УДК 004.056

МЕТОДЫ ЗАЩИТЫ JAVA ПРОГРАММ

Д. С. Замятин; Я. В. Пишта,
Национальный технический университет Украины «КПИ»
pishta.yaroslav@bk.ru

Создание программных продуктов, устойчивых ко взлому, теоретически невозможно, независимо от языков и технологий, которые используются при написании программы. Единственно возможный путь защиты – это усложнить взлом программы настолько, что это станет не- выгодным и будет легче и дешевле купить программу.

В данной статье мы сконцентрируем внимание на защите Java-программ, так как в отличие от C или C++, Java- компилятор не создает конечный машинный код, а всего лишь его платформенно независимое представление. В результате сгенерированный байт-код содержит слишком много осмысленной информации, которая помогает разобраться взломщику программы в принципах её работы. При написании программы вводятся понятные названия для методов, переменных классов, что тоже упрощает взлом.

Наши действия должны быть сконцентрированы на защиту кода от декомпиляции, так как это позволит злоумышленнику разобраться в методах работы программы и изменить ее для своих целей.

Существуют следующие стратегии победы декомпиляторов [5]:

1) активное использование флагов компиляции;
2) написание двух похожих версий программного продукта;
3) затемнение кода;
4) изменение байт-кода;
5) использование JNI;
6) хранение в атрибутах методов;
7) применение глухих классов.

1. Использование флагов декомпиляции

В основном нас интересуют только 3 основных флага компиляции, это –g, –O и без флага. Флаг –g указывает компилятору добавлять номер строки и имя локальных переменных в конечный байт-код. Без флага – будут потеряны имена локальных переменных, но сохраняются номера строк. Флаг –O – будут дополнительно удалены номера строк.

Данный метод может являться лишь первым шагом на пути защиты Java-кода от взлома. Отсюда можно сделать единственный вывод - необходимо всегда компилировать программу только с ключом компиляции –O[6].

2. Написание двух версий программного продукта

Необходимо создать демо-версию программы с ограниченными возможностями и распространять ее бесплатно, а полный функционал – уже платен. Таким образом нет особой нужды защищать программу. В случае покупки программного продукта выдаётся полная версия программы [4].

Данный метод имеет несравненный плюс при демонстрации, при которой вручается полурабочий образец, который сложно будет применять в каких-либо других целях, кроме как ознакомление с программным продуктом. И, конечно же, в нем будет отсутствовать

достаточно важная часть кода, которую невозможно будет получить даже при помощи декомпиляции.

3. “Затемнение” кода

Если проект переходит определённый предел сложности, то разобраться в логике программы можно только с использованием комментариев в коде и технической поддержки. Это и есть основа данного метода. Во всей программе происходит замена декларативных названий полей классов и методов на абстрактные, которые не несут какой-либо смысловой нагрузки.

После применение метода код станет плохо читаемым и поэтому, чтобы разобраться в логике программы, необходимо приложить на порядок больше усилий.

Данный метод очень сложно реализуем, если в программном продукте необходимо предоставлять открытый API, или если в проект вводят EJB (которые тоже должны иметь строго определённые методы открытый API), но можно применить не тотальное переименование классов и методов, а только частичное, но для этого необходимо изначально проектировать систему с учётом дальнейшего её “затемнения”.

4. Изменение байт-кода

Против несложных декомпиляторов можно побороться следующим способом. Вставлять лишние инструкции в байт-код. Например, если после return в методе класса вставить java инструкцию pop, то многие декомпиляторы воспримут это как ошибку и не смогут корректно декомпилировать этот код. Хотя при этом JVM его сможет исполнять без ошибок. Этот метод требует знания структуры файла классов и неэффективен против некоторых декомпиляторов.

Полученный код уже не откомпилируется. Этот метод не универсален, и это, собственно, его единственный недостаток, но он может эффективно применяться наряду с другими методами защиты программных продуктов.

6. Хранение классов в атрибутах

Пусть у нас есть определённая область памяти, где у нас хранится зашифрованный байт-код некоторых классов. Для того чтобы получить экземпляр класса – нам предварительно нужно расшифровать этот класс, а затем при помощи метода, схожего с ClassLoader, создать экземпляр класса.

Например, мне представляется хорошим сценарий вызова native-метода, который возвращает расшифрованный байт-код класса (если лицензия корректна)[5]. А затем мы можем работать с ним, как с обычным классом. Это получается своего рода аналог ClassLoader. Но при этом нужно обязательно проверить подписчика класса, а то слишком умный взломщик и вызывающий класс сможет подменить, тем самым получить недостающий ему байт-код. Так же нужно проверять, не запущена ли JVM с возможностью отладки, иначе существует риск того, что может быть получен класс напрямую из памяти.

В сочетании с использованием JNI это достаточно эффективный метод борьбы с декомпиляцией.

7. Применение глухих классов

Этот метод прежде всего нацелен на то, чтобы препятствовать повторной компиляции программы. Приведу пример, который поможет понять всю прелесть идеи.

Создаем глухой класс SomeClass.java:

```
public class SomeClass {   public SomeClass() {    }
   public boolean getSome() {  return true;  }}
```

Дале в своей программе вставляем следующий код:

```
static boolean veriable1 = false;

if (!veriable1) { SomeClass fc = new SomeClass(); veriable1 = fc.getSome(); }
```

Например:

```
public class A {    static boolean veriable1 = false;
   public static void main(String[] args) {
      veriable1 = true;  System.out.println("Hello, World!\n");
      if (!veriable1) {SomeClass fc=new SomeClass();veriable1= fc.getSome();  }
   }}
```

Далее SomeClass запаковывается, например, fake.jar. Этот архив прописывается в CLASSPATH и затем компилируется A.java. После этого можно со спокойной душой отдавать A.class заказчику. Он, конечно же, без проблем его декомпилирует, но вот скомпилировать его он уже не сможет, для этого необходимо иметь SomeClass.class. В приведенном примере, конечно же, очевидно, что условие if (!veriable1) никогда не выполнится, но при доле смекалки можно достаточно хорошо спрятать это условие, да и вызов класса SomeClass сделать страшнее, чтобы злоумышленник не сразу догадался, что это только защита кода. А если таких классов сделать пару десятков, то задача компиляции может стать достаточно сложной [1].

Заключение. Из всех вышеперечисленных методов противостояния взломщику ни один не является достаточным для предотвращения нелегального использования программного продукта, написанного на Java. Но вот комплекс этих мер способен реально защитить программу от взлома. Из наиболее эффективных хочется отметить:

1) затемнение кода;
2) применение native-методов;
3) хранение классов в атрибутах.

1. http://togethercrack.nm.ru/index.html
2. http://www.preemptive.com/products.html
3. http://www.4thpass.com/purchase/price.html
4. http://java.sun.com/docs/books/tutorial/native1.1
5. Технологии программирования на Java / Х. М. Дейтел, П. Дж. Дейтел, С. И. Сантри.– Бином-пресс, 2003.– 540с.
6. Thinking in Java, Second Edition/ Eckel B., BruceEckel, 1999 – 730 с.