных интерфейсов** происходит более полная обфускация всей сборки. Однако будьте осторожны, использование этой опции возможно лишь при замкнутости системы (все сборки обфусцируются в одном сеансе, из других приложений никакие методы обфусцируемых сборок не используются). В большинстве случаев безопасно проводить обфускацию публичных интерфейсов для exe-сборок (кроме случаев использования технологии Reflection на самой сборке или экспорта типов для других приложений).
- **Используйте файл с исключениями**. Для его генерации можно воспользоваться утилитой ExclusionUtility.exe, входящей в комплект разработчика. В общем, разработчик приложения должен отчетливо понимать какие типы, методы и свойства необходимо вносить в исключения символьного обфускатора. Как правило, это все те языковые элементы, что могут быть использованы извне. Обратить внимание следует на использование технологий *Serialization, Reflection, DataDinding*.

При использовании утилиты автоматической защиты MSIL-кода необходимо помнить:

- Используйте разумно **опцию с указанием процента защищаемых методов**. Чем больше и сложнее защищаемое приложение, тем меньше необходимо указывать процент. В общем случае, процент 100 можно выставлять для самых простых и нетребовательных к скорости выполнения сборок, содержащих 2 – 3 типа по 10 – 15 методов каждый. Для больших проектов % следует понижать до 10.
- Не защищайте **чувствительные к скорости работы** сборки, типы и методы. Помните, что вынос MSIL-кода в шифрованную область и выполнение его на виртуальной машине может в отдельных случаях существенно замедлять его выполнение (особенно при первом вызове). При дальнейших вызовах также присутствуют накладные расходы, и чем меньше размер метода, тем заметнее падает скорость ее выполнения.
- Пользуйтесь **опцией и утилитой для задания исключений**. Исключайте из защиты все небольшие по объему методы, не содержащие логики, имеющую коммерческую ценность.
- При работе совместно с символьным обфускатором необходимо использовать **опцию создания MAP-файла** (подробнее см. Часть 1 настоящей документации).
- Исключайте вся языковые элементы, построенные на **асинхронной передаче данных**. Защита MSIL-кода таких элементов может приводить к непредсказуемым результатам.