Защита программного кода .NET

Разумное применение обфускации

Андриевский Леонид

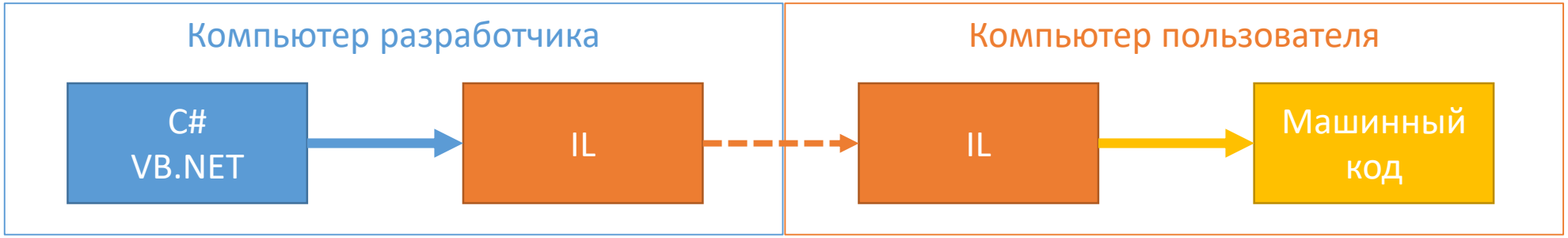
E-mail: me@slenik.net
Telegram: slenik

План

- Введение: лезем «внутрь» не запускающейся игры.
- Зачем защищать программный код?
- Как защищать программный код?
- Обфускация как один из способов защиты.
- Обзор методов обфускации.
- Как реализовать простейший деобфускатор.

Зачем защищать программный код?

Технические причины



- Код в представлении IL содержит
  - имена всех классов, их полей, и методов,
  - названия параметров методов, названия и значения констант.
- Код IL может быть декомпилирован в полуавтоматическом режиме.
- Существуют бесплатные и эффективные утилиты для подобной декомпиляции.
- Существуют способы модификации как файла сборки, так и образа сборки в памяти.

Как защищать программный код?

Юридические методы

...будут рассмотрены не мной и не на этой конференции

Технические методы

1. Просто не давать клиенту защищённую часть ПО.
2. Компилировать ПО не в IL.
3. Внедрение в ПО механизма проверки правомерности владения копией ПО (лицензионного файла/ключа/привязка к учётной записи).
4. Контроль целостности файлов ПО.
5. Обфускация исполняемых файлов.

Обфускация (запутывание)

Это эквивалентное преобразование, которое придает программе форму, затрудняющую понимание алгоритмов и структур данных, реализуемых программой, а также препятствующую извлечению из текста программы определенной секретной информации, содержащейся в ней.

Помогает:

- от изучения программного кода ПО,
- от копирования программного кода или его частей,
- (иногда) от внесения изменений в программный код без ведома авторов.

Обускация: перед началом

- Какого эффекта от обфускации вы хотите добиться?
  - Защита данных.
  - Защита кода (алгоритмов).
- Какую часть кода требуется обфусцировать?
- Как действовать, если код будет деобфусцирован?
  - Как действовать, если обфусцированная часть кода была деобфусцирована?

Методы оббфускации

- Переименование литералов
- Перенос локальных переменных в глобальную область видимости
- Кодирование/шифрование строк
- Добавление мусорного кода
- Разбиение констант
- Формирование непрозрачных предикатов
- Преобразование "диспетчер"
- Переплетение функций
- Виртуализация кода

Переименование литералов

- из `void Method(int dlinnoeMnemoni4eskoeImya = 1)` делаем
  - либо `void a(int b = 1)`,
  - либо `void ·(int · = 1)` (в зависимости от цены обфускатора).
- В .net обфусцировать публичные имена придётся одновременно во всех библиотеках, где они используются.
  - Отдельный момент – явные реализации публичных интерфейсов.
- вернуть читаемые названия популярных обфускаторов (а иногда и исходные) можно с помощью публичных утилит деобфускации.

Переименование литералов

Что это за алгоритм?

```
void A(int[] a, int b, int c)
{
    int d = b, e = c;
    var f = a[(b + c) / 2];

    while (d < e)
    {
        while (a[d] < f) d++;
        while (a[e] > f) e--;
        var g = a[d]; a[d] = a[e]; a[e] = g;
    }

    if (b < e) A(a, b, e);
    if (e < c) A(a, e, c);
}
```

Это **QuickSort**

Кодирование/шифрование строк

- `return "лицензионный ключ введён неверно"` превращается в `return ∞.·(−6403498).`

- метод сам по себе не спасает от целевой атаки:
  - Находим класс, ответственный в библиотеке за ресурсы.
  - Находим и анализируем код, соответствующий инициализации класса ресурсов.
  - Находим все вызовы методов кода, извлекающего ресурсы.
  - Загружаем библиотеку в память; при необходимости вызываем через рефлексию инициализацию ресурсов с необходимыми параметрами.
  - Последовательно вызываем все методы для получения раскодированных строк.
  - Заменяем вызовы методов получения ресурсов на раскодированные строки.

Добавление мусорного кода

- Фиктивные циклы, выполняющиеся 1 раз или не выполняющиеся никогда,
- Вызовы методов, результаты выполнения которых не влияют на выполнение текущего кода.

В теории преобразования подбираются так, чтобы статическому анализатору не хватало данных для принятия решения по упрощению данного участка кода.

Формирование непрозрачных предикатов

До

```
double F(int x)
{
    double r = 1;
    while (x > 1)
    {
        r *= x--;
    }
    return r;
}
```

После

```
double F(int x, int y)
{
    double r = 1;
    do
    {
        y--;
        while (x > 1) r *= x--;
    } while (y * (y * y - 1) % 3 != 0);
    return r;
}
```

Переплетение функций
Преобразование «Диспетчер»

```
private IReadOnlyCollection<Result> DoWork(object param, D mode)
{
    Func<int, bool, IReadOnlyCollection<Result>> someFunc; int a; bool b;
    switch (mode)
    {
        case Mode.One when param is int p:
            a = p; b = false; someFunc = InternalMethod1;
            break;
        case Mode.Two when param is bool w:
            a = 0; b = w; someFunc = InternalMethod2;
            break;
          ...
        default:
            throw new ArgumentOutOfRangeException(nameof(mode), mode, null);
    }
    return someFunc(a, b);
}
```

Виртуализация кода

А давайте в виртуальной машине .NET сделаем ещё одну виртуальную машину?

- Значительное влияние на быстродействие и потребление памяти.
- В программу будет внесено *значительное* количество вспомогательного кода.
- Возможности JIT-оптимизации кода средствами .NET будут сильно урезаны (если они вообще останутся).

Зато изучение такого кода станет на порядок сложнее.

Сценарии, когда обфускация может помешать

1. Заказчик готов приобрести продукт, НО с предварительной экспертизой исходного кода.
2. Заказчик испытывает проблемы при использовании ПО совместно с другим продуктом.
3. Заказчик потерял исходный код или его часть.

Я купил и применил самый крутой обфускатор – теперь я защищён?

Ответ: НЕТ

Пишем простой деобфускатор: переименование литералов

Исходный код: https://github.com/SLenik/simple-deobfuscator

- Метод применяется во всех обфускаторах – в бесплатных ограничиваются только его применением.
- Уже реализован в проектах по деобфускации, например, de4dot

Алгоритм:

1. Считываем исходную библиотеку через Mono.Cecil.
2. Перебираем все объекты (классы, методы, поля, свойства)
   - считаем необфусцированными имена, начинающиеся с буквы и содержащие только буквы, цифры, подчёркивания и точки.
   - Остальное переименовываем.

Итоги

1. Перед применением обфускации следует проанализировать, что требуется защитить и как это можно сделать. И нужна ли для этого именно обфускация.
2. Затем провести хотя бы минимальный контроль качества обфускации.
3. А также проработать различные сценарии действий, на случай преодоления обфускации.
4. Если у вас спрашивают исходный код – подумайте, что дешевле: продать его (можно и подороже) или с некоторой вероятностью получить статью в интернете «50 оттенков запахов кода библиотеки XX».

Список литературы

- Гражданский кодекс Российской Федерации (часть четвертая)
- Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации. Труды ИСП РАН, том 26, вып. 3, 2014.
- Иванников В.П. и др. Реализация запутывающих преобразований в компиляторной инфраструктуре LLVM Труды Института 27 системного программирования РАН Том 26. Выпуск 1. 2014 г. Стр. 327-342.
- Репозиторий de4dot (https://github.com/0xd4d/de4dot)
- Репозиторий Mono.Cecil (https://github.com/jbevain/cecil)
- PreEmptive Solutions: .NET Obfuscator and Integrity Protector (https://www.preemptive.com/products/dotfuscator/overview)
- Eazfuscator.NET – .NET Obfuscator and Optimizer (http://www.gapotchenko.com/eazfuscator.net)
- Spices.Net Obfuscator (http://www.9rays.net/Category/53-spicesnetsuite.aspx)
- ISO/IEC 23271:2012 Standard. ECMA-335. Common Language Infrastructure (CLI) 6th edition (June 2012).

Спасибо за внимание!

Андриевский Леонид

E-mail: me@slenik.net
Telegram: slenik