На правах рукописи

Аранов Владислав Юрьевич

МЕТОД И СРЕДСТВА ЗАЩИТЫ ИСПОЛНЯЕМОГО ПРОГРАММНОГО КОДА ОТ ДИНАМИЧЕСКОГО И СТАТИЧЕСКОГО АНАЛИЗА

Специальность 05.13.19 – «Методы и системы защиты информации, информационная безопасность»

Автореферат диссертации на соискание ученой степени кандидата технических наук

Санкт-Петербург – 2014

Работа выполнена в федеральном государственном автономном образовательном учреждении высшего образования «Санкт-Петербургский государственный политехнический университет».

Научный руководитель:

Заборовский Владимир Сергеевич
доктор технических наук,
профессор,
заведующий кафедрой «Телематика» ФГАОУ ВО «Санкт-Петербургский государственный политехнический университет»

Официальные оппоненты:

Котенко Игорь Витальевич
доктор технических наук
профессор, заведующий лабораторией проблем компьютерной безопасности Федерального государственного бюджетного учреждения науки «Санкт-Петербургский институт информатики и автоматизации» Российской академии наук

Воробьев Евгений Германович
кандидат технических наук
доцент, заведующий кафедрой «Информационная безопасность» Санкт-Петербургского государственного электротехнического университета «ЛЭТИ»

Ведущая организация:

Балтийский Государственный Технический Университет "ВОЕНМЕХ" им. Д.Ф. Устинова, г. Санкт-Петербург

Защита состоится «23 » декабря 2014 г. в :00 на заседании диссертационного совета Д 212.229.27 при ФГАОУ ВО «Санкт-Петербургский государственный политехнический университет» по адресу 195251, Санкт-Петербург, ул. Политехническая, 29, ауд. 175 главного здания.

С диссертацией можно ознакомиться в библиотеке и на сайте http://www.spbstu.ru/science/defences.html ФГАОУ ВО «Санкт-Петербургский государственный политехнический университет».

Автореферат разослан « » ноября 2014г.

Ученый секретарь
диссертационного совета:

Платонов Владимир Владимирович

АКТУАЛЬНОСТЬ ТЕМЫ ДИССЕРТАЦИИ

Защита программного обеспечения от неавторизованного использования, модификации и копирования является важнейшей задачей современных информационно-вычислительных систем. Компьютерное пиратство и незаконное использование программ наносит большой вред экономике страны, особенно ее высокотехнологичному сектору. Согласно оценкам специалистов совокупные потери от нелицензионного использования программ постоянно растут, что свидетельствует о необходимости дальнейшего повышения эффективности методов их защиты. Учитывая широкое распространение технологий виртуализации и облачных вычислений, рамках которых прикладное программное обеспечение часто исполняется в недоверенной вычислительной среде, особую актуальность приобретают исследования и разработки, направленные на создание новых методов защиты программ от незаконного использования и обратного проектирования применяемых в них алгоритмов.

Разработка эффективных методов противодействия неавторизованному использованию программных кодов является основой создания надежного барьера на пути распространения нелицензионной программной продукции.

Применяемые в настоящее время методы защиты от нелицензионного использования и копирования программного обеспечения можно разделить на организационно-инфраструктурные и функциональные. Первые направлены на формирование доверенной вычислительной среды, тогда как вторые - на блокирование действий, разрушающих существующие средства защиты программ от копирования и обратного проектирования. Функциональные методы защиты используют различные алгоритмы и методы преобразования исполняемого кода прикладных программ к виду, затрудняющему анализ алгоритмов, положенных в основу их функционирования с помощью технологии реинжиниринга и средств анализа данных. Одним из известных функциональных методов защиты программ является использование запутывающих преобразований (обфускации) исполняемого кода. После таких преобразований программный код может исполняться только на виртуальных машинах (ВМ), оснащенных специальным процессором с набором предназначенных для этого команд и средствами доступа к области оперативной памяти, в которой хранится защищенный программный код и данные. Использование ВМ с процессором, исполняющим обфусцированный код (ПИОК), повышает доверие пользователей к среде исполнения программ, а также обеспечивает защиту программных систем от реинжиниринга или нелицензионного использования.

Учитывая, что при использовании технологии облачных вычислений создание доверенных сред выполнения программ является приоритетной задачей, Применение методов запутывающих преобразований становится важным направлением защиты программ от компьютерных атак, использующих средства статического и динамического анализа исполняемого кода с целью нарушения его целостности и функциональности.

Актуальность решения задач повышения информационной безопасности компьютерных систем, построенных на базе современных технологий виртуализации и облачных вычислений, отмечается многими российскими и зарубежными учёными, в том числе В.А. Захаровым, П.Д. Зегждой, В.П. Бойко, В.С. Заборовским, Н.Н. Кузюриным, А.В. Шокуровым, Р.И. Подловченко, В.Л. Щербина, Н.П Варновским, С.С. Гайсаряном, В.П. Иванниковым, А.И. Аветисяном, К. Коллбергом, К. Сомборсоном, Д. Викстромом, Д.Лоу, Б. Бараком, Д. Бергстромом, Ф. де Клю.

В работах указанных авторов отмечается необходимость создания методов защиты прикладного программного обеспечения, которые обладают высокой стойкостью к компьютерным атакам, организованным с использованием инструментальных средств обратного проектирования. С учетом вышеизложенного, диссертационная работа посвящена разработке методов защиты прикладного программного обеспечения в среде облачных вычислений, уровень доверия к которой не соответствует требованиям принятой политики информационной безопасности и условиям лицензионных соглашений. Особенностью

предлагаемых решений является масштабируемость по отношению к объемам защищаемых программных кодов, и эффективная адаптация к различным операционным средам исполнения программы.

Целью исследования является разработка метода защиты исполняемого программного кода от компьютерных атак обратного проектирования на основе динамического и статического анализа данных.

Для достижения поставленной цели в диссертационной работе были решены следующие задачи:

1. Разработка модели угроз, связанных с использованием технологий обратного проектирования исполняемого программного кода, основанных на динамическом анализе (отладке) и статическом исследовании исполняемого кода.
2. Разработка метода защиты от компьютерных атак обратного проектирования прикладного программного обеспечения, основанного на замещении выбранного критически важного сегмента кода защищенным программным объектом.
3. Разработка масштабируемого алгоритма, позволяющего динамически модифицировать глубину защиты программного кода для его исполнения в среде виртуальных машин.
4. Разработка инструментальных средств, используемых разработчиками программного обеспечения для затруднения использования стандартных методов обратного проектирования за счет преобразований на основе полиномиального алгоритма.

Методы исследования: для решения сформулированных задач использовался аппарат теории графов, теории автоматов, теории защиты информации и методы обратного проектирования программных кодов.

Объект исследования: средства обфускации программного обеспечения, обладающие масштабируемостью и адаптируемостью к различным средам исполнения.

Предмет исследования: метод защиты исполняемого программного кода от статических и динамических средств анализа и обратного проектирования.

Научная новизна работы: метод защиты при помощи запутывающих преобразований программного кода путем создания ПИОК с псевдослучайной архитектурой для его исполнения в недоверенной среде, подвергающейся компьютерным атакам обратного проектирования. В диссертационном исследовании, получены следующие новые научные результаты:

1. Использование конечного автомата в виде сети Петри для осуществления запутывающих преобразований исполняемого кода.
2. Использование решения уравнения в целых числах, решаемого в кольце вычетов, для начальной маркировки сети Петри, обеспечивающую семантическую корректность преобразованного программного кода.
3. Полиномиальное повышение сложности анализа защищенного кода при использовании запутывающих преобразований, основанных на многоуровневой защите исполняемого кода с динамически определяемой глубиной обфускации при помощи последовательности вложенных виртуальных машин с псевдослучайной архитектурой и системой команд.

Положения, выносимые на защиту:

1. Модель угроз обратного проектирования машинного кода при условии доступа к исполняемому программному коду.
2. Алгоритм создания ПИОК с псевдослучайной архитектурой, исполняющих защищенный от обратного проектирования код программ, предназначенных для выполнения в вычислительной среде с возможностью неавторизованного доступа к исполняемому коду.

3. Метод защиты исполняемого кода, основанный на запутывающих преобразованиях, которые используют изменение потока управления и внесения избыточности с помощью сетей Петри.
4. Алгоритм определения начальных состояний сети Петри, основанный на решении системы уравнений, решаемых в целых числах, в кольце вычетов, получаемом на основе разрядности инструментального процессора.
5. Архитектура программной системы, предназначенной для затруднения статического исследования защищаемого кода и противодействия динамическому анализу.

Обоснованность и достоверность представленных в диссертационной работе научных положений обеспечивается формальным доказательством полученных результатов и апробацией полученных результатов в печатных трудах и докладах на всероссийских и международных научных конференциях.

Практическая значимость работы. Результаты исследований, полученные в ходе выполнения диссертационной работы, были успешно апробированы при выполнении государственного контракта на проведение НИР (шифр «Защита-НД») по заказу министерства промышленности и торговли РФ за номером 192/11-ЭКБ-27.09ок в рамках работ по федеральной целевой программе "Развитие электронной компонентной базы и радиоэлектроники" на 2008-2015 годы, а также в учебном процессе и научных исследованиях на кафедре «Прикладная математика» ФГБОУ ВПО «СПб ГПУ» в рамках курса «Защита информации: Защита программных продуктов от нелегального копирования и методы исследования вредоносного ПО»

Апробация и публикация результатов работы. Основные результаты исследования обсуждались на Общероссийской научно-технической конференции «Информационная безопасность регионов России» 2013, г. Санкт-Петербург; на XLII научно-практической конференции с международным участием «Неделя науки СПб ГПУ»; на Научном семинаре «Проблемы современных информационно-вычислительных систем» под руководством д. ф.-м. н., проф. В. А. Васенина в МГУ им. Ломоносова.

Основные результаты и положения работы опубликованы в 5 научных статьях, в том числе 2 статьи в изданиях, входящих в перечень Высшей аттестационной комиссии Министерства образования и науки Российской Федерации.

Структура и объем диссертационной работы. Диссертационная работа объемом 113 машинописных страниц, содержит введение, четыре главы и заключение, список литературы, содержащий 37 наименований. Общий объём работы – 131 с., 19 рис., 17 таб.

СОДЕРЖАНИЕ ДИССЕРТАЦИИ

Во введении обоснована важность и актуальность темы диссертации, определены цель и задачи исследований, показана научная новизна и практическая значимость.

Первая глава содержит анализ современных подходов к созданию средств защиты прикладного программного обеспечения от компьютерных атак, организованных с целью обратного проектирования, несанкционированного копирования и модификации исполняемых кодов. Указываются преимущества использования методов, обеспечивающих защиту от различных средств статического и динамического анализа исполняемого кода с помощью запутывающих преобразований. Важной характеристикой таких преобразований является оценка количества информации по Колмогорову, которая характеризует длину программы реализующей целевой алгоритм. Показано, что сложность обратного проектирования таких алгоритмов с помощью кода исполняемой программы, которая модифицирована в процессе применения запутывающих преобразований, количественно выражается как относительная мера увеличения числа используемых машинных команд. Анализируется возможность применения метода защиты программного кода, основанного на использовании запутывающих преобразований применительно только части исполняемой программы. Показано, что

сформированный байт-код может выполняется с использованием ПИОК с псевдослучайной архитектурой (рис.1). В этом случае интерпретатор ПИОК может стать частью исполняемого кода, к которому применяются запутывающие преобразования, построенные на базе сетей Петри. Преимуществом такого метода обфускации является то, что алгоритмы преобразования исполняемого кода имеют полиномиальную вычислительную сложность, в то время как алгоритм восстановления защищенного кода, с целью восстановления состояния кода до защиты, имеет временную сложность NP, при условии секретности конфигурации используемой сети Петри.

Рис. 1. Метод защиты исполняемого программного кода

Анализируются недостатки существующих методов обфускации и показано, что для их преодоления можно использовать процедуры поэтапного преобразования исполняемого кода, инвариантом которого является семантика целевого алгоритма. В главе делается вывод о перспективности, актуальности и практической значимости применения метода поэтапного преобразования, а также выдвигаются требования к программным и пользовательским интерфейсам. В конце главы приведена постановка задачи исследования, которая базируется на анализе исполняемого кода с помощью инструментальной системы трансформации и оптимизации программ LLVM (Low Level Virtual Machine), формировании списка всех доступных для защиты подпрограмм, находящихся в объектном файле, анализа набора инструкций, используемых подпрограммой, а также выбора критического блока машинных инструкций, которые трансформируется в байт-код команд ПИОК (рис. 2).

При этом замещение защищаемого исполняемого кода происходит на уровне вызова удаленной подпрограммы интерпретатором ПИОК с параметрами, идентичными тем, которые использовались в исходном исполняемом коде с добавлением адреса точки входа в байт-код защищенной подпрограммы.

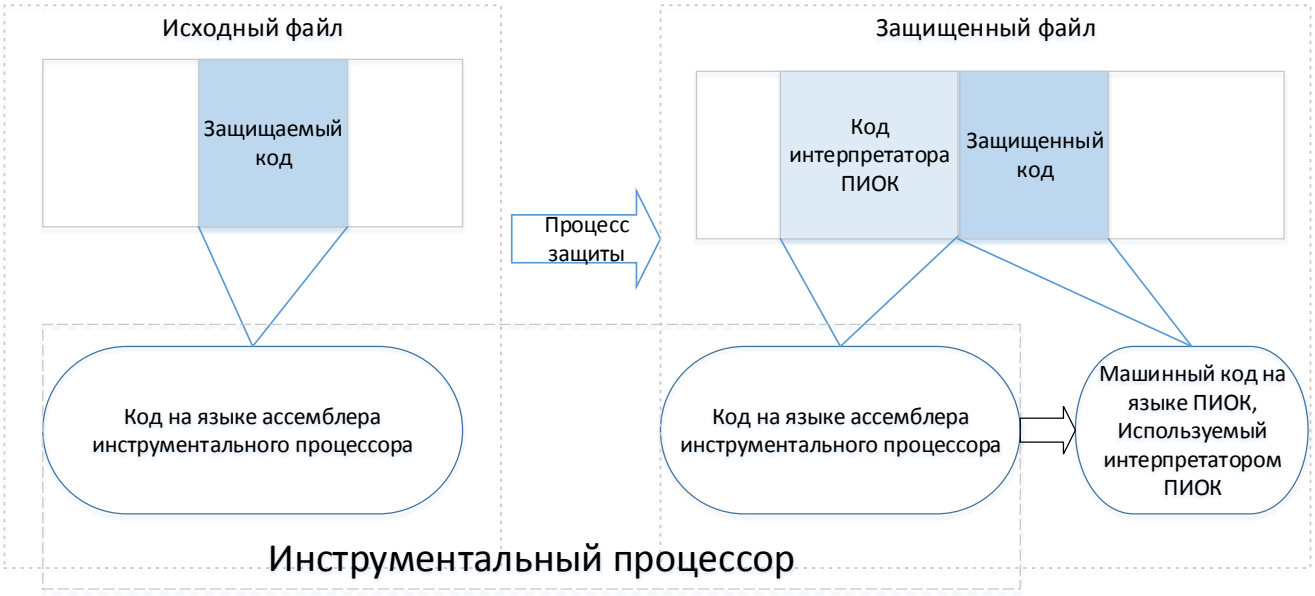

Рис. 2 – Пример незащищенного (исходного) и защищенного программного кода

Во второй главе разработаны теоретические основы реализации предложенного метода защиты с помощью формирования байт-кода, исполняемого с помощью виртуальных машин с псевдослучайной архитектурой ПИОК, и генерации интерпретатора ПИОК при помощи автоматных моделей, основанных на сетях Петри.

Предложена модель угроз обратного проектирования исполняемого кода программы с помощью методов динамического и статического анализа. Модель угроз представляет собой множество из трех компонент <S, O, A>, где S – субъект атаки, O – объект атаки, A – атакующее действие. Показано, что структура компонент модели может быть представлена следующим образом:

- Субъект атаки S = <s, ss, is>:
  - Множество автоматизированных средств динамического анализа кода (s): отладчики, средства мониторинга вызовов сервисов операционной системы.
  - Множество полуавтоматических средств анализа кода (ss): дизассемблеры, анализаторы графа достижимости.
  - Множество интеллектуальные средства анализа (is): семантическая сеть, генетические алгоритмы.
- Объекты атаки O = <m, t, p>:
  - m - алгоритм, который реализует исполняемый код программы. Обычно он недоступен для атакующего воздействия, так как кодируется на языке высокого уровня (ЯВУ) и потому не предается в недоверенную вычислительную среду.
  - t - код программы на ЯВУ. Обычно он также недоступен для атакующего воздействия, так как транслируется в машинный код инструментального процессора компилятором и потому не предается в недоверенную вычислительную среду.
  - p - машинный код. Непосредственно передается в недоверенную вычислительную среду и, поэтому, непосредственно доступен для атаки.
- Атакующее действие A = <o, c>, **A**= <A, b>:
  - o - операция по сбору данных о переменных, адресах областей памяти исполнения кода и их порядка выполнения.
  - c - преобразование кода к текстовому формату.

- b – наблюдение за поведением защищаемого алгоритма с использованием парадигмы «черного ящика» путем анализа входных и выходных данных.

Учитывая, что в соответствие с современной практикой использования коммерческого программного обеспечения, конечному пользователю ни исходный код программы, ни внутренняя документация разработчика программного обеспечения не передаются, а защита объектов m и t осуществляется с помощью организационно-инфраструктурных методов, поэтому в работе не рассматривается.

В результате модель угроз обратного проектирования может быть редуцирована до множества <S, p, A>. В дальнейшем объектом рассмотрения являются компьютерные атаки на исполняемый машинный код, для защиты от которых предложен новый вид запутывающих преобразований, формирующих байт-код для ПИОК с псевдослучайной архитектурой.

Важной особенностью предложенного метода защиты является то, что исполняемый на ПИОК байт-код не требует специальных привилегий, так как для своего выполнения не нуждается в установке новых драйверов и не использует специальных системных инструкций. Показано, что применение ПИОК, кроме обеспечения защиты кода от атак обратного проектирования, позволяет сформировать независимую от инструментального процессора среду выполнения различных прикладных задач.

Программная реализация разрабатываемого метода, в дальнейшем называемая протектором, также формирует машинно-независимый код, исполняемый интерпретатором ПИОК.

Систему команд ПИОК предложено формировать так, чтобы каждая инструкция имела: символическое имя, код (уникальное число, однозначно характеризующее действие) и набор аргументов-операндов, которые параметризируют действие. В результате набор машинных команд ПИОК можно сгенерировать таким образом, чтобы количество информации по Колмогорову в защищаемой сегменте исполняемого кода увеличилось: $Q_p > Q_u$, где Q_p, – количество информации в защищенном коде, а Q_u– количество информации в незащищенном коде, а сама структура команд имела псевдослучайный характер. Эффект увеличения количества информации по Колмогорову достигается за счет таких факторов как: добавление обфусцирующих вычислений, изменение набора исполняемых инструкций и включение в защищаемый сегмент кода интерпретатора ПИОК, который становится неотъемлемой частью исполняемого алгоритма (рис. 2). Последовательность команд ПИОК, реализующая целевой алгоритм в дальнейшем рассматривается как виртуальная программа, в которой обфускации подвержен как целевой алгоритм, так и исполняемый байт-код.

С учетом того, что для анализа защищаемого кода используется инструментальная система LLVM, для реализации разработанного метода обфускации предложено использовать архитектуру ПИОК программным счетчиком.

Показано, что скорость исполнения защищенных сегментов программы обеспечиваются эффективностью процесса компиляции исполняемого кода в байт-код ПИОК и возможностью запуска разработанного протектора под управлением различных операционных систем и архитектур инструментальных процессоров. Это обеспечивается тем, что сформированный в результате обфускации байт-код ПИОК исполняется без преобразования в машинный код инструментального процессора. В результате, статический анализ кода существенно усложняется так как требует анализа архитектуры ПИОК и создания дизассемблера, распознающего используемую псевдослучайную архитектуру. Таким образом, в процессе защиты кода для его реализации используется ПИОК, обладающий случайным для данной реализации набором машинных команд, что требует создания специализированного программного обеспечения для каждого конкретного ПИОК, принципиальным образом усложняя процесс обратного проектирования целевых алгоритмов. Так как определенный блок инструкций ассемблера инструментального процессора может быть трансформирован в различные блоки инструкций для каждой виртуальной машины (рис. 3), то преобразование,

основанное на предложенном методе обфускации, затрудняет процесс дизассемблирования программ, повышая доверенность среды выполнения команд виртуальной программы.

Показано, что интерпретатор, реализующий ПИОК, может быть реализован в виде оператора типа «switch» с большим количеством условий, соответствующих различным командам ПИОК, где код, содержащийся в каждом условие реализует ту или иную машинную инструкцию ПИОК. При этом система команд для конкретного ПИОК динамически формируется в процессе реализации предложенного метода защиты.

Коды машинной инструкции ПИОК или *опкод* определяет последовательность действий, которая должна быть выполнена ПИОК вплоть до команды выхода, например:

```
while (1) {
  opcode = NextOpcode();
  if (HasArg(opcode))
    oparg = NextArg();
  switch (opcode) {
     case opCode1: // код на ЯВУ реализующий семантику opCode1
     case opCode2: // код на ЯВУ реализующий семантику opCode2
     ... // и т.д. для всех опкодов ПИОК
  }
}
```

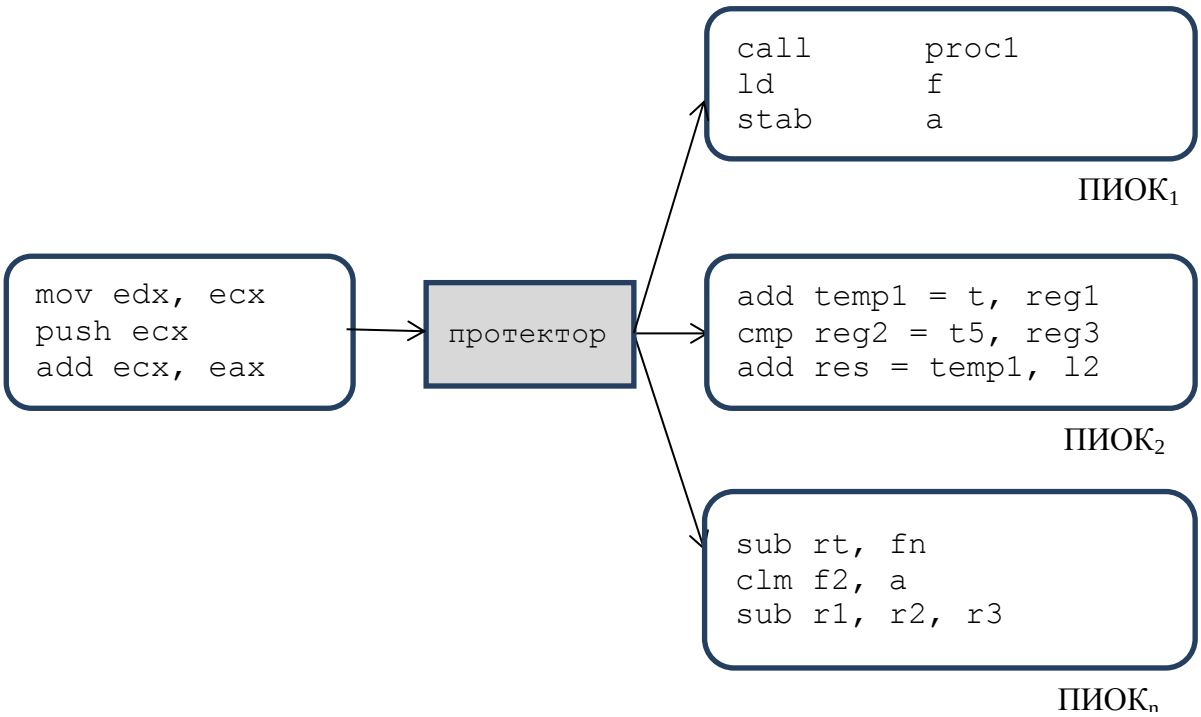

Рис. 3 - Генерация кодов ПИОК на основе одного блока исходного кода для различных ПИОК при последовательных запусках протектора на одних и тех же исходных данных

Преимуществом предложенного подхода к защите исполняемого программного кода при помощи обфускации по отношению с существующим аналогам, используемым в промышленных решениях, является то, что в зависимости от архитектуры ПИОК, операнды опкода могут быть значениями регистров, значениями стека, ссылками на ячейки памяти или непосредственными значениями. Другой важной особенностью предложенного метода защиты является возможность его рекурсивного применения, при котором защищенность интерпретатора ПИОК обеспечивается применением метода обфускации на основе сетей Петри.

.

Метод защиты исполняемого кода с использованием запутывающих преобразований на основе сетей Петри может применяться к каждому конкретному базовому блоку защищаемой программы в отдельности. Это обеспечивает последовательное выполнение всех раундов сети Петри, которые определяются как одновременная попытка выполнения всех переходов, то есть перемещения меток из одной позиции в другую. Реализация метода защиты исполняемого кода с использованием обфускации при помощи сетей Петри включает в себя пять этапов. На первом этапе выбирается один или несколько базовых блоков защищаемой программы максимального размера, т.к. чем больше размер защищаемого базового блока, тем выше эффективность метода защиты, что иллюстрируется результатами в четвертой главе. На втором этапе выбирается множество защищаемых значений, которые могут быть как значениями констант, так и начальным адресом ячейки памяти выбранного базового блока. Пример защищаемого базового блока представлен на рис. 4.

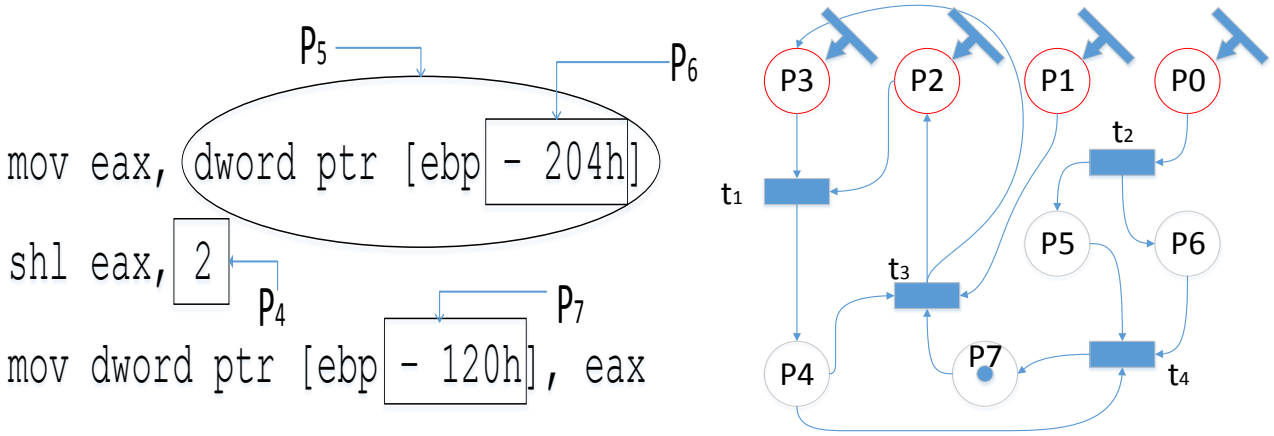

Рис. 4 Базовый блок программы с 4 параметрами (P_4-P_7) (слева) и сеть Петри для его обфускации (справа)

На рис. 4 представлено два вида защищаемых значений: константы, обозначенные прямоугольниками и четырехбайтовое значение, лежащее по адресу ячейки памяти, обозначенной овалом. Выделено два типа позиций сети Петри: входные позиции, значения которых известны на момент выполнения защищенного блока и динамически получаемые (P_4-P_7).

На третьем этапе выбирается количество раундов сети Петри с учетом того, что увеличение числа раундов создает большую запутанность кода для выбранного базового блока защищаемой программы. Важным преимуществом метода, повышающего его практическую стойкость, является то, что часть значений ячеек памяти, связанных с метками сети Петри, можно выбрать случайным образом. На четвертом этапе осуществляется выбор шаблона сети Петри, размер которого зависит от количества защищаемых значений, что достигается за счет операции масштабирования с использованием принципа функционального самоподобия. Пример масштабирования сети Петри с двукратным увеличением числа позиций сети приведен на рис. 5. При этом, в полученной сети, каждая позиция имеет не более одного входа. Одновременно, каждая выходная позиция обязана иметь, как минимум, один вход. На пятом этапе производится определение начальных пометок сети Петри при помощи решения системы уравнений в поле $Z_{2^{31}-1}$, определяемом разрядностью инструментального процессора (в нашем случае - 32). Так, в случае использования сети Петри с восьмью позициями и всеми разрешенными переходами (рис.4), полная система уравнений в целых числах для выбора начальных пометок имеет вид:

$$\begin{cases} x_2 = x_7 + x_1 + x_4 \\ x_3 = x_7 + x_1 + x_4 \\ x_4 = x_3 + x_2 \\ x_5 = x_0 \\ x_6 = x_0 \\ x_7 = x_5 + x_6 + x_4 \end{cases}, \quad (1)$$

где x_i – значения пометок сети Петри. При составлении первого уравнения в (1) учитывается то, что согласно рис. 4, в позицию p_2 пометки могут прийти только из позиций 1, 4 и 7. Система (1) получена преобразованием матрицы инциденций сети Петри на рис.4.

Для вычисления начального состояния сети Петри предложено редуцировать систему уравнений в целых числах до описания только разрешенных на текущем раунде переходов. Так на первом раунде система (1) редуцируется к уравнению: $\{x_{4_1} = x_{3_1} + x_{2_1}$, на втором – к уравнению $\{x_{2_2} = x_{7_2} + x_{4_2}$ и так далее. В результате задача достижимости подмаркировки для конкретной конфигурации сети Петри решается за полиномиальное время[1], зависящее от размерности системы используемых уравнений в целых числах. Результатом выполнения данного этапа является базовый блок со вставленными инструкциями расчёта раундов работы сети Петри.

Теоретическую оценку безопасности предложенных методов обфускации можно провести на основе экспертных оценок, полученных при анализе решений с похожими свойствами. Каждый псевдослучайный ПИОК увеличивает сложность взлома на мультипликативную константу. Система защиты на основе сетей Петри эффективна имеет более чем полиномиальную сложность взлома только в случае секретности сети на рис. 4. В том же случае если метод, начальное значение генератора псевдослучайных чисел (ГПСЧ) и все возможные используемые сети Петри известны, то восстановление обфускации на основе сети Петри тривиально и занимает O(n) + O(m^3), где n – число инструкций в защищаемом блоке, а m – число защищаемых значений. В случае неизвестности начального значения ГПСЧ восстановление возможно за время O(n^2) + O(m^3). В том же случае, когда известен только метод – полиномиальный алгоритм преобразования защищенного кода к исходному виду неизвестен.

Третья глава посвящена практической реализации разработанного метода запутывающих преобразований и возможностям повышения его эффективности на основе применения алгоритма многоуровневой масштабируемой виртуализации исполняемого кода в байт-код ПИОК, выражающейся в увеличении оценки количества информации по Колмогорову.

Для создания промежуточного представления защищаемого кода предложено использовать инструментарий LLVM, который предоставляет программные интерфейсы для трансформации исполняемого во время компиляции, компоновки и выполнения.

Показано, что максимальное количество операндов исполняемого кода для процессоров x86 равно 7 в случае использования 8 битовых операндов. В главе предложен алгоритм кодирования опкодов, позволяющий обеспечить сокрытие структуры семантических близких инструкций. Показано, что для сохранения семантической целостности защищаемой программы алгоритм преобразования должен биекцией. Например, если для каждого опкода сгенерировать 3 произвольных значения:

key1 = randFromTo(0, 7);
key2 = randFromTo(0, 0xFF);
key3 = randFromTo(0, 7).

то, автоморфные преобразования будут иметь следующий вид:

b = ror(b, key1), циклический сдвиг вправо;

[1] Авдошин С.М., Савельева А.А. Алгоритм решения систем линейных уравнений в кольцах вычетов / С.М. Авдошин //Информационныетехнологии. 2006.No2.с.50-54.

b = b ^ key2, поразрядное исключающее или;
b = rol(b, key3), циклический сдвиг влево.

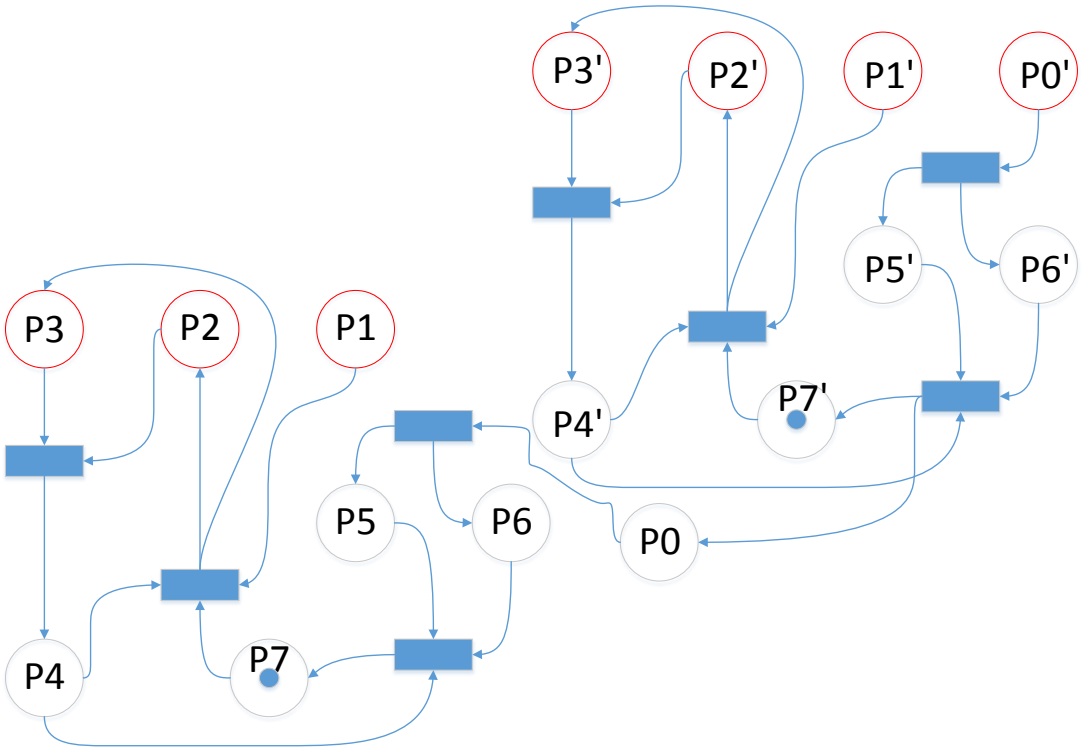

Рис. 5 Масштабирование сети Петри путем замены одного из начальных состояний сети на вычисляемое значение

Показано, что для автоматической генерации системы команд ПИОК необходимо сформировать множество регистров процессора R, которые задаются кортежем: $\forall r \in R$, где r = <name, size>.

Предложенный алгоритм позволяет упростить правила построения множества всевозможных инструкций ПИОК, но при этом повысить сложность алгоритмов статического анализа программного кода, так как в имени каждого регистра кодируется информация о размере регистра и типе кодирования хранимых в регистре данных. Причем данная информация доступна только на этапе создания ПИОК, поэтому недоступна для использования с целью анализа исполняемого кода. В результате набор создаваемых кодов (имен) регистров становится уникальным, что позволяет скрыть структуру построения кодов регистров, используя автоморфные преобразования, аналогичные преобразованиям для кодов инструкций.

Исходя из ограничений на реализацию, накладываемых инструментальными средствами LLVM для всех ПИОК введем следующие параметрические ограничения: максимальное количество регистров не меньше двух и не превышает целого числа n, которое динамически определяется на этапе формирования требований к уровню защищенности исполняемого кода. Для формирования допустимого множества архитектур ПИОК, предложено задавать следующие наборы операций, которые соответствуют группам команд LLVM, выбранных по признаку сходства функциональных характеристик: группа пересылки данных между памятью и регистрами, арифметические операции и т.д. При этом, каждая операция из выбранного набора может быть реализована с использованием одной или нескольких операций из следующих групп, а именно: арифметические; логические; сравнения; управления потоком выполнения; сдвига; со стеком; пересылки данных.

Показано, что перечисленные группы задают полный набор операции языка виртуальной машины LLVM и поэтому позволяют описать всё многообразие псевдослучайных архитектур ПИОК. Учитывая, что большинство инструкций, реализующих данные операции, допускает

довольно широкий спектр операндов, как с точки зрения синтаксиса, так и с точки зрения семантики, для генерировании инструкций ПИОК и соответствующих им опкодов предложено использовать следующую последовательности шагов:

1) указывается опкод, отображающий семантику команды;
2) для команд с вещественными операндами перед соответствующей командой ставится префикс f;
3) определяются размеры операндов (8, 16, 32, 64 или 80 бит) – одно число, если у всех операндов один и тот же размер, несколько чисел, каждое из которых соответствует размеру одного из операндов, если операнды имеют разный размер. Так как после размера операнда указывается его тип, визуально эти значения разделены;
4) после размера указывается тип операнда:
    - m – ячейка памяти,
    - r – регистр,
    - i – непосредственное значение,
    - b – регистр, содержащий адрес ячейки памяти;
5) если команда имеет различные операции для знаковых/беззнаковых соответствующих инструкций, то после указания типа операнда (операндов) указывается знаковый тип операнда (операндов):
    - s – знаковый (signed),
    - u – беззнаковый (unsigned);
6) для команд управления потоком выполнения определяется тип перехода: abs – абсолютный переход, переход по адресу в памяти, rel –относительный переход, переход относительно текущей позиции;
7) через пробел указываются операнды - роль и тип (источник – приемник).

Для создания большого числа инструкций ПИОК определены следующие правила наименования. Источник данных в инструкции обозначается через i (input), приемник через o (output), операнд, одновременно являющийся приемником и источником через io. Для команд условных переходов у второго операнда стоит буква j – jump, что говорит о том, что этот операнд определяет смещение относительно текущего значения счетчика команд; для команд сдвига второй операнд с префиксом sh определяет сдвиг (shift). По аналогии также вводится соглашение о наименовании остальные необходимых типы операндов:

- reg –регистр,
- mem –ячейка в памяти,
- imm - непосредственное значение,
- brg - регистр, содержащий адрес ячейки памяти.

Введение данных соглашения исчерпывающим образом определяет множество возможных, но при этом семантически различных команд ПИОК.

Пример описания:

div64s_mr io_mem, i_reg – команда знакового деления 64 битных значений, где первый операнд – ячейка памяти, второй – регистр, при этом первый операнд является одновременно приемником и источником, а второй только источником.

Для обеспечения возможности компилирования произвольного не содержащего привилегированных инструкций алгоритма при помощи LLVM в байт-код ПИОК, данный ПИОК должен содержать команды из всех вышеперечисленных семи групп. Показано, что некоторые из этих 7 групп можно разделить на подгруппы (например, операции пересылки данных разделяются на 3 подгруппы: загрузка константы в регистр, загрузка из регистра в память и загрузка из памяти в регистр) формируя 17 различных подгрупп, то есть в каждую группу включается от 1 до 3 подгрупп. В том случае если в наборе инструкций присутствует хотя бы один представитель для каждой из этих 17 подгрупп, то такой набор инструкций предлагается называется полным.

Таким образом, у различных ПИОК системы команд существенно различаются по количеству, составу, а также семантикой. У современных инструментальных процессоров количество команд достигает нескольких сотен, поэтому ПИОК должен иметь сравнимое число опкодов. С целью формирования системы команд ПИОК предложено для каждого типа инструкций выбрать инструкцию, работающую только с одним конкретным типом данных. Этот тип данных для каждой группы выбирается произвольно и независимо от других групп. Данное правило применяется для всех групп инструкций за исключением инструкций пересылки данных.

Показано, что разработанная система команд ПИОК является функционально-полной, при этом инструкции, реализующие операции из одной группы команд, могут быть заменены на другие инструкции из той же самой группы. Используя данное свойство, предложен подход, который позволяет управлять набором инструкций ПИОК и формировать его размер исходя из требований защищаемого алгоритма. В главе приведены примеры замены инструкций в процессе реализации предложенного метода обфускации, в качестве которых выбраны логические инструкции, инструкции сравнения, инструкции циклического сдвига и пара инструкций (add, sub), целочисленного деления, сложения, сравнение и условной передачи управления.

Все инструкции, входящие в архитектуру ПИОК, предложено использовать с одним произвольным размером операндов, а из подгрупп инструкций выбрать по одному опкоду. Полученный в результате набор инструкций является достаточным для того, чтобы сопоставить любой инструкции языка LLVM, одну или несколько команд ПИОК. Разработанный алгоритм позволяет увеличивать выбранный набор инструкций до заданного числа, так чтобы число инструкций ПИОК было сопоставимо с набором инструкций ассемблера инструментального процессора. Показано, что после проведения преобразований количество информации по Колмогорову в коде, реализующем защищаемый алгоритм, увеличивается, обеспечивая рост временных затрат на решение задачи обратного проектирования защищаемого алгоритма.

Обфускацию кода при помощи сети Петри предложено реализовывать с помощью оптимизационного прохода LLVM, в котором используется несколько (семь в текущей реализации) различных структур сетей Петри с возможностью их масштабирования на основе использования принципа самоподобия (рис. 5).

Применение методов создания полиморфного кода позволяет производить самомодификацию кода во время выполнения, включая изменение имен регистров и модификацию машинных команд. При этом изменение имен регистров происходит во время выполнения кода, используя различные арифметические и логические операции. Таким образом, эффективность использования стандартных средств динамического анализа запущенной программы и статического дизассемблирования снижается, что усложняет поиск сигнатур в исполняемом коде и тем самым препятствует его изучению. Модификация инструкций осуществляется аналогично с модификацией кодов регистров, но при сохранении оригинального алгоритма его работы, причем исполняемый код модифицируется каждый раз при создании новой копии. Следует отметить, что использование данного метода возможно только на последнем этапе, т.к. дальнейшая защита самомодифицирующегося кода невозможна.

Предложенный метод защиты исполняемого кода при помощи обфускации использует технологию создания метаморфного кода, при помощи генератора метаморфного кода, который состоит из следующих функциональных частей:

- блока переименования регистров,
- подсистемы изменения порядка выполнения и подмены инструкций,
- блока встраивания, перестановки, переноса подпрограмм и вставки кодового «мусора».

Разработанный генератор метаморфного исполняемого кода интерпретатора ПИОК повышает эффективность реализации методов обфускации как при помощи псевдослучайных ПИОК на любом уровне вложенности виртуальной машины, так и при помощи сетей Петри даже к том случае, когда объем защищаемого сегмента программы имеет малые размеры.

В результате, генерация исполняемого кода основывается на создании модуля компиляции, содержащего только защищаемый код. Этот модуль являлся результатом работы пятого этапа алгоритма защиты при помощи сети Петри, особенности которого описаны во второй главе. Показано, что в случае, если в качестве исходных данных был использован модуль компиляции на ЯВУ, то данный этап реализации алгоритма преобразований является завершающим, так как не требуется модификация объектного файла и создание объектного файла с дополнительным сегментом, содержащим защищенных исполняемый код, а возможно использование стандартных алгоритмов создания объектных файлов, используемых компилятором.

Количество этапов реализации алгоритма получения с защищенного исполняемого кода зависит от того, находилась ли защищаемая функция в отдельной секции объектного файла. В случае если функция лежала в отдельной секции, то преобразование объектных файлов происходит в следующей последовательности: из исходного объектного файла удаляется секция с обфусцированной функцией и в конец файла дописывается секция с новой, защищенной функцией. Показано, что сегмент исполняемого кода с незащищенной функцией может заменяться на псевдослучайную последовательность байтов. При этом все используемые ссылки преобразуются в символы с автоматически сформированными именами, а сегмент кода с защищенной функцией дописывается в конец секции кода файла, содержащего код, подлежащий защите. При обфускации интерпретатора ПИОК код интерпретатора дописывается в конец файла с байт-кодом защищенной функции.

Полученные в главе результаты показывают, что реализация предложенного комбинированного метода обфускации исполняемого кода (рис.1), повышает защищенность прикладного программного обеспечения путем увеличения количества информации по Колмогорову за счет добавления в защищаемый алгоритм дополнительных вычислений (раунды сети Петри, изменения регистров интерпретатора ПИОК и т.д.) и предоставляет разработчикам новые возможности использования программных объектов, предназначенных для выполнения в недоверенных вычислительных средах.

В четвертой главе производится исследование разработанного метода защиты (рис.1) исполняемого программного кода на основе запутывающих преобразований, реализуемых с помощью вложенной виртуализации и применении сетей Петри, а также проводится анализ эффективности разработанного метода при противодействии атакам, основанным на технологиях обратного проектирования.

Исследования проводятся на основе разработанного второй главе подхода к построению ПИОК, содержащий набор инструкций достаточный для выполнения целевого алгоритма, реализованного с помощью исполняемого кода инструментального процессора. В процессе проведения исследований реализовано 7820 инструкций виртуального процессора. Сгенерированный набор по построению включает множество инструкций с различной семантикой и размером операндов.

При использовании всего набора инструкций размер защищенных исполняемых кодов файлов будет гораздо больше исходных, увеличивая количество информации по Колмогорову, так как интерпретатор ПИОК записывается в защищенный исполняемый файл. С целью повышения эффективности процесса исследований и формирования количественных оценок разработан псевдослучайный алгоритм, который из всего многообразия инструкций выбирает функционально-полный набор, причем состав инструкций из этого набора обновляется при каждом запуске генератора ПИОК.

Приводятся оценки, указывающие на количественные и качественные отличия между представлениями байт-кода исходного и защищенного исполняемых файлов, что затрудняет процесс анализа алгоритма, команд и операндов. При формировании оценок, характеризующих алгоритмическую сложность решения задач обратного проектирования учитывается то, что при реализации предложенного метода защиты исполняемого кода генерируется виртуальная машина с псевдослучайной архитектурой, а эффективных программных инструментов для

динамического анализа таких исполняемых байт-кодов не существует, Показано, что применяемый метод существенно повышает алгоритмическую сложность процесса обратного проектирования при использовании процессорных плагинов, что делает их применение критически зависимым от особенностей псевдослучайной ПИОК.

В случае использования ПИОК, одним из методов обратного проектирования является частичный анализ машинного кода интерпретатора ПИОК с последующим частотным анализом опкодов ПИОК. Поэтому, кроме увеличения меры количества информации по Коломогорову, эффективность предложенного метода рассматривается и с помощью частотного анализа защищенного байт-кода (таб. 1). Показано, что частота появления опкода, соответствующего определенной команде, в длинных текстах разных программ не изменяется, что создает дополнительный уровень защищенности, затрудняя применение статистических методов отладки в том числе на основе анализа парных корреляций.

Таблица 1

Таблица частот опкодов для исследуемой совокупности защищенных файлов с частотой более 5%

Код операции	Относительная частота (защищенный код)	Относительная частота (незащищенный код)
00	0.26	0.01
CF	0.14	0.01
44	0.14	0.01
45	0.08	0.02
A9	0.06	0.00

Частотный анализ алгоритма вычисления синуса методом разложения в ряд Тейлора для защищенного и незащищенного кода, результаты которого представлены в таблице 1 показывает, что в защищенном коде самой часто встречающейся командой является опкод, закодированный нулем, реализующий операцию вычитания двух чисел с фиксированной запятой, что при осуществлении анализа кода позволяет предположить, что данная команда, закодированная нулем является командой пересылки данных, так как именно команды пересылки данных являются чаще всего встречающимися в незащищенной версии данного алгоритма. В тоже время в данном примере ПИОК нулем кодируется инструкция вычитания, частота которой в незащищенном коде составляет менее 1%.

Проведенное исследование показывает эффективность противодействия частотному анализу с помощью модификации, основанной на добавлении инструкции, не влияющих на алгоритм функционирования программы, то есть не изменяющих семантику защищаемого алгоритма, но меняющих частотную характеристику появления команд в защищенном файле. В главе предложен вариант реализации разработанного метода, позволяющего получить исполняемые файлы с равномерным распределением байт-кодов инструкций ПИОК.

Разработан подход для оценки качества защиты исполняемого кода при помощи сетей Петри с использованием информации по Колмогорову, где в качестве меры выбрано количество инструкций, реализующих алгоритм, с помощью машинных команд инструментального процессора x86. Показано, что наибольшие трудности применения разработанного метода обфускации при помощи сетей Петри связаны с защитой исполняемых сегментов, которые содержат малое количество инструкций. На рис. 6 приведен пример обфускации блока размером в 480 инструкций для инструментального процессора x86, который показывает, что для достижения эффективного увеличения количества информации, которая служит мерой качества, проведенной обфускации, все три основных настроечных параметра, а именно число случайных меток, количество неслучайных меток и число раундов сети Петри должны изменяться совместно. Совместное увеличение всех трех параметров необходимо для обеспечения неограниченного роста количества инструкций в защищаемом блоке, в противном

случае (рис. 6) график зависимости числа инструкций лишь от одного из параметров выходит на уровень насыщения.

Показано, что за счет изменения даже одного из перечисленных выше параметров, количество информации по Колмогорову увеличивается в 4 раза, что эквивалентно экспоненциальному росту сложности процесса обратного проектирования, т.к. задача проверки эквивалентности программ трудноразрешима. На примере малых по размеру блоков исполняемого кода показано, что хотя выбор большого числа неслучайных меток затруднен, но применение предложенного метода обфускации, тем не менее, позволяет осуществлять защиту кода, что показано на рис.6, где даже в случае выбора только одной неслучайной метки достигнуто увеличение числа инструкций более чем в 4 раза. В случае же если количество неслучайных меток не ограничено, то можно достичь произвольной, наперед заданной, меры количества информации для защищенного кода и, таким образом, повысить защищенность исполняемого кода до требуемой величины.

Проведенные в главе исследования показали, что разработанный метод обфускации устойчив к стандартным методам преобразования к незащищенному виду, а организация компьютерной атаки на защищенные программы сопряжен со значительными вычислительными и организационными трудностями, которые связаны с необходимостью динамического анализа многопоточного кода и большого числа одновременно изменяющихся параметров. Исследования показали, что применение метода обеспечивает увеличение стойкости кода к методам обратного проектирования за счет масштабирования защиты исполняемого кода основанной на преобразовании машинного кода в байт-код ПИОК и совместного использования различных методов обфускации.

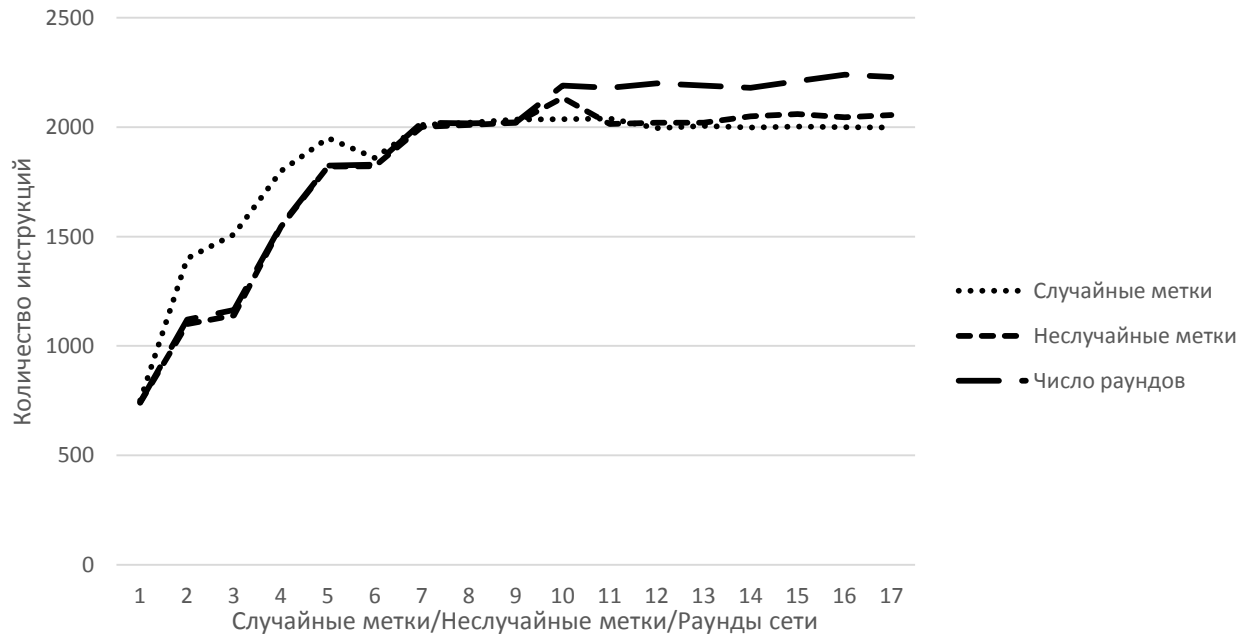

Рис. 6 Количество инструкций в коде после деобфускации при помощи стандартного инструментария анализа (IDA с плагином Hex-Rays)

Применение метода в проекте, выполненном в рамках госконтракта 192/11-ЭКБ-27.09ок, показало его высокую стойкость к стандартным методам анализа, так как позволяет увеличивать меру информации по Колмогорову путем настройки параметров разработанного для защиты исполняемого кода метода.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

В процессе выполнения работы были получены следующие результаты:

1. Предложена модель угроз, связанных с использованием технологии обратного проектирования.
2. Разработан метод защиты исполняемых кодов прикладного программного обеспечения, основанного на замене защищаемого кода байт-кодом виртуальной машины с псевдослучайной архитектурой.
3. Реализован алгоритм обфускации про помощи псевдослучайных ПИОК, допускающий масштабирование в процессе своей работы.
4. Разработано программное обеспечение, позволяющее реализовать защиту прикладного программного обеспечения в процессе разработки.

Дальнейшее развитие предложенных моделей, методов и алгоритмов в выбранной предметной области исследований (разработка методов и алгоритмов защиты ПО) может проводиться по следующим междисциплинарным направлениям:

- Разработка алгоритмов, способов и методов обфускации машинного кода, выполняемого в недоверенной вычислительной среде;
- Разработка методов создания исполняемого программного кода устойчивого к модификации за счет использования перекрывающихся или вложенных машинных команд виртуальных машин.
- Разработка подхода формирования защищенного исполняемого кода «на лету», таким образом, чтобы процедура, формирующая исполняемый код, изменялась при каждом новом запуске прикладной программы.

СПИСОК ПУБЛИКАЦИЙ ПО ТЕМЕ ДИССЕРТАЦИИ

1. V. Aranov, A. Terentiev, Generation of overlapped executable code/ V.Aranov/ Preliminary Proceedings of 8th Spring /Summer Young Researchers Colloquium on Software Engineering, May 29-31 2014 – Saint Petersburg, Russia ─ c. 150-154.

2. Аранов В.Ю., Заборовский В.С. Метод запутывающих преобразований программного кода при помощи сетей Петри. /В.Ю. Аранов// XLII Неделя науки СПбГПУ, Сборник лучших докладов. ─ 2014. ─ с. 129-133.

3. Аранов В.Ю., Заборовский В.С. **Разработка генератора псевдослучайных виртуальных машин/**В.Ю. Аранов// **"Инновации и инвестиции". ─** 2014. ─ №10.

4. Аранов В.Ю., Заборовский В.С. **Метод защиты от компьютерных атак, основанных на анализе исполняемого машинного кода/**В.Ю. Аранов**//Проблемы информационной безопасности. Компьютерные системы**. ─ 2013. ─ №4. ─ с. 73-81.

5. Аранов В.Ю., Заборовский В.С Обфускация машинного кода при помощи сетей Петри, /В.Ю. Аранов // XLII Неделя науки СПбГПУ, Материалы конференции. ─ 2013. ─с. 41-43.

6. Аранов В. Ю. Защита исполняемого машинного кода от обратного проектирования при помощи многоуровневой виртуализации. /В.Ю. Аранов// ИБРР-2013, Материалы конференции. ─ 2013. ─ с. 76-77.