Введение

Защита кода программы от декомпиляции всегда была, есть и будет актуальной. В первую очередь это относится к коммерческим продуктам, которые, как правило, представляют фирменную тайну для конкурентов. Абсолютной защиты от декомпиляции не существует, но на сегодняшний день у разработчиков программного обеспечения имеется множество программных продуктов, называемые обфускаторами (от английского “obfuscate – ставить в тупик, запутывать”), которые способны усложнить понимание декомпилированного кода. Они имеют отношение к программам, написанным на языках платформы .NET и Java. И это легко объясняется, тем, что и те и другие программы используют промежуточное представление в виде байт-кода, который, несомненно, проще получить, чем код программы, скомпилированной на конкретную платформу, так как и там и там содержится много символьной информации, которая представляет имена членов классов, имена сборок и пакетов, имена самих классов и другую символьную информацию, используемую в константных пулах этих байт-кодов. Сегодня на рынке защиты программного обеспечения преобладают обфускаторы для пакетов написанных на языке Java. Это объясняется тем, что, во-первых, Java старше языков платформы .NET, во-вторых, промежуточный язык компании Microsoft (Microsoft Intermediate Language, MSIL), несколько сложнее байт-кода языка Java, так как он един для всех языков платформы .NET (в одной сборки может находиться классы, написанные на разных языках этой платформы). Но, ситуация может резко измениться, если в будущем .NET станет платформонезависимой, так как в этом случае байт-код .NET будет универсальнее байт-кода Java.

Обфускация – это не наука, а опыт различных компаний, разрабатывающих методы защиты программ. До сих пор не существует строгой теории обфускации, которая, могла бы дать ответ на такой вопрос как: какой из обфускаторов более надежно защищает код, а какой из них это делает хуже? То есть такого понятия как степень обфускации (степень “запутанности”), на сегодняшний день нет. Тем не менее, различные фирмы разрабатывают все более изощренные методы запутывания. Порой, для того чтобы получить боле менее понятный код программы не помогает не один декомпилятор, и в этом случае приходиться анализировать графы потоков управления методов вручную, что надолго приостанавливает получение представления о том, что конкретно делает тот или иной метод какого-либо класса. Вот пример обфусцированного класса:

Исходный класс:

```
public class Example1 {
    private int InCnt;
    private int Current;
    private byte[] InBuff;

    public int readbytes(byte[] b1, int i2) {
        int i3;

        if (InCnt <= 0) {
            return -1;
        }

        if (i2 > InCnt) {
            i2 = InCnt;
        }
        i3 = 0;

        while (i3 < i2) {
            b1[i3] = InBuff[Current++];
            InCnt = InCnt - 1;
            i3++;
```

Обфусцированный класс после декомпиляции:

```
1   public class a {
2       private int a;
3       private int b;
4       private byte[] c;
5
6       public int a(byte[] b1, int i2) {
7           boolean b3;
8           b3 = a;
9           a;
10          {
11              if (if (b3) goto L28 <= 0) {
12                  return -1;
13              }
14              i2;
15          }
16
17          if (if (b3) goto L46 > a) {
18              i2 = a;
19          }
20          0;
21      l4 = L46:
```

```
        }
        return i3;
    }
}
```

```
22
23          if (b3 == false) goto L84;
24      L52:
25          b1[l4] = c[b++];
26          a = a - 1;
27      L81:
28          l4++;
29      L84:
30          if (l4 < i2) goto L52;
31          if (b3) goto L81;
32          return l4;
33      }
34  }
```

Данный класс обфусцирован пакетом *Zelix KlassMaster version 4.2* (http://www.zelix.com) компании *Zelix Pty Ltd*, декомпиляция осуществлена пакетом *JShrink version 2.31* компании *Eastridge Technology* (http://www.e-t.com) (единственный из семи декомпиляторов, который смог хоть что-то сделать с этим классом, остальные попросту выдавали сообщения о неспособности декомпилировать его). Особый интерес представляют строки 9, 11, 14, 17, 21. Согласитесь, что не зная алгоритмов обфускации разобраться в таком коде весьма проблематично. Даже если алгоритмы обфускации известны, то время, которое уйдет на анализ графа потока управления метода этого класса, может превысить время, в течение которого данный код будет нужен. Таким образом, максимум, чего может ожидать пользователь от того или иного обфускатора, это выполнение хотя бы одного из двух следующих критериев защищенности:

- Стоимость деобфускации превышает стоимость исходного кода программы.
- Время, которое требуется на деобфускацию кода, превышает время, в течение которого его значимость остается актуальной.

Оптимизация как разновидность обфускации

Оптимизация байт-кодов Java и платформы .NET, также является своего рода обфускацией, так как, в основном декомпиляторы пишут для кодов программ, откомпилированных стандартными компиляторами. Данные декомпиляторы работают в соответствии с работой конкретного компилятора. Например, javac - стандартный компилятор языка Java – построит байт-код некоторого класса в соответствии с некоторыми правилами – шаблонами, грубо говоря, он рассуждает так – этому блоку надо поставить в соответствие такую последовательность команд, а этому такую и т.д. Данный компилятор при встречи объявления нового экземпляра класса поставит в соответствие строго определенную последовательность команд, если же объявление экземпляра этого класса представить другой, измененной последовательностью команд, которая также будет содержать меньшее их число и являться исполняемой, то обычный декомпилятор, скорее всего, не сможет декомпилировать метод, в котором использовался такой прием. Следовательно, если программа компилируется каким либо неизвестным оптимизирующим компилятором, то в этом случае в получении исходного кода могут помочь только те декомпиляторы, которые выполняют детальный анализ графа потока управления методов программы и знают приемы конкретного оптимизатора.

Также оптимизаторы выполняют удаление неиспользуемых записей из константных пулов байт-кодов классов, что зачастую приводит к невозможности декомпилировать их. Этот факт еще раз подтверждает то, что многие декомпиляторы не проводят “честный” анализ байт-кодов, а работают в соответствии с шаблонами стандартного компилятора.

Обзор методов обфускации

Далее будут описаны, лишь некоторые способы обфускации, так как многие из них представляют коммерческую тайну фирм, занимающихся разработкой средств защиты программ. Конечно же наиболее простой и легко анализируемой схемой обфускации является так называемая name obfuscation – обфускация имен класса, т.е. изменение имен членов классов, имен самих классов и изменение имен Java-пакетов и сборок .NET. Этот вид обфускации может быть осуществлен следующими подходами:

1. Обфускация имен производится с использованием длинных имен. В этом случае расчет делается на то чтобы сбить с толку потенциального взломщика разнообразием имен вроде следующего “kDgftEsdlS78Ccs”. Но этот способ является весьма отчаянным, так как хорошо известен и многие декомпиляторы адоптированы к нему следующим образом: при встречи серии членов, к примеру полей, с такими именами декомпилятор назначит им более понятные имена, например “field_1”, “field_2” и т.д.
2. Изменение имен производится с использованием имен, содержащих “неизвестные” символы, например символы японского или китайского алфавитов. Несмотря на то, что Java и .NET используют кодировку символов UTF8 (подробнее ознакомиться с ним можно на сайте http://www.unicode.org), некоторые декомпиляторы не работают со всеми символами этой кодировки, другие же попросту заменят имена, содержащие подобные символы на более понятные, третьи могут выводить на месте символа соответствующий ему код.
3. Использование имен меняющих смысл того, что делает метод или означает поле или класс. Это означает следующее: пусть есть класс “Car”, в нем есть поле “motor” и метод “move”, после проведения обфускации имен эти сущности могут иметь следующие имена “butterfly”, “petroleum”, “home”.
4. Использование максимально коротких имен. В этом случае, в отличие от предыдущих, уменьшается размер класса, так как мало того, что имена являются короткими, но и одно и тоже имя можно использовать несколько раз (например, одно и тоже имя можно использовать и для имени пакета или сборки и для имени класса и для нескольких имен членов этого класса).
5. Использование зарезервированных слов языка, таких как while, for, do, if и т.д. Но опять же этот способ защитит только от самых “примитивных” декомпиляторов, которые при встречи члена с именем, представляющим зарезервированное слово, откажутся декомпилировать класс или отдельную его часть.

Обфускация имен не значительно защищает программу от декомпилирования, так как в этом случае запутывается только символьная информация, содержащаяся в константных пулах классах, и то не вся, так как помимо имен членов классов в константном пуле содержится и другая символьная информация, представляющая собой строки выводимые программой в процессе работы, имена других пакетов используемых в этом классе (объявленные в директивах “import”, “using”) и прочее. Что касается строк, используемых в классах, то они также могут быть подвержены обфускации. Обфускация строк выполняется их кодированием (string encryption), в этом случае при обращении к закодированной строке во время выполнения программы производится вызов алгоритма декодирования строки, который представлен в виде отдельного метода данного класса, либо при запуске программы сначала выполняется декодирование всех ее закодированных строк (алгоритм декодирования помещается в отдельный класс), и только потом выполняется программа. Кстати обфускаторы, осуществляющие только изменение имен относят к обфускаторам первого поколения. Гораздо более изысканными видами обфускации обладают обфускаторы второго поколения, которые выполняют обфускацию данных (data obfuscation) и обфускацию потока управления (control flow obfuscation). Рассмотрим сначала методы обфуфскации данных, а затем и методы изменения графов потока управления методов,

содержащихся в классах и графов исключений. К наиболее популярным методам обфускации данных относятся следующие:

1. Для определенных полей и методов класса может быть выполнена финализация (finalization), т.е. данные член могут быть защищены от переопределения в классах потомков, созданных от класса, которому принадлежит данный член. Но этот процесс скорее имеет отношение к оптимизации байт-кода, т.к. уменьшает время выполнения программы, тем не менее преобразование данных осуществляется.
2. Перегруппировка данных подразумевает объединение нескольких сущностей в одну или наоборот представление одной сущности несколькими. К примеру, две 32-битные переменные могут быть представлены одной 64-битной, одномерный массив может быть разбит на несколько массивов и представиться многомерным, булевы переменные могут быть представлены целочисленными. Конечно же, после таких преобразований необходимо изменить все обращения к измененной сущности.
3. Непосредственно в байт-код вставляют неиспользуемые данные, например, логические переменные, на основе которых создают ложные условия или не нужные циклы.
4. Меняется способ хранения данных, например, некоторые закрытые члены делаются открытыми и наоборот.

Что касается обфускации графа потока управления, то этот вид обфускации является наиболее устойчивым к попыткам анализирующей стороны деобфусцировать программу. При хорошо проведенной обфускации потока управления может не помочь ни один декомпилятор, а в этом случае для получения исходного кода придется использовать дизассемблер и производить анализ ассемблерного кода, что надолго приостановит получение исходного кода. Другой вариант анализа в этой ситуации состоит в проведении непосредственного анализа графа потока управления, что также уменьшает вероятность скорейшего получения исходного кода. Наиболее популярным приемом проведения этого вида обфускации является вставка в граф управления ложных условий. Например, перед входом в цикл можно вставить ложное условие вроде следующего «if (3 == 2) {} else {}», при этом дугу, которая соответствует истине бросить во внутрь цикла, а дугу, которая соответствует лжи бросить на начало цикла. Но, скорее всего, этот прием не собьет с толку многие декомпиляторы. Можно сделать цикл с двумя и более входами, например перед началом цикла вставить условие, которое всегда истинно, истинную дугу бросить на начало цикла, ложную на базовый блок (basic block), который помещается в конец физической последовательности, представляющий данный граф, а выходную дугу из этого блока на начало цикла. Таким образом можно получить цикл с несколькими входами. Проводя манипуляции с графом исключительных ситуаций также можно добиться неплохих результатов, например если в графе потока управления исключения отсутствуют (метод не выбрасывает никаких исключений), те мне менее можно наделить этот граф некоторыми исключениями.

Другой способ состоит в замене вызова метода самим телом методом (inlining) или оформление части кода в виде отдельного метода (outlining), при этом запутыванию подлежит трасса стека:

Трасса стека при выполнении необфусцированного класса

```
Exception in thread "main" java.lang.NullPointerException
   at com.mycompany.MyClass2.method1(String)(MyClass2.java:9)
   at com.mycompany.MyClass1.method1(String)(MyClass1.java:9)
   at com.mycompany.MyClass0.method1(String)(MyClass0.java:9)
   at com.mycompany.MyClass0.main(String[])(MyClass0.java:25)
```

Трасса стека при выполнении обфусцированного класса

```
Exception in thread "main" java.lang.NullPointerException
      at a.a.c.a(c.java:7)
      at a.a.a.main(a.java:10)
```

В данном примере в результате замены вызова метода телом самого метода цепочка методов, ответственных за исключение стала короче. Если же воспользоваться методом оформления части кода в виде отдельного метода, то эта цепочка наоборот будет длиннее.

Еще одним способом обфускации графа потока управления является перемешивание случайным образом линейных участков. Надо сказать, что проведение обфускации потока

управления, схоже с действиями, которые выполняют оптимизаторы. Например, операция замены вызова метода телом самого метода является неотъемлемой частью оптимизатора так же как и операция финализации членов. Некоторые декомпиляторы с трудом получают исходный текст класса после того как последний подвергся оптимизации.

Заключение

Выше приведены лишь наиболее известные методы обфускации, более изощренные приемы в этой области являются коммерческой тайной занимающихся защитой программного кода фирм. Как правило, эти методы математически строго обосновываются и более надежны.

Литература

1. James Gosling, Bill Joy and Guy Steele. The Java Language Specification. Addison-Wesley, 1996.
2. Tim Lindholm, Frank Yellin. The Java Virtual Machine Specification.
3. John Gough. Compiling for the .NET Common Language Runtime (CLR).
4. Java Decompilers. http://andromeda.com/people/ddyer/java/decompiler-table.html.
5. Cristian Collberg, Clark Thomborson and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Departament of Computer Science, University of Auckland, July 1997. http://www.cs.auckland.ac.nz/~collberg/Research/Publications/CollbergThomborsonLow97a.