Профилирование .NET-приложений

Принцип

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

Ограничения .Net-защиты:

Важно!

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.

Основным механизмом защиты **.NET-приложений** является шифрование тел защищаемых функций и их динамическое расшифрование во время работы приложения. При совместном использовании символьного обфускатора и шифратора строковых констант на электронном ключе защита становится беспрецедентно стойкой.

Однако в ряде случаев за это приходится платить снижением скорости работы:

- Если функция зашифрована, то при ее вызове на расшифрование тратится определенное время,
- Даже если функция уже расшифрована, она, тем не менее, вызывается с использованием защитных механизмов, что может вносить задержку в ее выполнении на доли секунды.

В среднестатистическом случае увеличение времени запуска приложения и понижение производительности его работы на несколько процентов не вызывает особых проблем. Однако если функция в процессе работы приложения вызывается сотни и тысячи раз, то накладные расходы по работе защитных механизмов именно этой функции могут серьезно снижать производительность приложения. В таком случае, необходима технология выбора функций для защиты. Именно для этого существует **.NET-профайлер.**

* * *

Рассмотрим процесс защиты тестового приложения **Paint.NET.**