На правах рукописи

Леошкевич Илья Олегович

СИСТЕМА ВЫЯВЛЕНИЯ НЕДЕКЛАРИРОВАННЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ВЛЕКУЩИХ НАРУШЕНИЕ КОНФИДЕНЦИАЛЬНОСТИ ИНФОРМАЦИИ

Специальность: 05.13.19 – методы и системы защиты информации, информационная безопасность

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Автор: _____

Москва – 2011

Работа выполнена в Национальном исследовательском ядерном университете «МИФИ».

Научный руководитель:	кандидат физико-математических наук, доцент Велигура Александр Николаевич
Официальные оппоненты:	доктор технических наук Конявский Валерий Аркадьевич
	кандидат технических наук Тульский Сергей Александрович
Ведущая организация:	Факультет вычислительной математики и кибернетики Московского государственного универсистета имени М.В.Ломоносова

Защита состоится 1 июня 2011 г. в 16 часов 00 минут на заседании диссертационного совета ДМ 212.130.08 при Национальном исследовательском ядерном университете «МИФИ», расположенном по адресу: 115409, г. Москва, Каширское шоссе, д. 31. Тел. для справок: +7 (495) 323-95-26, 324-84-98.

С диссертацией можно ознакомиться в библиотеке Национального исследовательского ядерного университета «МИФИ».

Отзывы и замечания по автореферату в двух экземплярах, заверенные печатью, просьба высылать по вышеуказанному адресу на имя ученого секретаря диссертационного совета.

Автореферат разослан «____» _____ 2011 г.

Ученый секретарь
диссертационного совета Горбатов В.С.

Общая характеристика работы

Актуальность работы. Обеспечение конфиденциальности информации является одной из важнейших задач в сфере информационной безопасности. Действующий стандарт ГОСТ Р ИСО/МЭК 17799-2005 определяет конфиденциальность как "обеспечение доступа к информации только авторизованным пользователям". Серьёзным источником угрозы конфиденциальности информации является наличие в обрабатывающем её программном обеспечении недекларированных возможностей (НДВ), которые определяются руководящим документом Гостехкомиссии от 4 июня 1999 года как "функциональные возможности программного обеспечения, не описанные или не соответствующие описанным в документации, при использовании которых возможно нарушение конфиденциальности, доступности или целостности обрабатываемой информации".

В данной работе рассматривается разновидность НДВ, создающих угрозу нарушения конфиденциальности информации, далее называемая НДВ-К. НДВ-К связаны с таким хранением или распространением конфиденциальной информации, которое не предусмотрено заложенными в основу обрабатывающего её программного обеспечения архитектурой и алгоритмами. НДВ-К могут возникнуть как в результате ошибки программирования, так и в результате преднамеренного внедрения программной закладки. В целях данной работы наличие или отсутствие умысла лица, вносящего НДВ-К, не имеет значения. Наличие НДВ-К может позволить злоумышленнику скомпрометировать систему защиты информации, использующую математические надёжные алгоритмы и протоколы.

Необходимость выявления НДВ-К обусловлена тем, что во многих современных операционных системах предполагается корректность программ, обрабатывающих конфиденциальную информацию от имени пользователя. Это допущение отсутствует в системах, работающих по модели Белла-Лападула, где выполняется свойство "звезды" - процессы с более высоким уровнем конфиденциальности не могут писать в области с более низким уровнем конфиденциальности. Хотя реализации модели Белла-Лападула для современных операционных систем существуют, зачастую они не используются, в том числе из соображений удобства.

Актуальность разработки новых способов выявления НДВ-К обуславливается тем, что задача выявления НДВ-К, как и любые другие задачи исчерпывающего анализа поведения программного кода, алгоритмически неразрешима. Это следует из результата, полученного Аланом Тьюрингом в 1936 году.

Проблемой выявления НДВ-К в последнее десятилетие занималось множество исследователей, из которых особо можно выделить Nicholas Nethercote

(автора утилиты Valgrind), Dawn Song (университет Беркли), Jim Chow (Стэнфордский университет), и Вартана Падаряна (Институт системного программирования РАН). Ими спроектированы и разработаны средства динамического анализа, позволяющие отслеживать зависимости между обрабатываемыми работающей программой данными, и таким образом позволяющие обнаруживать в ней НДВ-К. Однако, они не позволяют доказать отсутствие НДВ-К, так как с их помощью невозможно исследовать поведение программы при всех возможных входных данных.

Автором данной работы предлагается новая система статического анализа, предназначенная для выявления НДВ-К и работающая на уровне исполняемого кода. Статический анализ предоставляет больше возможностей по рассмотрению всех путей выполнения программы, нежели динамический, а работа на уровне исполняемого кода повышает точность за счёт того, что он является конечным результатом процесса сборки и не подлежит изменениям перед выполнением на ЭВМ конечного пользователя. Применение такой системы позволит существенно ускорить и уменьшить затраты на выявление НДВ-К.

Объектом исследования диссертационной работы является исполняемый код программного обеспечения.

Предметом исследования диссертационной работы являются НДВ, влекущие нарушение конфиденциальности информации.

Целью диссертационной работы является синтез методики выявления НДВ, влекущих нарушение конфиденциальности информации.

Решаемые задачи. Для достижения поставленной цели в диссертационной работе решаются следующие задачи:

- анализ существующих методов выявления НДВ, влекущих нарушение конфиденциальности информации;
- анализ существующих методов верификации исполняемого кода;
- синтез архитектуры системы выявления НДВ, влекущих нарушение конфиденциальности информации;
- синтез алгоритма построения формальной модели программного обеспечения по его исполняемому коду;
- реализация системы выявления НДВ, влекущих нарушение конфиденциальности информации.

Основными методами исследований, используемыми в работе, являются абстрактная интерпретация, символьное выполнение и алгоритмы на графах.

Научная новизна проведенных исследований и полученных в работе результатов заключается в следующем:

- предложен язык описания архитектур процессоров, позволяющий преобразовывать различные машинные инструкции в единое внутреннее представление;
- построены два абстрактных домена для представления адресов, позволяющие анализировать низкоуровневые идиомы работы с указателями, основывающиеся на побитовых операциях;
- построены два домена абстрактных состояний: домен простых состояний, позволяющий быстро получать возможные значения и зависимости между ячейками памяти, и домен символьных состояний, позволяющий получать точные формулы, описывающие работу фрагмента исполняемого кода;
- предложен алгоритм построения и анализа формальной модели программного обеспечения по его исполняемому коду, для достижения производительности и точности совмещающий абстрактную интерпретацию и символьное выполнение.

Практическая значимость результатов работы определяется разработанной системой выявления НДВ, влекущих нарушение конфиденциальности информации. Результаты работы представляют практическую ценность для разработчиков программного обеспечения, обрабатывающего конфиденциальную информацию.

На защиту выносятся следующие основные результаты и положения:

- предложенный язык описания архитектур процессоров и функций операционных систем, позволяющий использовать одни и те же алгоритмы анализа для различных платформ;
- построенные численные абстрактные домены: домен выровненных адресов, позволяющий точно учитывать результаты операции выравнивания указателей, и домен битовых полей, позволяющий описывать работу с адресами, в неиспользуемые биты которых помещена дополнительная информация;
- построенные домены абстрактных состояний: домен простых состояний, позволяющий эффективно проводить различные виды анализа потоков данных, и домен символьных состояний, позволяющий точно описывать поведение участков кода;

- предложенный алгоритм построения и анализа формальной модели программного обеспечения по его исполняемому коду, основывающийся на абстрактной интерпретации и уточняющий полученные результаты с помощью символьного выполнения.

Внедрение результатов работы. Результаты работы используются в ООО ИБМ Восточная Европа/Азия при разработке программного обеспечения, а также в учебном процессе кафедры №42 "Криптология и дискретная математика" Национального исследовательского ядерного университета «МИФИ».

Публикации и апробация работы. Основные положения диссертации изложены в 8 публикациях, 4 из которых в изданиях, включенных в Перечень ведущих рецензируемых изданий. Результаты работы докладывались и получили одобрения на следующих конференциях:

- XV Всероссийская научная конференция. Проблемы информационной безопасности в системе высшей школы. 2008;
- XVII Всероссийская научная конференция. Проблемы информационной безопасности в системе высшей школы. 2010;
- 12-й Национальный форум информационной безопасности. Информационная безопасность России в условиях глобального информационного общества. 2010;
- VII межрегиональная научно-техническая конференция студентов и аспирантов. Применение кибернетических методов в решении проблем общества XXI века. 2010;
- Научная сессия МИФИ. 2010.

Структура и объем диссертации. Диссертация состоит из введения, обзора литературы, 4 глав, заключения и библиографии из 101 позиции. Общий объем диссертации составляет 154 страницы, включая 19 рисунков и 8 таблиц.

Содержание работы

Во **введении** обосновывается актуальность темы диссертации, выделяются и формулируются цели и задачи исследования, описывается структурно-логическая схема диссертационной работы.

В **первой главе** определяется рассматриваемый в данной работе тип НДВ - НДВ-К, проводится анализ существующих методов выявления НДВ и анализа исполняемого кода.

Рассматриваемый вид НДВ связан с копированием и хранением конфиденциальной информации, не предусмотренным алгоритмами, заложенными в основу программного обеспечения. Выявление НДВ-К в данной работе предлагается осуществлять с помощью рассмотрения всех возможных действий по распространению информации в процессе её обработки анализируемым программным обеспечением и проверки возможности совершения хотя бы одного действия, недопустимого относительно задаваемой аналитиком политики обработки информации.

Задача выявления НДВ-К может быть формализована в терминах абстрактной вычислительной машины, построенной в соответствии с принципами архитектуры фон Неймана.

Зададим машину A как тройку (M_A, I_A, d_A), состоящую из множества состояний оперативной памяти, набора инструкций и функции декодирования. Каждый такт работы машины заключается в выполнении одной инструкции из множества $I_A \subseteq (M_A \to M_A)$. Функция декодирования $d_A \in (M_A \to I_A)$ служит для однозначного определения следующей выполняемой инструкции по содержимому оперативной памяти. Функционирование машины описывается формулой $m_{n+1,A} = d_A(m_{n,A})(m_{n,A})$, где $m_{0,A}$ - некоторое начальное состояние. В целях настоящей работы $m_{0,A}$ задаётся как $l_A(m_{*,A})$, где $l_A \in (M_A \to M_A)$ - функция загрузки проверяемого программного обеспечения в оперативную память, а $m_{*,A}$ - входные данные.

Назовём политикой обработки информации для машины A четвёрку $P_A = (C_{P_A}, t_{P_A}, k_{P_A}, M_{*,P_A})$. C_{P_A} - это множество, описывающее конфиденциальность хранящейся в оперативной памяти информации. Теневая память $M_{P_A} = (M_A \times C_{P_A})$ описывает содержимое оперативной памяти и его конфиденциальность. t_{P_A} - это преобразование, дополняющее семантику набора инструкций правилами продвижения меток конфиденциальности: $t_{P_A} \in (I_A \to I_{P_A})$, где $I_{P_A} = (M_{P_A} \to M_{P_A})$. Наконец, $k_{P_A} \in (M_{P_A} \to Boolean)$ - функция проверки допустимости состояний теневой памяти, а $M_{*,P_A} \in M_{P_A}$ - множество начальных состояний. Политика обработки информации P_A может быть использована для преобразования машины A в машину A_{P_A}, функционирующую по формуле $m_{n+1,P_A} = t_{P_A}(d_A(m_{n,A}))(m_{n,P_A})$. С учётом проверяемого программного обеспечения l_A, $m_{0,P_A} = (l_A(m_{*,A}), c_{*,A})$.

Программа l_A удовлетворяет политике обработки информации P_A, если при выполнении l_A на любых допустимых входных данных не возникнет состояния теневой памяти, отвергаемого функцией допустимости k_{P_A}:
$\forall m_{*,P_A} \in M_{*,P_A}, i \in \mathbb{N} : k_{P_A}(m_{i,P_A})$. Задача выявления НДВ-К состоит в проверке этого условия.

С такой постановкой задачи связаны два ограничения. Во-первых, определённые детали процесса обработки конфиденциальной информации, например, соответствие реализаций криптографических преобразований стандар-

там, указать в политике не представляется возможным. Поэтому ситуации, подобные использованию не предусмотренного архитектурой криптографического алгоритма, не отслеживаются. Примерами отслеживаемых НДВ являются отсутствие необходимых операций удаления конфиденциальной информации, а также её сохранение или передача в не оговорённых политикой полях файлов и сетевых соединений. Во-вторых, от аналитика требуется определённый уровень знаний о внутреннем устройстве анализируемой программы. Это объясняется тем, что по умолчанию зашифрованная информация будет считаться конфиденциальной, поскольку она зависит от конфиденциальных ключа и открытого текста. Поэтому аналитик должен указать в политике точки завершения шифрования и добавить в них команды удаления меток конфиденциальности с шифртекста.

На настоящий момент аудит безопасности программного обеспечения может проводиться вручную или автоматизированно. Ручной анализ позволяет выявить широкий спектр НДВ, однако, сам по себе крайне трудоёмок. Автоматизированный анализ предполагает наличие инструментального средства, проверяющего выполнение формально определённых правил. Такое средство формирует список потенциально нарушающих эти правила фрагментов кода и передаёт его аналитику.

Автоматизированные средства можно разделить на статические и динамические. Известными динамическими методами выявления НДВ-К являются анализ меток (taint analysis), анализ содержимого оперативной памяти, а также отслеживание системной активности, в частности, работы с диском и сетью.

В статические методы обнаружения входят поиск типовых конструкций, проверка формальных моделей и анализ потоков данных. Поиск типовых конструкций заключается в идентификации последовательностей операций, зачастую сопутствующих распространённым ошибкам. В качестве примера можно привести отсутствие перед оператором освобождения памяти оператора её очистки. Заметим, что наличие оператора очистки памяти не гарантирует корректность, так как ему может быть передан неправильный размер очищаемых данных. Кроме того, занимаемая конфиденциальной информацией память в некоторых случаях может быть вообще не освобождена. Таким образом, анализ, ориентированный только на синтаксические конструкции, позволяет выявить лишь самые очевидные ошибки. С другой стороны, он выполняется исключительно быстро и может быть включён в процесс непрерывного тестирования.

Проверка формальных моделей заключается в представлении программного кода и его свойств в терминах некоторой хорошо изученной теории. Например, код может быть преобразован в конечный автомат, а его свойства сведены к достижимости некоторых состояний. Также применяется преобра-

зование кода в логическую формулу, которая истинна только тогда, когда код обладает определёнными свойствами. После проведения такого преобразования полученная формальная модель передаётся специализированной системе. SAT-системы позволяют установить истинность логической формулы. SMT-системы, с точки зрения функционала, являются расширениями SAT-систем и позволяют использовать в формулах понятия из дополнительных теорий, например, арифметики Пресбургера. Наконец, системы доказательства теорем позволяют определять истинность утверждений в задаваемых пользователем теориях.

Анализ потоков данных представляет собой частный случай проверки формальных моделей, но традиционно считается отдельным классом методов. Анализ потоков данных работает с графом потока управления программы. Он позволяет отслеживать определённые свойства во всех её точках при любом ходе её выполнения. Для этого составляются уравнения потоков данных, вытекающие из интересующих нас свойств, начальных условий и правил перехода по рёбрам. Для решения этих уравнений может использоваться итеративный алгоритм. Во многих случаях имеется возможность составить их таким образом, чтобы гарантировать отсутствие ошибок первого или второго рода.

Специфика автоматизированного анализа исполняемого кода заключается в сложности получения его формальной модели. В исполняемом коде отсутствует информация о переменных: их местоположении, типах, областях видимости, времени жизни и правилах доступа. Память представляется в виде большого массива, доступ к любому элементу которого может быть осуществлён в любой момент времени. Полный граф потока управления изначально недоступен, а для его построения требуется оценка значений переменных, что необходимо для разрешения косвенных переходов - в исполняемом коде ими представляются такие важные даже для простейших видов анализа вещи, как вызовы виртуальных методов и возвраты из процедур.

Существует ряд систем, реализующих разновидности приведённых выше методов. К динамическим системам анализа исполняемого кода относятся Valgrind, BitBlaze TEMU, TaintBochs, TrEx, EMU и НКВД 2.5. В них имеется функционал для решения поставленной задачи, однако, им присуще принципиальное ограничение динамических средств - отсутствие гарантий полноты; поэтому видится актуальным разработка дополняющего их статического средства. К статическим системам анализа исполняемого кода относятся IDA/HexRays, CodeSurfer/x86, BitBlaze Vine, Jakstab и BoundT. В них отсутствует функционал для решения поставленной задачи, поэтому было принято решение разработать собственную систему анализа, опирающуюся на существующие наработки.

Во **второй главе** перечисляются связанные с НДВ-К угрозы конфиден-

циальности информации, строится модель нарушителя, определяются требования к системе выявления НДВ-К, и предлагается её архитектура. Из списка угроз и модели нарушителя вытекает целесообразность борьбы с НДВ-К, на этапе разработки с помощью анализа программного кода, в противовес их нейтрализации на этапе эксплуатации с помощью организационно-технических мер. Из соображений удобства использования, полноты анализа, универсальности и простоты реализации к системе предъявляются следующие требования:

1. Входными данными должны являться исполняемый модуль, возможно, с отладочной информацией, и составленная аналитиком политика обработки информации.

2. При обнаружении возможности наличия НДВ-К система должна построить понятный аналитику пример хода выполнения программы, при котором НДВ-К будет задействована.

3. Система должна выдавать заключение об отсутствии НДВ-К только в том случае, если программа полностью, то есть, при любом ходе работы, удовлетворяет политике обработки информации. Если система в ходе анализа делает допущения, невыполнение которых может привести к нарушению гарантий полноты, она обязана проинформировать об этом аналитика.

4. Работа с системой может проводиться в несколько итераций - на основании результатов работы аналитик может произвести уточнение политики и повторить проверку.

5. Система должна поддерживать множество архитектур процессоров; добавление поддержки новых архитектур не должно быть связано с принципиальными затруднениями.

6. Система должна работать непосредственно с семантикой машинных команд, а не ориентироваться на их типовые последовательности. Использование знаний о типовых последовательностях допустимо лишь для повышения производительности.

7. Система должна быть способна анализировать разновидности самомодифицирующегося кода, применяющиеся в рамках распространённых идиом программирования.

Для удовлетворения этих требований была разработана архитектура системы выявления НДВ-К, состоящая из следующих компонентов (рисунок 1):

1. **Дизассемблер** переводит машинные команды в универсальное внутреннее представление. Основными элементами этого представления являются микроинструкции присваивания и косвенного перехода, оперирующие символьными выражениями. Дизассемблирование управляется описаниями архитектур процессоров на разработанном в рамках данной работы языке.

2. **Интерфейс к системам компьютерной алгебры**. Некоторые фазы анализа оперируют символьными выражениями, точно описывающими работу фрагментов кода. Для поддержания компактности и выявления свойств таких описаний используются сторонние системы компьютерной алгебры. Предусмотрена возможность подключения произвольных систем, имеющих требуемый системой функционал. В реализованном прототипе используются Wolfram Research Mathematica и Parma Polyhedra Library.

3. **Анализатор потоков данных**. Многие задачи анализа кода могут быть сведены к задаче анализа потоков данных и решены с помощью итеративного алгоритма. Анализатор потоков данных реализует итеративный алгоритм с возможностью выбора деталей режима его работы и абстрактного домена для требуемого вида анализа.

4. **Статический анализатор** проводит комплексный анализ исполняемого кода. Он отвечает за построение графа потока управления, получения приближений значений всех переменных в каждой точке программы, а также информации о зависимостях. Статический анализатор может использовать различные домены состояний, причём сначала используются более производительные, но менее точные. Возможно подключение модулей, реализующих дополнительный функционал.

5. **Модуль выявления НДВ-К** является расширением статического анализатора. Он дополняет создаваемую статическим анализатором формальную модель программы исходя из политики обработки информации, анализирует дополненную модель, и выдаёт заключение о её соответствии политике обработки информации. Используемые при этом абстрактные домены гарантируют отсутствие ошибок второго рода.

В **третьей главе** рассматриваются аспекты внутреннего устройства компонентов системы: предлагаются новый язык описания архитектур процессоров, численные абстрактные домены и абстрактные домены для описания состояний, а также алгоритмы анализа исполняемого кода, проводится выбор технологий для их реализации.

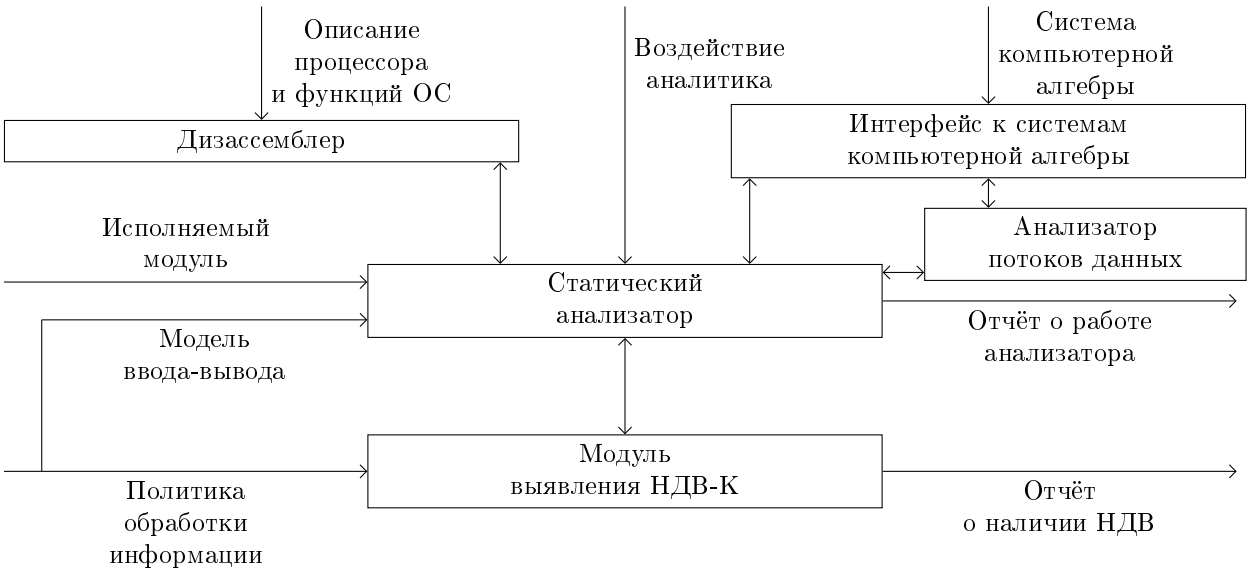

Рис. 1. Архитектура системы выявления НДВ-К

Дизассемблер. Дизассемблер переводит инструкции различных процессоров в единое внутреннее представление, каковой процесс управляется описаниями архитектур процессоров на разработанном языке. Инструкции внутреннего представления предназначены для абстрактной машины, конкретные параметры которой задаются в описаниях. В число параметров входят адресные пространства, однородно моделирующие регистры и оперативную память, а также типы данных, с которыми работает процессор.

Описания составляются с использованием символьных выражений, включающих в себя около 20 типов операций. Выражения могут содержать обращения к памяти, задаваемые с помощью четырёх значений: адресного пространства, смещения, типа и атрибутов доступа. В качестве примера можно привести выражение `UL(32) [$memory : esp * 8 : rpl -> ss_rpl] + 42`, обозначающее значение вершины стека архитектуры x86, увеличенное на 42.

В описаниях содержатся правила разбора машинных инструкций, поступающих из битового потока. В работе используются два вида битовых потоков - для доступа к массивам с заранее известным содержимым и для доступа к абстрактным состояниям. Второй вид битовых потоков используется для анализа самомодифицирующегося кода.

На рисунке 2 приведён фрагмент описания одной разновидности инструкции `not` архитектуры x86. Первый элемент - `0xF6 == 8`, обозначает, что первые 8 бит инструкции имеют значение `0xF6`. Второй элемент отвечает за разбор байта ModR/M с помощью одноимённого правила, у которого есть несколько параметров - имена правил, с помощью которых следует разбирать поля R/O и R/M, а также имена переменных, в которые следует поместить

результат. `op1 := 0xFF - op1` отвечает за генерацию микроинструкции присваивания, `mnemonic = "not"` - за помещение в переменную времени разбора с именем `mnemonic` имени распознанной инструкции, и, наконец, вызов правила `Next()` - за генерацию микрокоманды увеличения счётчика инструкций. Все именованные сущности: `ModRM`, `C3_2`, `Byte` и т.д., являются частью описания и не встроены в язык.

```
/* not r/m8: "F6 /2" */ /* Комментарий. */
0xF6 == {8};            /* Код операции. */
ModRM(                  /* Байт ModR/M. */
  RegOpcode -> "C3_2",  /* R/O = 2. */
  RegMemory -> "Reg8",  /* R/M - либо 8-битный регистр, */
  type -> Byte,         /* либо значение типа "Byte" в памяти. */
  result1 -> temp,      /* R/O сохранять не нужно. */
  result2 -> op1);      /* R/M следует занести в переменную op1. */
op1 := 0xFF - op1;      /* Результат работы: инверсия битов. */
mnemonic = "not";       /* Название. */
Next();                 /* Переход к следующей инструкции.*/
```

Рис. 2. Инструкция `not`

Правила представляют собой набор вариантов, а каждый вариант состоит из последовательности элементов: $Rule = Var^*, Var = Elem^*$. Перед использованием правила компилируются в графы разбора, для чего проводится объединение всех вариантов внутри каждого правила с помощью модификации алгоритма построения префиксных деревьев. Пример объединения вариантов показан на рисунке 3.

Во время дизассемблирования происходит спуск по дереву разбора. Если с вершиной связана операция проверки соответствия константе размера N, то из входного потока запрашивается значение размера N и происходит переход по ребру, помеченному полученным значением. Для других видов вершин соответствующее связанному с ними оператору действие - присваивание переменной времени разбора, вызов другого правила или генерация микрокода, а затем осуществляется переход по единственному исходящему ребру. Микропрограмма для инструкции `not al`, полученная с помощью приведённого выше описания, выглядит следующим образом:

```
1: al := 0xFF - al;
2: eip := eip + 2;
```

Граф потока управления. Анализируемая программа с помощью дизассемблера преобразуется во внутреннее представление, а затем в граф потока управления, над которым проводится дальнейший анализ. Каждой команде целевого процессора в графе соответствует так называемая большая

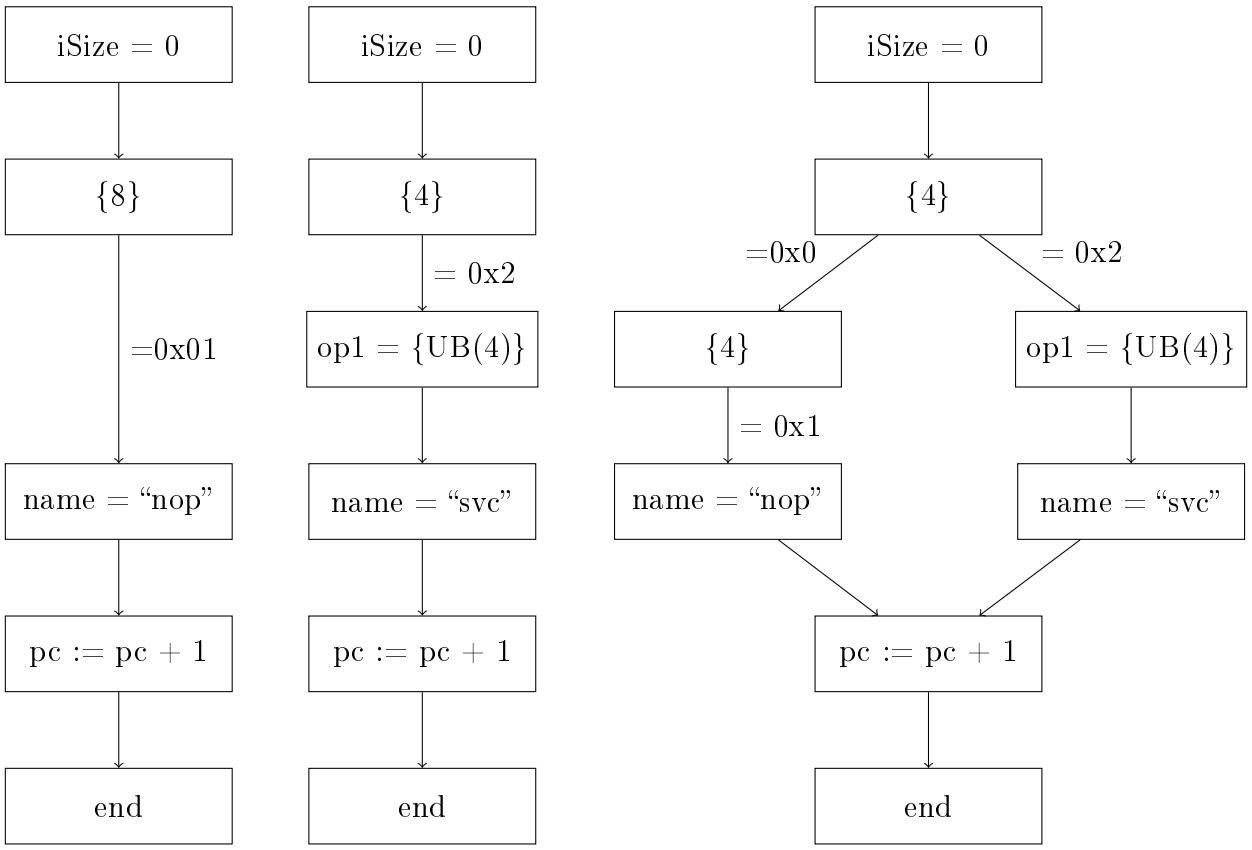

Рис. 3. Объединение вариантов

вершина; большие вершины однозначно идентифицируются своим адресом и значениями полей. Соответствующая большой вершине микропрограмма, получаемая с помощью дизассемблера, представляется в виде графа малых вершин, соответствующих индивидуальным микрокомандам. К нему добавляются две искусственные малые вершины - одна для сбора входных состояний, и одна для сбора выходных. Большие и малые вершины служат для представления программы, а непосредственно для анализа используются контекстные вершины. Контекстные вершины представляют собой малые вершины в определённом контексте, который в общем случае может быть представлен по-разному, например, строкой вызовов. Контекстные вершины, соответствующие конечным и начальным искусственным малым вершинам, связываются друг с другом рёбрами в соответствии со значениями счётчика инструкций, что позволяет получить полный граф потока управления. Пример графа приведён на рисунке 4.

Анализатор потоков данных. Многие задачи анализа кода могут быть сведены к задаче анализа потоков данных. Поэтому в основе системы анализа лежит обобщённый анализатор, решающий задачу потоков данных для произвольно фиксированных доменов с помощью итеративного алгоритма. Задачи анализа формулируются в терминах доменов абстрактных состо-

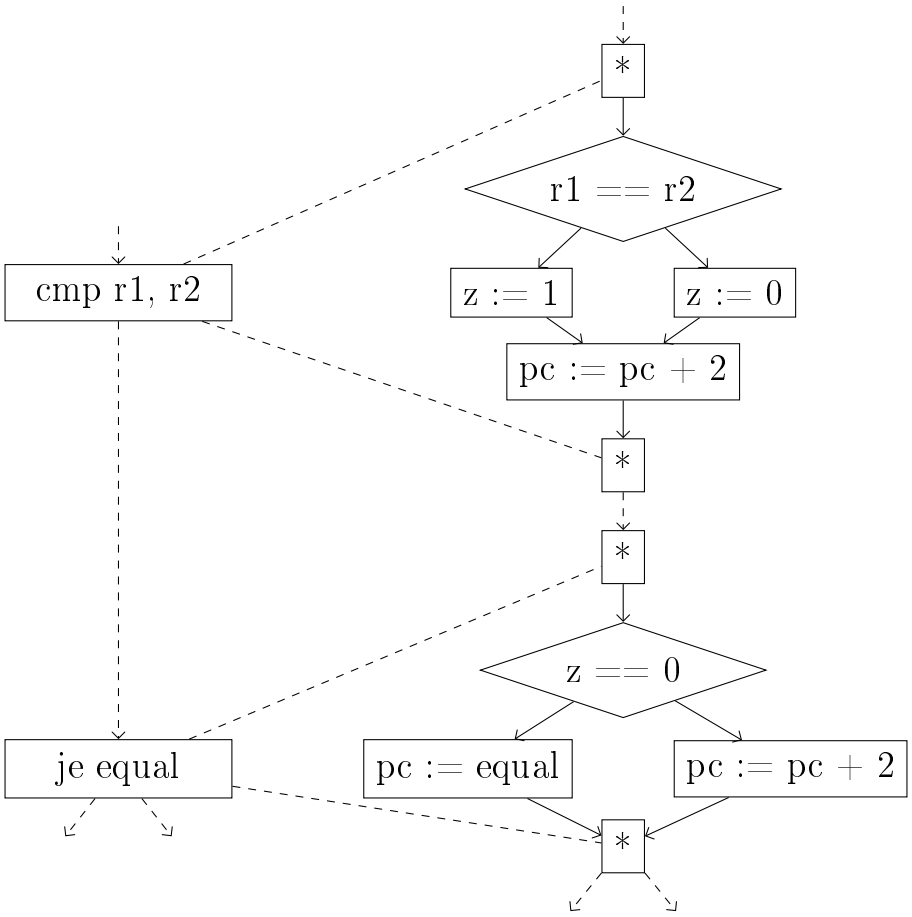

Рис. 4. Граф потока управления

яний, передаваемых анализатору. В работе используются два вида доменов состояний - основанные на регионах (домен простых состояний) и на алгебре массивов (домен символьных состояний).

В домене простых состояний адресное пространство представляет собой набор регионов - непересекающихся участков памяти, точные адреса которых заранее неизвестны. С помощью регионов моделируются статическая память, стек и куча. Регионы состоят из записей, с которыми связаны начало, конец, тип хранящегося значения и само значение. За счёт того, что на тип и значение не накладываются ограничения, этот домен может использоваться в различных видах анализа. Записи не пересекаются и упорядочены, что позволяет эффективно хранить их в сбалансированном дереве. Операции над доменом простых состояний - чтение, обновление, объединение и т.д., определены таким образом, что точно учитывают перекрывающиеся операции доступа к памяти.

Для моделирования численных значений при работе с доменом простых состояний разработано несколько численных доменов. Приведём их определе-

ния. Абстрактной суммой называется пара $b+x$, где b - база, а x - абстрактное целое. Мультисуммой называется множество абстрактных сумм с различными базами. Региональной суммой называется абстрактная сумма, в которой b представляет собой регион. Выровненным смещением называется абстрактная сумма, в которой b представляет собой тройку $-((b+x)\ mod\ 2^M)$, де M - целое число, называемое показателем выравнивания. Набором битовых полей называется сумма вида $\sum_i 2^{n_i} x_i$, где x_i - абстрактные целые, а n_i - неотрицательные целые числа. Основной численный домен, используемый для анализа, представляет собой региональную мультисумму наборов битовых полей выровненных смещений. С его помощью можно описывать выровненные указатели, указывающие на различные регионы. Использование битовых полей обусловлено необходимостью поддержки распространённых идиом адресной арифметики.

Символьные состояния представляют собой последовательности символьных присваиваний, с которыми связаны два символьных условия: предположение и условие пути. Использование анализатора потоков данных с доменом символьных состояний позволяет получать передаточные функции ациклических участков кода. Для анализа циклов в работе используется алгоритм, основанный на разбиении выполнения цикла на фазы. Для каждой фазы составляются описывающие её рекурсивные уравнения, которые решаются с помощью внешней системы компьютерной алгебры. Результаты последовательного выполнения фаз объединяются, что позволяет получить символьное состояние, описывающее весь цикл.

Политика обработки информации позволяет задавать их источники, структуру и метки конфиденциальности. Структура входных данных задаётся в виде порождающей всевозможные корректные входные данные недетерминированной “входной” микропрограммы. За установку меток конфиденциальности на входные данные отвечают размещаемые во входной микропрограмме операторы `taint`, принимающие адрес, длину и метку конфиденциальности. Выходные данные обрабатываются детерминированной “выходной” микропрограммой, целью которой является снятие меток конфиденциальности, например, с шифртекста, с помощью оператора `untaint`. Считается, что программа удовлетворяет политике, если после её завершения или любого вызова выходной микропрограммы меток конфиденциальности не остаётся.

В ходе анализа вызовы входных и выходных микропрограмм подставляются в точки открытия или закрытия сеансов связи с источниками информации. Для того, чтобы выбрать нужную микропрограмму (а для различных источников информации их может быть несколько), в политике содержится информация о их соответствии именам источников. Таким образом, анализатор может устанавливать соответствие между точкой открытия источника и самим источником с помощью информации, определяющей его имя. В каче-

стве основных источников имён можно перечислить жёстко закодированные строки, параметры командной строки, конфигурационные файлы (в случае Windows ещё и реестр) и графический интерфейс; имена одних источников могут получаться из других. Например, в политике могут содержаться утверждения вида “для файла, чьё имя получено из параметра командной строки `--key`, следует использовать входную микропрограмму с именем `Key`”.

Алгоритм анализа исполняемого кода выполняется в несколько проходов, каждый из которых состоит их фаз.

Первой фазой анализа является дополнение графа потока управления, который перед первым проходом состоит только из точки входа. В существующий граф, с вершинами которого связаны полученные на предыдущем проходе простые состояния, в соответствии с содержимым этих состояний добавляются новые вершины и рёбра. После этого проводится перерасчёт иерархии регионов - сильно связных компонент, прямых и обратных деревьев доминаторов и фронтиров доминирования, а также классификация рёбер. На всех последующих фазах обновлённый граф анализируется так, как будто он является полным.

Целью второй фазы является удаление неиспользуемых фрагментов микрокода. Было замечено, что большинство таких фрагментов представляют собой операции с регистрами и флагами, для выявления которых достаточно провести продвижение констант и, исходя из результатов продвижения, построить граф использований ячеек памяти.

Целью третьей фазы является оценка возможных значений, которые могут возникнуть в памяти при любом ходе выполнения программы. Для реализации третьей фазы в работе предложена смешанная стратегия продвижения значений, в рамках которой итеративный алгоритм применяется ко всей иерархии регионов в топологическом порядке. Каждый регион подвергается продвижению численных значений в домене простых состояний. Если регион является сильно связной компонентой, то после этого к нему применяется символьный анализ циклов. В его ходе цикл разбивается на фазы, для каждой из которых составляются и решаются описывающие её рекурсивные уравнения, а на основе их решений описывается поведение цикла в целом.

На четвёртой фазе происходит вычисление зависимостей - построение графов определения и использования ячеек памяти. Для этого используются две модификации домена простых состояний с соответствующим образом переопределёнными операциями и типами значений. При этом используется полученная на предыдущей фазе численная информация: два обращения к памяти считаются зависимыми, если затрагиваемые ими диапазоны адресов пересекаются.

Проходы продолжаются до тех пор, пока все фазы не перестанут давать новые результаты. В этом случае проверяется, присутствуют ли неразрешён-

ные переходы. Если да, то модель считается неполной и выдаётся соответствующее предупреждение. В любом случае на этом построение модели завершается.

Модуль обнаружения НДВ-К модифицирует ход выполнения алгоритма анализа в соответствии с переданной ему политикой обработки информации. В частности, он добавляет к фазе дополнения графа потока управления функциональность по подстановке входных и выходных микропрограмм. Кроме того, после завершения построения формальной модели он проводит проверку её соответствия политике, для чего используется продвижение меток. Продвижение меток представляет собой анализ потоков данных с абстрактным доменом, реализованном на основе домена простых состояний и связывающим с каждым диапазоном адресов метки конфиденциальности. Операции в этом домене отражают правила продвижения меток, в частности, `taint` и `untaint` добавляют или удаляют метку с заданных диапазонов адресов, а при присваивании выражения $expr$ ячейке памяти mem метки конфиденциальности, наложенные на mem, заменяются объединением меток конфиденциальности, наложенных на все ячейки памяти, присутствующие в $expr$. Для отслеживания зависимостей по управлению с каждой точкой программы связывается информация о переходах, которыми обусловлено попадание в неё; они могут быть вычислены как её итерированный обратный фронтир доминирования.

В **четвёртой главе** приводятся примеры применения разработанной системы.

IBM. Система была использована в ООО ИБМ Восточная Европа/Азия для верификации диагностических компонентов базовой контрольной программы операционной системы z/OS, в частности, подсистемы трассировки системных событий. Ошибки в ней могут привести не только к аварийному останову системы, но и выдаче неавторизованному пользователю конфиденциальной информации или предоставлению ему полного контроля над системой.

На основе рекомендаций IBM по созданию безопасного программного обеспечения для z/Architecture был сформирован перечень ошибок, подлежащих автоматизированному поиску. В него входят хранение конфиденциальной информации в “пользовательской” памяти и запись в “пользовательскую” память без понижения текущего уровня привилегий.

Было проанализировано 15 модулей общим объёмом 130 тыс. строк. Были обнаружено 2 ошибки (отражённые в таблице 1), заключающиеся в записи в “пользовательскую” память без понижения уровня привилегий.

Кафедра №42. Система была использована в учебном процессе кафедры №42 “Криптология и дискретная математика” Национального исследовательского ядерного университета «МИФИ» в рамках курсов “Информатика”

Фрагмент GTF	Размер (тыс. строк)	НДВ найдено (шт.)
LINKLIB	20	0
LPALIB	95	2
Макросы	15	0
Всего	130	2

Таблица 1. Результаты верификации подсистемы трассировки событий

- для поиска распространённых ошибок в результатах практических работы, и “Защита программного обеспечения” - в качестве инструмента для отслеживания распространения информации учебными образцами, а также для обучения противодействию статическому анализу.

В рамках курса “Информатика” студенты выполняют практические работы, заключающиеся в написании программ на языках Си и Ассемблер (x86). Чтобы помочь студентам лучше подготовиться к сдаче, была реализована Web-система предварительной сдачи на основе пакета ORZ Online Judge. Студент посылает код через Web-интерфейс, а система выполняет предопределённые тесты и вызывает разработанную систему для обнаружения потенциальных ошибок работы с памятью. Функционал по продвижению меток конфиденциальности в данном случае не используется. Собранная статистика работы отражена в таблице 2.

Задание	Средний размер (строк)	Среднее время (сек.)	Число образцов	Ошибок найдено
3.5	100	54	113	0
4.2	150	72	97	5
5.3	500	251	171	3
6.2	150	31	53	7
6.3	200	107	92	4

Таблица 2. Результаты автоматических проверок практических заданий

Курс “Защита программного обеспечения” посвящён изучению защит программного обеспечения от несанкционированного копирования. В рамках курса студентам предлагается как исследовать существующие защиты, так и разрабатывать свои собственные. Разработанная система используется в одной из лабораторных работ для анализа распространения вводимой пользователем регистрационной информации.

В **заключении** обобщаются результаты проделанной работы

Основные выводы и результаты работы

1. Проведён анализ существующих методов и средств выявления НДВ, влекущих нарушение конфиденциальности информации. Сделан вывод, что выявление НДВ, влекущих нарушение конфиденциальности информации, должно проводиться на уровне исполняемого кода. Также сделан вывод о том, что существующие средства нуждаются в усовершенствовании. С помощью существующих динамических средств невозможно исследовать поведение программы при всех возможных входных данных, что является необходимой предпосылкой для доказательства отсутствия НДВ. Существующие статические средства анализа исполняемого кода не имеют функционала, позволяющего выявлять НДВ, влекущие нарушение конфиденциальности информации.

2. Разработаны требования к системе выявления НДВ, влекущих нарушение конфиденциальности информации. Для их удовлетворения предложена архитектура системы выявления НДВ, влекущих нарушение конфиденциальности информации; спроектированы её основные компоненты - дизассемблер, интерфейс к системам компьютерной алгебры, анализатор потоков данных, статический анализатор и модуль выявления НДВ, влекущих нарушение конфиденциальности информации, а также связи между ними.

3. Разработан алгоритм построения формальной модели программного обеспечения по его исполняемому коду и выявления с её помощью НДВ, влекущих нарушение конфиденциальности информации, сочетающий абстрактную интерпретацию и символьное выполнение.

4. Для проведения абстрактной интерпретации разработаны численные абстрактные домены выровненных адресов и битовых полей, предназначенные для точного учёта эффектов побитовых операций над указателями, а также домены простых состояний и символьных состояний, соответственно для поверхностного и точного анализа поведения исполняемого кода.

5. Разработано программное обеспечение для выявления НДВ, влекущих нарушение конфиденциальности информации.

6. Разработанное программное обеспечение успешно применено для анализа диагностических компонентов базовой контрольной программы операционной системы z/OS, а также в учебном процессе в рамках курсов “Информатика” и “Защита программного обеспечения”.

Основные публикации по теме диссертации

- **Леошкевич, И.О.** Поиск уязвимостей в программном обеспечении с помощью анализа исполняемого кода / И.О. Леошкевич // XV Всероссийская научная конференция. Проблемы информационной безопасности в системе высшей школы — 2008. — С. 93–94

- **Леошкевич, И.О.** Анализ методов идентификации структур данных в исполняемом коде / И.О. Леошкевич, А.Ю. Тихонов // Безопасность информационных технологий — 2009. — №2. — С. 94–97

- **Леошкевич, И.О.** Получение архитектурно-независимой семантики исполняемого кода / И.О. Леошкевич // Безопасность информационных технологий — 2009. — №4. — С. 120–124

- **Леошкевич, И.О.** Разработка системы автоматического анализа исполняемого кода / И.О. Леошкевич // XVII Всероссийская научная конференция. Проблемы информационной безопасности в системе высшей школы — 2010. — с. 158

- **Леошкевич, И.О.** Анализ зависимостей в исполняемом коде / И.О. Леошкевич // Труды научной сессии МИФИ — 2010. — С. 208–211

- **Леошкевич, И.О.** Поиск утечек информации в программном обеспечении с помощью анализа исполняемого кода / И.О. Леошкевич // Безопасность информационных технологий — 2010. — №1. — С. 89–91

- **Леошкевич, И.О.** Система для семантического анализа исполняемого кода / И.О. Леошкевич // VIII межрегиональная научно-техническая конференция студентов и аспирантов. Применение кибернетических методов в решении проблем общества XXI века. Тезисы докладов. — 2010. — С. 9–11

- **Леошкевич, И.О.** Низкоуровневая политика обработки данных / И.О. Леошкевич, А.Н. Велигура // Безопасность информационных технологий — 2010. — №3. — С. 53–55