Для обеспечения неравенста $s_i^e(\tau) < h, i \in I$, задачи (3) в момент $\tau \in \overline{T}_h$ решим двойственными методами [1, 2], взяв в качестве начальных опор оптимальные опоры соответствующих задач, решенных для предыдущего момента $\tau - h$. В момент t_* в качестве начальных опор при решении задач наблюдения (3) возьмем пустые опоры.

В [1, 2] показано, что задачи, сформированные для момента τ, незначительно отличаются от задач, решенных для момента $\tau - h$, и поэтому достаточно небольшого количества итераций для проведения коррекций оптимальных опор. Каждая итерация сопровождается интегрированием прямой или сопряженной системы на небольших промежутках времени, поэтому описанный метод позволяет децентрализованно наблюдать в реальном времени за группой систем достаточно высокого порядка.

Литература

1. *Балашевич Н. В., Габасов Р., Кириллова Ф. М.* // Журнал вычислительной математики и математической физики. 2000. Т. 40. № 6. С. 838–859.
2. *Габасов Р., Дмитрук Н. М., Кириллова Ф. М.* // Изв. РАН. Теория и системы управления. 2002. №2. С. 35–46.

СПОСОБЫ ЗАЩИТЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ НА ОСНОВЕ СОВРЕМЕННЫХ АЛГОРИТМОВ ОБФУСКАЦИИ

А. Н. Титович

ВВЕДЕНИЕ

Развитие средств разработки программного обеспечения сопровождается двумя параллельными процессами: появлением новых механизмов взлома и созданием новых способов защиты от существующих и будущих средств взлома. Этот процесс чем-то напоминает холодную войну, которую непонятно кто и как должен останавливать. С началом использования платформы Java, а затем и платформы .NET также возникла проблема защиты написанных для этих платформ приложений, заключавшаяся в том, что технические особенности Java и .NET (в частности, сам механизм запуска приложений в виртуальной среде) допускают легкую и эффективную декомпиляцию, результатом которой является код, практически идентичный исходному. Обфускация позволяет обойти этот недостаток.

Обфускация (от англ. *to obfuscate* – запутывать, маскировать) – процесс запутывания кода программы с целью затруднить возможность его

понимания при декомпиляции. Обфускация как предмет, подлежащий научному изучению, возникла сравнительно недавно. Первые попытки систематических научных исследований были предприняты еще в начале 90-х. Но существенный толчок к развитию дало распространение Java-технологий. Описанные выше проблемы привели к всплеску развития теории обфускации. Первые серьезные работы в этой области были представлены Collberg, Thomborson, Low, а также Chenxi Wang. В работе [1] были впервые классифицированы известные методы обфускации, а также предложены некоторые аналитические методы измерения уровня обфускации. Был предложен вариант общего алгоритма обфускации для языков высокого уровня, но конкретных примеров реализаций и результатов их работы еще не было. Логическим продолжением стала работа [2], в которой на примере обфускации assembler-кода рассмотрены теоретические и практические аспекты обфускации.

ТЕОРЕТИЧЕСКОЕ ОПРЕДЕЛЕНИЕ ОБФУСКАЦИИ

В зависимости от области применения могут быть даны различные определения обфускации. Следующие определение было предложено в [1] и рассматривает обфускацию как преобразование программы в программу.

Определение 1. Обфусцирующей трансформацией (обфускацией) будем называть такой набор преобразований $T = \{T_1,...,T_n\}$, который из программы $P = \{S_1,...,S_i\}$ формирует программу $P^* = \{...,S_k^* = T_i(S_k),...\}$, обладающую схожим с P поведением (должна сохранить семантику поведения).

Если рассматривать обфускацию с точки зрения криптографии, то можно дать следующее определение

Определение 2. Пусть существует некоторое транслирующее преобразование TR, такое что $P \xrightarrow{TR} B$, где P – исходный код, B – бинарный код программы. TR называется однонаправленным транслирующим преобразованием, если время, которое необходимо затратить на обратное преобразование больше некоей определенной константы T.

Легко заметить аналогию, между однонаправленным транслирующим преобразованием и однонаправленным кодированием, понятием, хорошо изученным современной криптографией.

ИЗМЕРЕНИЕ РЕЗУЛЬТАТОВ ОБФУСКАЦИИ

Способы проверки результатов обфускации можно разделить на две группы: аналитические и эмпирические. Первые оперируют различными

числовыми параметрами и результатами и хорошо подходят для сравнения различных алгоритмов, но не могут дать ответ на главный вопрос: насколько эффективно данное преобразование? Человеческий фактор имеет первостепенное значение в оценке результатов обфускации, и только эмпирическое исследование может дать более-менее точный ответ на этот вопрос. Но, к сожалению, эмпирические методы плохо подходят для широкого применения и интересны в первую очередь с научной точки зрения. Хотя их использование может улучшить эффективность обфускации на этапе проектирования соответствующего программного обеспечения.

Можно выделить следующие категории аналитической оценки эффективности обфускации:

- Сложность обфускации – мера сложности, добавляемой к программе в результате обфускации. Оценивает затруднение понимания программы человеком. Чем выше мощность, тем больше ресурсов уйдет на понимание программы.
- Упругость обфускации – оценивает способность обфускации сопротивляться известным алгоритмам деобфускации. Чем выше упругость, тем больше ресурсов потребуется для автоматической деобфускации.
- Скрытность обфускации – определяет, насколько успешно результаты работы обфусцирующих преобразований могут маскироваться в коде программы (насколько статистические показатели результата схожи с соответствующими показателями входных данных).
- Цена обфускации – обозначает изменение в затратах ресурсов на запуск программы (процессорная мощность, место на диске, место в памяти). Чем меньше цена, тем меньше ресурсов будет затрачивать приложение после обфускации.

Эти меры не зависят друг от друга, поэтому их можно называть ортогональными. Общая мера определяется как комбинация приведенных мер.

МЕТОДЫ ОБФУСКАЦИИ

Методы обфускации – это отдельные трансформации, изменяющие исходный код по определенному принципу. Совокупность тех или иных методов и составляет конечную обфускацию. Современные методы обфускации можно классифицировать следующим образом:

- Лексическая обфускация
  - Замена идентификаторов
  - Индуктивная перегрузка

    - Кодирование текстовой информации
- Изменение управляющего потока
    - Изменение итерационных операторов
    - Условное ветвление
    - Перестановка блоков
    - Распараллеливание потока
    - Дополнительные Inline и Outline вставки
    - Упругие предикаты
    - Изменение представления переменных

Лексическая обфускация является базовой, необходимой для любой полноценной обфускации. Замена идентификаторов и индуктивная перегрузка заключаются в замене всех допускающих обфускацию идентификаторов максимально узким набором бессмысленных имен (A,B,OlllOll,l10 и т.д.) с целью удаления смысловой нагрузки использованных имен идентификаторов. Кодирование текстовой информации необходимо для сокрытия от прямого поиска текстовых элементов программы, определенных пользователем.

Группа методов, изменяющих управляющий поток, объединяет в себе все обфусцирующие трансформации тем или иным образом преобразующие исходный управляющие поток программы. Изменение итерационных операторов (замена итераторов, дополнительные входы/выходы), ложное распараллеливание программы, создание ложных никогда неиспользуемых веток программы, перестановка логических блоков программы, различные комбинации изменения представления переменных – все это вместе с обильным применением оператора GOTO способно *существенно* затруднить взломщику понимание исходного кода программы и поиск в нем интересующих его элементов. Различные примеры реализаций данных методов можно найти в [1] и [2].

Определение 3. Упругим предикатом называется переменная или блок кода программы, значение которых легко вычисляется при выполнении обфусцированной программы, но при деобфускации вычисление их является трудоемким процессом (с точки зрения вычислительной сложности).

Упругие предикаты – один из самых эффективных и перспективных методов обфускации. Основная цель их использования - затруднение автоматической деобфускации. Работа [3] рассматривает использование упругих предикатов для защиты Java-приложений.

Литература

1. *Christian Collberg, Clark Thomborson, Douglas Low*, A Taxonomy of Obfuscating transformations // Technical Report №148, The University of Auckland, Department of Computer Science, 1997

2. *Grzegorz Wroblewski*, General Method of Program code Obfuscating // PhD Dissertation, Wroclaw, 2002
3. *Chenxi Wang*, A Security Architecture for Survivability Mechanisms // PhD Dissertation, The University of Virginia, Department of Computer Science, October 2000

ЧИСЛЕННОЕ РЕШЕНИЕ ЖЕСТКИХ СИСТЕМ НА ОСНОВЕ КУСОЧНО-ГЛАДКОЙ АППРОКСИМАЦИИ

Б. В. Фалейчик

На протяжении многих лет проблема численного интегрирования жестких систем обыкновенных дифференциальных уравнений не теряет своей актуальности. Обусловлено это, прежде всего, практической важностью этого класса задач. Например, к решению жестких систем сводятся многие современные задачи моделирования физических процессов.

Иногда в качестве синонима к термину «жесткость» употребляется понятие «разномасштабность». Разномасштабная дифференциальная задача характеризуется тем, что некоторые компоненты ее решения изменяются во времени намного быстрее, чем другие. В качестве простейшего примера такой задачи можно привести систему из двух линейных дифференциальных уравнений $u'(x) = Au(x)$, где $u(x) = (u_1(x), u_2(x))^T$, $A = diag\{\lambda_1, \lambda_2\}$. При $\lambda_1 >> 1$, $1 >> \lambda_2 > 0$ решение этой системы будет состоять из «быстрой» компоненты $u_1(x) = c_1 \exp(\lambda_1 x)$ и «медленной» $u_2(x) = c_2 \exp(\lambda_2 x)$. Проследим за тем, что произойдет при численном решении этой задачи на ЭВМ явным методом Эйлера на отрезке $[t, t+\tau]$ с начальным условием $u(t) = u_0$.

Пусть значение τ велико, тогда весь отрезок интегрирования обычно разбивают на более мелкие отрезки длины h (для простоты будем считать шаг постоянным) и на каждом шаге находят соответствующее приближение в виде $y_i + h\lambda_i y_i$, $i = 1, 2$. В силу большой скорости изменения $u_1(x)$, для достижения удовлетворительной точности приближения шаг h не должен быть большим. При этом величина $h\lambda_2 y_2$ для второй, «медленной», компоненты решения может оказаться настолько малой, что при ее сложении с y_2 на вычислительной машине произойдет потеря значимости, которая приведет к тому, что значение $y_2 + h\lambda_2 y_2$ останется равным y_2. Если такая ситуация повторится на последующих шагах, то в качестве приближения к $u_2(x)$ на отрезке $[t, t+\tau]$ мы получим константу. В [1] приведен пример возникновения этой проблемы в реальной прикладной задаче. Очевидно, что такое решение как количественно, так