средства обработки защищаемой информации, и для разработки безопасных алгоритмов передачи данных от карты к ридеру терминала.

СПИСОК ЛИТЕРАТУРЫ:

1. MasterCard PayPass Mag Stripe Acquirer Implementation Requirements. Официальный сайт Mastercard PayPass. Version 1.0. October 2008. URL: http://www.paypass.com/documentation.html#.
2. Security in Near Field Communication (NFC): printed handout of Workshop on RFID / Ernst Haselsteiner, Klemens Breitfub. 06.07.2006. URL: http://events.iaik.tugraz.at/RFIDSec06/Program/papers/002%20-%20Security%20in%20NFC.pdf.
3. MasterCard PayPass Security: largest community for sharing presentations. 2011. URL: http://www.slideshare.net/szalaydaniel/pay-pass-security-fact-sheet-2010-final.

С. Д. Кулик, И. А. Лукьянов

ЗАЩИТА .NET-СБОРОК ОТ ОБРАТНОЙ ИНЖЕНЕРИИ И ИНЪЕКЦИЙ КОДА С ПРИМЕНЕНИЕМ МЕТОДОВ КРИПТОГРАФИИ

При распространении различного рода систем достаточно остро встает проблема их защиты от обратной инженерии. Другими словами, от декомпиляции и дизассемблирования. Для программ, разработанных на платформе .NET или любой другой, использующей промежуточный код, выполнить эти операции несколько проще, чем для программ, компилируемых непосредственно в машинный код. Как правило, промежуточный код представляет собой ассемблер высокого уровня, не зависящий от архитектуры процессора, что в совокупности упрощает его анализ.

В большинстве случаев применяются подходы к защите от обратной инженерии трех типов: обфускация, статическое и динамическое шифрование. Перечисленные подходы обладают как достоинствами, так и недостатками, однако не обеспечивают полной защиты от декомпиляции.

Обфускация заключается в запутывании кода, например через присвоение классам, полям, методам и т. п. бессмысленных случайно сгенерированных имен, добавление логически незначимых команд, разложение математических операций, добавление условных переходов и т. д. Для платформы .NET разработано достаточно большое число обфускаторов, однако ни один из них не дает гарантированной сложности декомпиляции. Злоумышленники применяют программы-дезобфускаторы для преодоления этого типа защиты. Данные программы создаются путем анализа соответствующих обфускаторов и основаны на выполнении обратного преобразования (из обфусцированного кода в исходный).

Подходы, основанные на шифровании, подразумевают шифрацию сборок одним из криптографических алгоритмов и поставку в зашифрованном виде. Недостатком этих подходов является внесение дополнительной сложности в разрабатываемую программу.

При статическом шифровании расшифровка происходит один раз во время инициализации программы, после чего расшифрованный код выполняется. Данный подход достаточно прост в реализации, однако для его преодоления злоумышленнику достаточно снять дамп памяти выполняемого процесса, проанализировав который он может дизассемблировать расшифрованный код. Чтобы предотвратить такой вид вмешательства, применяют различные механизмы защиты памяти, выходящие за рамки рассмотрения данной работы.

Динамическое шифрование не предполагает загрузку расшифрованной сборки в память. Расшифровка частей сборки осуществляется по мере необходимости. Такой подход делает дамп малоинформативным, однако несет в себе достаточно большую сложность реализации и высокие накладные расходы ресурсов, вследствие чего его эффективное применение целесообразно для ограниченного количества критичных к защите от дизассемблирования участков программы. Данный тип защиты также несет в себе дополнительные уязвимости, вследствие чего основанные на нем алгоритмы следует использовать в совокупности с дополнительными средствами защиты, которые сами по себе достаточно сложны для реализации и не будут рассмотрены в рамках данной работы.

Предлагаемый подход основан на подходах с использованием шифрования. Преимущественно в его основе лежит статическое шифрование как компромиссный вариант (незначительные накладные расходы производительности, простота реализации и интеграции, приемлемый уровень защиты). Однако подход не несет в себе ограничений на тип шифрования и может быть легко адаптирован для использования динамического шифрования.

Платформа Mono 2.10.6 (как и ее более ранние версии), распространяемая с открытым исходным кодом, содержит богатые возможности встраивания в сторонние приложения, что позволяет тесно интегрировать ее с разрабатываемой .NET-программой, требующей защиты от декомпиляции. Непосредственно для обеспечения защиты код Mono должен быть подвергнут модификации, позволяющей платформе работать с зашифрованными сборками прозрачно для нее самой . Преимущество данного подхода также состоит в том, что при полном сохранении совместимости с Microsoft .NET (готовые .NET-сборки подвергаются постобработке) виртуальная машина, используемая в Mono, может быть достаточно сильно модифицирована, что создает дополнительную сложность при анализе кода злоумышленником. Данная возможность реализована в рамках предлагаемого подхода в совокупности с рядом методов по защите памяти от дампинга, нацеленных на снижение информативности получаемого дампа. Программное решение, реализующее предлагаемый подход, представляет собой модифицированную платформу Mono, а также программу-шифратор, выполняющую шифрацию .NET-сборок и их дополнительную модификацию, нацеленную на обеспечение контроля целостности в рамках предотвращения инъекций кода.

Выводы

Предложенный подход позволяет повысить эффективность защиты .NET-приложений от инъекций кода и обратной инженерии. Эффективность подхода подтверждена по результатам независимого аудита безопасности.

Тем не менее, как и любой другой подход в сфере информационной безопасности, предложенное решение не обеспечивает 100-процентной защиты, однако сильно усложняет процесс обратной инженерии и может быть рекомендовано для использования в проектах со средней стоимостью разработки.

С. Д. Кулик, Д. А. Никонец, К. И. Ткаченко, И. А. Лукьянов, Н. Е. Гунько

УСТРОЙСТВО ОПРЕДЕЛЕНИЯ РУКОПИСНЫХ ДОКУМЕНТОВ, ПРИНАДЛЕЖАЩИХ ОДНОМУ ИСПОЛНИТЕЛЮ

Практика убедительно показывает, что преступник практически всегда оставляет следы, например свой почерк. Существует средство ввода рукописного документа и перевода его в