На правах рукописи

БРЕЧКА ДЕНИС МИХАЙЛОВИЧ

РАЗРАБОТКА АЛГОРИТМОВ ПРОВЕРКИ ОТСУТСТВИЯ НЕСАНКЦИОНИРОВАННЫХ ДОСТУПОВ В КОМПЬЮТЕРНЫХ СИСТЕМАХ НА ОСНОВЕ ДИСКРЕЦИОННЫХ МОДЕЛЕЙ БЕЗОПАСНОСТИ

Специальность:

05.13.19 – Методы и системы защиты информации, информационная безопасность

АВТОРЕФЕРАТ

диссертации на соискание ученой степени кандидата технических наук

Омск-2011

Работа выполнена в Омском государственном университете
им. Ф.М.Достоевского.

Научный руководитель: доктор физико-математических наук, доцент
Белим Сергей Викторович

Официальные оппоненты: доктор технических наук, профессор
Зикратов Игорь Алексеевич

кандидат технических наук
Прохожев Николай Николаевич

Ведущая организация: Сибирский государственный аэрокосмический университет имени академика М.Ф. Решетнева

Защита состоится 29.03.2011 на заседании диссертационного совета Д 212.227.05 в 15-50 по адресу: 197101, Санкт-Петербург, пр. Кронверкский, д.49., СПбГУ ИТМО, ауд. 403.

С диссертацией можно ознакомиться в библиотеке Санкт-Петербургского государственного университета информационных технологий, механики и оптики (национальный исследовательский университет).

Автореферат разослан 28 февраля 2011 г.

Ученый секретарь диссертационного
совета Д 212.227.05 В.И. Поляков

АКТУАЛЬНОСТЬ ПРОБЛЕМЫ

Проблема выявления возможных каналов несанкционированного доступа к данным является одной из трех составляющих информационной безопасности компьютерных систем, отвечающей за конфиденциальность информации. Формальный анализ компьютерных систем с целью выявления возможных несанкционированных доступов принято проводить на основе моделей безопасности. Дискреционные модели безопасности, обеспечивающие произвольное разграничение доступа, являются основой для построения защищенных компьютерных систем[1,3,4]. Наличие произвольного разграничения доступа является минимальным требованием к автоматизированным системам обработки информации для того, чтобы они могли считаться защищенным согласно всем действующим стандартам. Дискреционная модель безопасности подразумевает возможность для администратора безопасности произвольным образом выдавать разрешение на доступ пользователей к объектам компьютерной системы.

Модель Харрисона-Руззо-Ульмана (Harrison-Ruzzo-Ulman, HRU) [1,7,10] была одной из первых моделей, ориентированных на анализ дискреционного доступа, и большинство более поздних моделей основаны на ее базовых положениях. Одним из основных результатов модели HRU является алгоритмическая неразрешимость задачи проверки произвольной системы на наличие каналов утечки информации вследствие несанкционированного доступа. В связи с этим в работах [9,10] предложен ряд ограничений модели, позволяющих выявить системы, для которых можно гарантировать защищенность. К гарантированно защищенным относятся монооперационные и моноусловные системы [1]. При этом существенно ограничивается функциональность компьютерной системы.

Модель Take-Grant [1,7,11] более предсказуема, чем модель HRU. Для модели Take-Grant сформулированы две теоремы, содержащие условия, при которых в системе гарантированно не произойдет утечек информации. Однако способы проверки выполнимости условий теорем не определены.

Целью диссертационной работы является развитие моделей дискреционного разграничения доступа HRU и Take-Grant с целью выявления новых классов безопасных компьютерных систем, а также построение алгоритмов проверки безопасности состояния компьютерной системы.

Методы исследования. При решении поставленных задач использовались математические модели, теория графов, теория алгоритмов, элементы булевой алгебры.

Научная новизна результатов исследований.

1. Впервые применен базисный подход для анализа безопасности модели HRU, что позволило расширить класс систем гарантированно защищенных от несанкционированного доступа.
2. Проведен анализ подсистемы безопасности операционных систем семейства Windows на возможность возникновения несанкционированных доступов с использованием дискреционных моделей безопасности.
3. Построены полиномиальные алгоритмы проверки возможности несанкционированного доступа заданного субъекта к заданному объекту.

Достоверность результатов работы. Научные результаты диссертационной работы получены с использованием субъектно-объектного подхода к

моделированию компьютерных систем, который является на сегодняшний день общепринятым и хорошо зарекомендовавшим себя при исследовании безопасности компьютерных систем.

Практическая ценность заключается в разработке алгоритмов проверки безопасности компьютерных систем и выявлении безопасных состояний, которые могут найти применение в программных реализациях средств предотвращения несанкционированного доступа в компьютерных системах.

Основные положения, выносимые на защиту:

1. Возможно расширение дискреционной модели безопасности HRU за счет введения различных операций преобразования матриц доступа.
2. На множестве операций преобразования матрицы доступа возможно выделение базисного набора операций, через который линейно выражаются все остальные операции.
3. Рассмотрение компьютерных систем в различных базисах позволяет расширить класс систем, для которых допустимо доказательство гарантированной защищенности от несанкционированных доступов.
4. Алгоритмы проверки отсутствия несанкционированных доступов могут быть построены на основе алгоритмов на графах, поиска в ширину и поиска в глубину.
5. Проверка отсутствия каналов несанкционированного доступа между заданными субъектом и объектом может быть осуществлена за полиномиальное время.

Апробация работы. Основные результаты диссертации докладывались и обсуждались на следующих конференциях: Межвузовская научно-практическая конференция «Информационные технологии автоматизации и управления» (Омск, 2009); XIII международная научно-практическая конференция «Решетневские чтения» (Красноярск, 2009); Вторая международная научно-практическая конференция «Современные проблемы гуманитарных и естественных наук» (Москва, 2010), «Научное творчество XXI века» (Красноярск 2010); Вторая международная научно-практическая конференция «Наука в современном мире» (Москва 2010); Сибирская научная школа-семинар с международным участием «Компьютерная безопасность и криптография» - SIBECRYPT'10 (Тюмень 2010), а также на научных семинарах Омского государственного университета им Ф.М. Достоевского.

Публикации. По теме диссертации опубликованы 14 научных работ, в том числе две статьи в журналах из списка, рекомендованного ВАК.

Структура и объем работы.

Диссертационная работа состоит из введения, трех глав, заключения и списка литературы. Работа содержит 116 страниц основного текста, 33 рисунка и 5 таблиц. Список литературы включает 80 наименований.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Во введении обоснована актуальность выбранной темы диссертационной работы и сформулированы основные цели исследования.

В первой главе, носящей обзорный характер, рассматриваются модели политик безопасности компьютерных систем с дискреционным разделением доступа и примеры их применения в современных информационных системах. Приводится описание моделей Харрисона-Руззо-Ульмана (HRU) и Take-Grant.

Вторая глава посвящена развитию базисного подхода в модели HRU и выявлению новых классов безопасности на его основе. Безопасными считаются, системы, в которых невозможен несанкционированный доступ.

Дискреционное разделение доступа в компьютерных системах принято задавать с помощью матрицы доступов. Строки матрицы доступов соответствуют субъектам компьютерной системы, столбцы — объектам компьютерной системы, а ячейки определяют набор разрешенных видов доступа для соответствующей пары субъект-объект компьютерной системы. Построим более строгую модель матрицы доступов. Будем считать, что все типы доступов в компьютерной системе определяются конечным множеством $\boldsymbol{A}$ ($|\boldsymbol{A}|<\infty$). Данное предположение выполняется во всех реальных компьютерных системах. Введем множество всех возможных доступов $\boldsymbol{R}$, как комбинации из типов доступа ($\boldsymbol{R}=2^{\boldsymbol{A}}$). Пусть множество $\boldsymbol{A}$ записывается в виде $\{a_1,a_2,...,a_n\}$. Выберем для элементов множества $\boldsymbol{R}$ представление в виде n-битной строки $r=(b_1,b_2,...,b_n)$. Причем $b_i=1$, если тип доступа a_i входит в доступ r и $b_i=0$ в противном случае. Предполагается, что множество субъектов и множество объектов компьютерной системы являются счетными, то есть может быть проведена нумерация всех возможных объектов и субъектов. Определим множество матриц над $\boldsymbol{A}$ с количеством строк не превышающим количество столбцов, которое в дальнейшем будем обозначать через $\boldsymbol{M}$. Очевидно, что каждой матрице доступов можно сопоставить матрицу из множества $\boldsymbol{M}$. Верно и обратное утверждение, в силу того, что множество объектов включает в себя множество субъектов, и, следовательно, для каждой рассматриваемой матрицы можно построить реализацию компьютерной системы. Изменение защиты компьютерной системы сводится к преобразованию матрицы доступов. Такие преобразования можно записать в виде операций на множестве $\boldsymbol{M}$. Будем считать, что преобразования также зависят и от некоторого набора параметров. Множество всех возможных наборов значений параметров обозначим через $\boldsymbol{P}$. Тогда любое преобразование матрицы доступов запишется в виде операции $op:\boldsymbol{M}\times\boldsymbol{P}\to\boldsymbol{M}$.

Определим суперпозицию операций $op_1\bullet op_2\bullet...\bullet op_n$ как их последовательное применение к некоторой матрице доступов $M\in\boldsymbol{M}$.

Базисом над $\boldsymbol{M}$ будем называть минимальный набор операций, через суперпозицию которых может быть определена любая операция на множестве $\boldsymbol{M}$. Рассмотрим следующий базис над множеством $\boldsymbol{M}$ ($\boldsymbol{S}$ – множество субъектов компьютерной системы, $\boldsymbol{O}$ –множество объектов компьютерной системы, $M[S,O]$ – элемент матрицы доступов, для субъекта $S\in\boldsymbol{S}$ и объекта $O\in\boldsymbol{O}$), определяющий преобразования матрицы доступов M**:**

1. *AddO(O,M)* – добавить столбец, соответствующий объекту O в M.
2. *DelO(O,M)* – удалить столбец, соответствующий объекту O в M.
3. *AddS(S,M)* – добавить строку и столбец, соответствующие субъекту S в M.
4. *DelS(S,M)* – удалить строку и столбец, соответствующие субъекту S в M.
5. *Inv(M*[*S,O*]*,k)* – инвертировать в элементе $M[S,O]$ бит с номером k. Эта операция выдает или отменяет право доступа a_k.

Набор операций { *AddO(O,M), DelO(O,M), AddS(S,M), DelS(S,M)*, *Inv(M[S,O],k)* } в дальнейшем будем обозначать через B_1. Из операций могут составляться команды, выполняющие несколько действий.

Теорема 1. Набор операций B_1 является базисом над множеством матриц доступа $\boldsymbol{M}$.
Выразим примитивные операторы HRU через базис B_1:
1. *Enter* a_k *into* $M[S,O]$ – внести право a_k в ячейку $M[S,O]$:
Enter a_k *into* $M[S,O]= Inv(M[S,O],k)$.
2. *Delete* a_k *from* $M[S,O]$– удалить право a_k из ячейки $M[S,O]$:
Delete a_k *from* $M[S,O]= Inv(M[S,O],k)$.
3.*Create subject* S – создать субъект S (т. е. новую строку матрицы M):
Create subject $S= AddS(S,M)$.
4. *Create object* O – создать объект O (т. е. новый столбец матрицы M):
Create object $O= AddO(O,M)$.
5. *Destroy subject* S – уничтожить субъект S: *Destroy subject* $S= DelS(S,M)$.
6. *Destroy object* O – уничтожить объект O: *Destroy object* $O= DelO(O,M)$.

Первый и второй примитивный оператор HRU представляются одинаково через операцию $Inv(M[S,O],k)$, однако условие выполнения у них разное. В первом случае право a_k первоначально отсутствует в ячейке матрицы доступов, а во втором случае наоборот присутствует.

Как видим, примитивных операторов HRU на один больше чем базисных операций, следовательно, они не являются базисом. Построим базис на основе примитивных операторов HRU. Введем набор операций $B_H=\{$*Enter* a_k *into* $M[S,O]$, *Create subject* S, *Create object* O, *Destroy subject* S, *Destroy object* $O\}$.

Теорема 2. Набор операций B_H является базисом модели HRU.

В рамках модели HRU система называется безопасной относительно права a_k, если для заданного начального состояния Q_0 не существует последовательности команд, в результате которой право a_k будет занесено в ячейку матрицы доступов M, в которой оно отсутствовало в начальном состоянии Q_0, то есть отсутствуют несанкционированные доступы. Как показано в работах [2,4], задача проверки данного критерия на истинность для произвольной системы алгоритмически неразрешима. Однако можно выделить отдельные классы систем, для которых возможно построение алгоритма, проверяющего безопасность системы. Важным случаем являются монооперационные системы. Система называется монооперационной, если каждая команда данной системы выполняет один примитивный оператор HRU.

Для выявления новых безопасных систем расширим понятие монооперационной системы, с учетом существования базисного набора операций. Монооперационной системой в базисе B будем называть систему, каждая команда которой содержит ровно один оператор из базиса B.

Теорема 3. Существует алгоритм, проверяющий: является ли исходное состояние монооперационной системы в базисе B безопасным по отношению к праву a_k.

Рассмотрим возможность применения базисного подхода в модели HRU для описания механизма дискреционного разделения доступа подсистемы безопасности ОС Windows. В рамках подсистемы безопасности ОС Windows, элементарные операции задаются стандартными и специальными правами доступа. Кроме того определены общие права доступа, являющиеся комбинацией стандартных и специальных прав, то есть командами. Для записи конкретной команды, соответствующей некоторому общему праву доступа, будем рассматривать объект операционной системы как совокупность объектов. Такое рассмотрение допустимо в рамках субъектно-объектного подхода, который требует, чтобы конкатенация двух объектов также была объ-

ектом. Для обозначения, какой-то части объекта будем указывать общий идентификатор объекта и идентификатор соответствующей части, разделенные точкой. Например, содержимое файла *F* будет иметь обозначение *F.Data*. При работе с файлом важными структурами являются его заголовок *F.Header*, дескриптор безопасности *F.DS* и списки контроля доступа *F.DACL*, *F.SACL*. Выпишем команды соответствующие общим правам доступа к файлу *F* процессом *S* в базисе B_1.

1. Чтение из файла:
   command Read_ File((S,F))
   Enter(r,M[S,F.Header]);
   Enter(r,M[S,F.Data]);
   Enter(r,M[S,F.DS]);
   Enter(r,M[S,F.DACL]);
   Enter(r,M[S,F.SACL]);
   end.
2. Запись в файл:
   command Write_ File((S,F))
   Enter(w,M[S,F.Header]);
   Enter(w,M[S,F.Data]);
   Enter(w,M[S,F.DS]);
   Enter(r,M[S,F.DACL]);
   Enter(w,M[S,F.SACL]);
   end.
3. Запуск исполняемого файла :
   command Execute_ File((S,F))
   Enter(r,M[S,F.Header]);
   Enter(x,M[S,F.Data]);
   Enter(r,M[S,F.DS]);
   Enter(r,M[S,F.DACL]);
   Enter(r,M[S,F.SACL]);
   end.
4. Полный доступ к файлу:
   command All_ File((S,F))
   Enter(r,M[S,F.Header]);
   Enter(w,M[S,F.Header]);
   Enter(r,M[S,F.Data]);
   Enter(w,M[S,F.Data]);
   Enter(x,M[S,F.Data]);
   Enter(r,M[S,F.DS]);
   Enter(w,M[S,F.DS]);
   Enter(r,M[S,F.DACL]);
   Enter(r,M[S,F.SACL]);
   Enter(w,M[S,F.SACL]);
   end.

Как следует из определений понятий безопасности и монооперационности системы, проверка безопасности состояния должна осуществляться после каждого действия в системе, что накладывает ограничения на подсистему аудита. Как легко увидеть из приведенного выше, если в рамках ОС Windows ограничится только стандартными или специальными правами, то система будет монооперационной и, как следствие, безопасной. Наличие общих прав доступа делает систему не монооперационной и вопрос о ее безопасности остается открытым.

Для исследования безопасности механизма дискреционного разделения доступа в ОС Windows применим базисный подход. Для этого рассмотрим представление прав доступа на битовом уровне с помощью маски доступов. Запрос субъекта на конкретный вид доступа к объекту преобразуется в маску доступа, которая сравнивается с масками разрешенных и запрещенных доступов в элементах частного списка контроля доступов (DACL) объекта [6,11]. Маска доступа, содержащаяся в элементе DACL, представляет собой значение длиной 32 бита. Первые 16 битов определяют специальные права доступа, биты с 16 до 23 - стандартные права доступа, бит 24 - право ACCESS_ SYSTEM_ SECURITY, бит 25 - право MAXIMUM_ALLOWED (полный доступ), биты 26 и 27 зарезервированы для дальнейшего использования, биты с 28 по 31 определяют общие права доступа, отображаемые в специальные и стандартные права при попытке доступа к объекту.

Приведем специальные, стандартные и общие права доступа и их значения, и покажем, каким образом эти права могут быть представлены в базисе B_I.

Специальные права доступа: чтение данных (0x00000001), запись данных (0x00000002), добавление данных (0x00000004), чтение расширенных атрибутов (0x00000008), запись расширенных атрибутов (0x00000010), исполнение (0x00000020), удаление (0x00000040), чтение атрибутов (0x00000080), запись атрибутов (0x00000100).

Стандартные права доступа: удаление объекта (0x00010000), чтение DS объекта (0x00020000), чтение DACL объекта (0x00040000), изменение права собственности объекта (0x00080000), использование объекта для синхронизации (0x00100000).

Общие права доступа: общее право на полный доступ (0x10000000), общее право исполнения (0x20000000), общее право записи (0x40000000), общее право чтения (0x80000000).

Для того чтобы добавить или удалить какое-либо право доступа, достаточно инвертировать соответствующий этому праву бит. Ниже для примера приведена команда добавления общего права исполнения на объект *F*.

command Generic_ Execute(F)
if (b_{29}*==FALSE)*
Inv(F,29)
end.

Команды удаления прав будут выглядеть аналогично, но условие выполнения команды изменится, например *if* (b_0*==FALSE)* – добавление права, *if* (b_0*==TRUE)* – удаление права. Таким образом, при переходе к базису B_I, как видно из приведенного выше, ОС Windows, будет монооперационной системой в данном базисе.

Для применения описанного подхода к реальной системе под управлением Windows необходимо явным образом построить матрицу доступов. Для построения матрицы доступов необходимо выполнить рекурсивный обход файловой системы и проанализировать дескриптор безопасности каждого объекта с целью выявления установленных прав доступа. При анализе дескриптора безопасности нас будут интересовать списки контроля доступа (DACL), содержащиеся в нем. DACL состоит из элементов (ACE), которые непосредственно содержат идентификатор субъекта (SID) и маску доступа, для этого субъекта. Маска доступа в ACE имеет вид битовой строки, потому, операции изменения матрицы доступов могут быть представлены в виде операций с битовой строкой. Следует учитывать, что существуют различные типы ACE, нас в данном случае интересуют разрешающие и запрещающие ACE. Разрешающие ACE образуют «белый список» доступов, запрещающие — «черный список» доступов. Тип ACE следует учитывать при построении матрицы доступов. Так, если объект имеет разрешающий ACE, то этот ACE копируется в соответствующую ячейку матрицы, если объект имеет запрещающий ACE, в матрицу помещается инвертированная копия данного ACE. Если объект имеет запрещающий и разрешающий ACE,то следует выполнить операцию побитовое «ИЛИ» над этими списками, результат операции инвертировать и поместить в матрицу доступов, так как запрещающий список имеет больший приоритет, чем разрешающий.

Таким образом, при рекурсивном обходе файловой системы для каждого нового объекта будет создаваться столбец матрицы доступов. При анализе дескриптора безопасности объекта будут создаваться строки матрицы, и заполняться ее ячейки.

После того, как матрица доступов будет построена, необходимо разработать систему аудита, отслеживающую изменения матрицы. Будем описывать изменение матрицы доступов с помощью команд модели HRU. Как было показано выше, команды Windows могут быть записаны в базисе B_1 , при этом каждая команда содержит не более одной операции из набора B_1. По определению такая система будет монооперационной в базисе B_1.

Для практической реализации безопасной системы в Windows необходимо предусмотреть дополнительную программу-монитор, которая будет отслеживать все изменения дескрипторов безопасности объектов, и отражать эти изменения в матрице доступов. Программа-монитор должна отслеживать вызовы функций, изменяющих дескриптор безопасности, и их параметры. Также необходимо дополнить систему аудита, которая будет фиксировать действия пользователей в системе в базисе B_1.

В третьей главе описываются алгоритмы поиска безопасных состояний компьютерной системы, описываемой моделью Take-Grant. В модели Take-Grant исследуется передача прав доступа между субъектами, для чего вводятся два дополнительных права доступа: *t (Take)* - дает возможность брать права на объект у другого объекта и *g (Grant)* - дает возможность передавать свои права другим субъектам. Состояние системы описывается с помощью графа доступов, вершинами которого служат объекты и субъекты системы, а ориентированные помеченные дуги соответствуют правам доступа. В данной модели центральную роль играет предикат *«возможен доступ»(α, x, y, G_0),* который истинен в случае, когда субъект *x* может получить право *α* на объект *y* в системе описываемой графом доступов G_0. В рамках модели могут быть доказаны две теоремы [8], которые дают необходимые и достаточные условия для истинности предиката *возможен доступ»(α, x, y, G_0)*:

1. Предикат *«возможен доступ»(α, x, y, G_0)* истинен для графа G_0, который содержит только вершины-субъекты, если в графе существует *tg*-путь между этими субъектами. *tg*-путем называется путь в графе доступов, каждая дуга которого помечена правом *t* или *g*. При этом направление дуг не учитывается.

2. Для того чтобы предикат *«возможен доступ»(α, x, y, G_0)* был истинным для произвольного графа доступов G_0, необходимо, чтобы *x* и *y* были соединены в G_0 начальным и конечным пролетом моста с островами (подграфами из субъектов, соединенных *tg*-путями), а сами острова в графе были соединены мостами. Мостами, начальными и конечными пролетами мостами принято называть особые виды *tg*-путей.

Таким образом, исследование возможных каналов несанкционированного доступа сводится к задаче поиска в графе доступов *tg*-путей, островов, мостов, начальных и конечных пролетов мостов.

Для поиска *tg*-путей в графе доступов G_0 построим граф G_0' следующим образом:

1) множество вершин графа G_0' совпадает с множеством вершин графа G_0 ($V_0' = V_0$);

2) ребра в графе G_0' не ориентированы, множество ребер графа G_0' включает лишь те ребра из G_0, которые содержат права *t* или *g* ($E_0 = E_0 \setminus E_0^{\alpha}$, где E_0^{α} - множество ребер графа G_0, не содержащих прав *t* или *g*);

3)все ребра графа G_0' имеют одинаковый вес.

Для графа G_0' применим алгоритм Дейкстры для поиска кратчайшего пути в графе [5]. Кратчайший путь между двумя указанными вершинами, найденный алгоритмом Дейкстры в графе G_0', будет являться *tg*-путем между этими вершинами в графе G_0. Трудоемкость такого алгоритма поиска *tg*-путей $O(N^2)$, где N – количество объектов системы.

Для поиска островов в графе будем использовать алгоритм Флойда для поиска всех кратчайших путей в графе [5]. Построим граф G_0^* по исходному графу доступов G_0 следующим образом:

1) множество вершин графа G_0^* совпадает с множеством вершин графа G_0 ($V_0^* = V_0$);

2) ребра в графе G_0^* не ориентированы, множество ребер графа G_0^* включает лишь те ребра из G_0, для которых началом и концом дуги является вершина-субъект и которые содержат права *Take* или *Grant* ($E_0^* = E_0 \setminus E_0^{\alpha} \cup E_0^{s-o}$, где E_0^{s-o} - множество ребер графа G_0, для которых начало и конец не являются субъектами; E_0^{α} - множество ребер графа G_0, не содержащих прав *Take* или *Grant*);

3) все ребра графа G_0^* имеют одинаковый вес.

Для графа G_0^* применим алгоритм Флойда. Алгоритм обнаружит кратчайшие пути между каждой парой вершин в графе G_0^*. Кратчайшие пути в графе G_0^* будут *tg*-путями, проходящими через вершины-субъекты в графе G_0. А сами вершины-субъекты, объединенные *tg*-путями, будут являться островами в графе G_0. Трудоемкость такого алгоритма поиска островов $O(N^3)$.

Поиск мостов, а также начальных и конечных пролетов моста сводится к поиску *tg*-путей заданного вида. Данная задача может быть решена с использованием алгоритмов-распознавателей грамматик.

Построим распознаватель мостов в графе доступов. При этом мост $\vec{t}^*$ будем рассматривать как частный случай моста $\vec{t}^*\vec{g}\vec{t}^*$. То есть для поиска этих двух типов мостов будем использовать один алгоритм.

Введем ряд обозначений. Начальную и конечную вершину моста будем обозначать *S(e)* и *F(e)* соответственно. Вершину, в которой мы находимся на данный момент, будем обозначать *C(e)*, а рассматриваемую вершину – *W(e)*. Дугу между вершинами e_1 и e_2, содержащее право $\vec{g}$ обозначим $\vec{g}(e_1, e_2)$. Для дуг с правами $\vec{t}$ и $\overleftarrow{t}$ также введем соответствующие обозначения $\vec{t}(e_1, e_2)$ и $\overleftarrow{t}(e_1, e_2)$. Множество всех объектов обозначим через *O*.

Графическая схема работы распознавателя представлена на рисунке 1. Опишем состояния распознавателя.

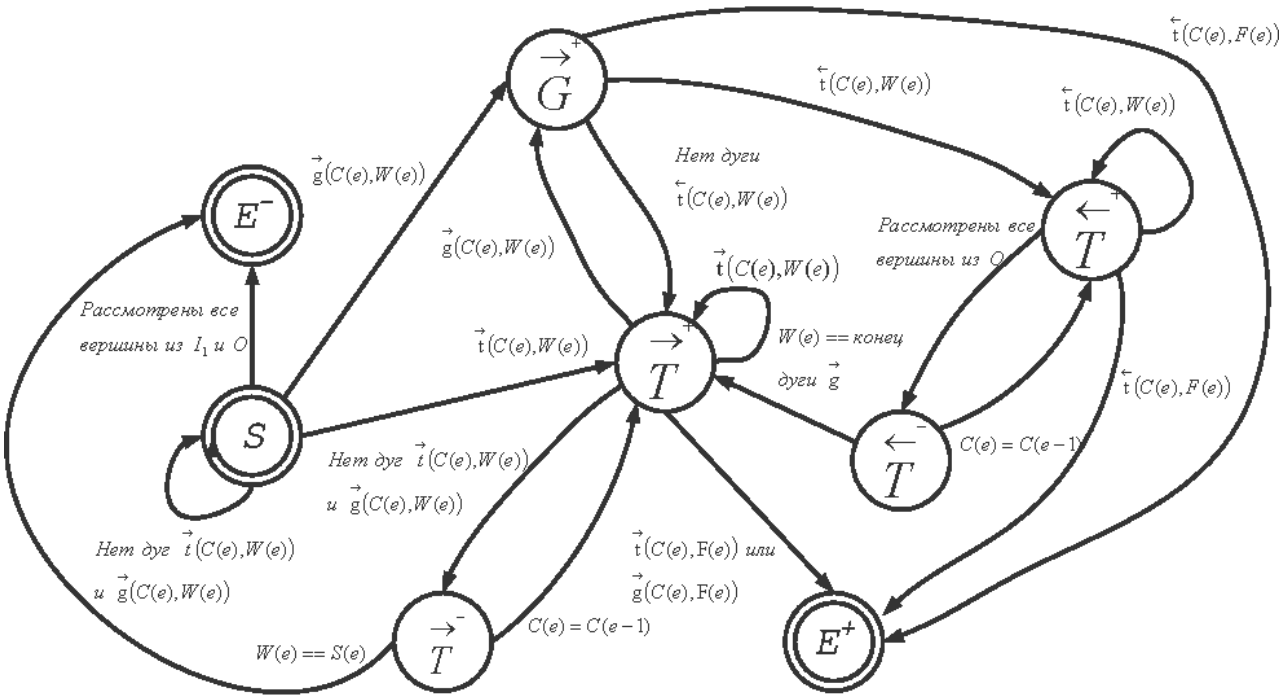

Рисунок 1 – Распознаватель моста $\vec{t}*\vec{g}\overleftarrow{t}*$

S: В самом начале работы *C(e)=S(e).* Для поиска моста между вершинами *C(e)* и *F(e)* нам необходимо рассмотреть вершины-объекты графа доступов. Выберем из *O* вершину *W(e)* и рассмотрим, каким образом эта вершина может быть связана с *C(e)*. Нас интересует, существует ли дуга $\vec{t}$ или $\vec{g}$ между *C(e)* и *W(e)*. Если такой дуги нет, необходимо выбрать для рассмотрения следующую вершину *W(e)* из *O*. Возможна ситуация, когда были рассмотрены все вершины из *O*, но подходящей дуги так и не нашлось, это означает что между *S(e)* и *F(e)* моста не существует, распознаватель переходит в конечное состояние с отрицательным результатом (E^{-}). Если обнаружена дуга $\vec{t}$*(C(e),W(e)))*, нам необходимо запомнить ее дугу, для этого пометим ее и вершину *W(e) положительно*, сделаем рассматриваемую вершину текущей (*W(e)=C(e)*), и переведем распознаватель в новое состояние $\vec{T}^{+}$. Если обнаружена дуга $\vec{g}$*(C(e),W(e)),* пометим ее и вершину *W(e)* положительно, выполним присвоение *W(e)=C(e)*, и переведем распознаватель в новое состояние, которое обозначим $\vec{G}^{+}$.

$\vec{T}^{+}$: Проверим, существуют ли *tg*-дуги между *C(e)* и *F(e)*. Если существует дуга $\vec{t}$*(C(e),F(e))*, это означает, что распознаватель обнаружил мост типа $\vec{t}*$; необходимо пометить эту дугу и перевести распознаватель в новое состояние (E^{+}). Если существует дуга $\vec{g}$*(C(e),F(e))*, это означает, что распознаватель обнаружил мост типа $\vec{t}*\vec{g}$. Пометим данную дугу и вершину положительно и переведем распознаватель в состояние E^{+}. Если *tg*-дуги между *C(e)* и *F(e),* то распознаватель выбирает вершину *W(e)* $\in$ *O* из тех, что еще не помечены. Пусть существует дуга $\vec{t}$*(C(e),W(e))* – пометим дугу и вершину *W(e)* положительно, сделаем рассматриваемую вершину текущей (*C(e)=W(e)*); распознаватель остается в состоянии $\vec{T}^{+}$ и выбирает

следующую вершину *W(e)* Є *O*. Если существует дуга $\vec{g}$*(C(e),W(e))* – пометим дугу и вершину положительно, сделаем рассматриваемую вершину текущей и переведем распознаватель в состояние $\vec{G}^{+}$. Наконец возможна ситуация, когда мы перебрали все непомеченные вершины из множества *O* и не нашли дуг $\vec{t}$ или $\vec{g}$ - это значит, что мы зашли в тупик. Попробуем вернуться к предыдущей вершине и выбрать другую, отличную от текущей. Для этого переведем распознаватель в состояние, которое будем обозначать $\vec{T}^{-}$.

$\vec{T}^{-}$: Пометим *C(e) отрицательно*, сделаем текущей предыдущую помеченную положительно вершину (*C(e)=C(e-1)*) и вернем распознаватель в состояние $\vec{T}^{+}$. Текущую вершину необходимо пометить отрицательно для того, чтобы распознаватель вновь не выбрал ее в состоянии $\vec{T}^{+}$.

$\vec{G}^{+}$: Проверим, существует ли дуга $\overleftarrow{t}$ между вершинами *C(e)* и *F(e)*. Если да, то это означает, что обнаружен мост $\vec{t}*\vec{g}\vec{t}*$. Пометим данную дугу положительно и переведем распознаватель в состояние E^{+}. Если такой дуги нет, то выбираем непомеченную вершину *W(e)* из *O*. Если существует дуга $\overleftarrow{t}$*(C(e),W(e))*, то пометим эту дугу и *W(e)* положительно, присвоим *C(e)=W(e)* и переведем распознаватель в новое состояние - $\overleftarrow{T}^{+}$. Возможна такая ситуация, когда мы перебрали все непомеченные вершины из множества *O*, но так и не нашли дуги, содержащей право $\overleftarrow{t}$. В этом случае нам придется пометить текущую вершину отрицательно, выбрать в качестве текущей предыдущую помеченную положительно (*C(e)=C(e-1)*), и вернуться в состояние $\vec{T}^{+}$.

$\overleftarrow{T}^{+}$: Проверим, существует ли дуга $\overleftarrow{t}$*(C(e),F(e))*. Если это так, то значит, обнаружен мост $\vec{t}*\vec{g}\overleftarrow{t}*$; пометим данное ребро положительно и перейдем в состояние E^{+}. В противном случае выберем непомеченную вершину *W(e)* Є *O* и проверим, существует ли дуга $\overleftarrow{t}$*(C(e),W(e))*. Если такая дуга существует, то пометим вершину *W(e)* и дугу положительно, присвоим *C(e)=W(e)* и вновь перейдем в состояние $\overleftarrow{T}^{+}$. В случае, когда перебрав все непомеченные вершины из *O* мы не нашли нужной дуги (попали в тупик), необходимо попробовать вернуться к предыдущей помеченной положительно вершине. Для этого переведем распознаватель в новое состояние, которое обозначим $\overleftarrow{T}^{-}$.

$\overleftarrow{T}^{-}$: Пометим текущую вершину отрицательно, вернемся к предыдущей помеченной положительно вершине (*C(e)=C(e-1)*) и вновь перейдем в состояние $\overleftarrow{T}^{+}$. Если при выборе предыдущей вершины мы попадем в вершину, которой закончилась дуга $\vec{g}$, то нам придется пометить ее отрицательно и вернуться в состояние $\vec{T}^{+}$.

Пусть $|E|=n, |O|=v$. Основной цикл алгоритма можно ограничить числом n. Состояния $\vec{T}^+$, $\vec{G}^+$ и $\overleftarrow{T}^+$ могут быть посещены не более чем v раз каждое, так как даже если приходится возвращаться в них из предыдущих состояний, то рассматриваются только те дуги и вершины, которые еще не были посещены. Состояния S, E^+, E^-, $\vec{T}^-$ и $\overleftarrow{T}^-$ не содержат сложных вычислений, поэтому при оценке их можно не учитывать. Таким образом, оценить сложность работы распознавателя можно как $O(3vn)$ или $O(N^3)$.

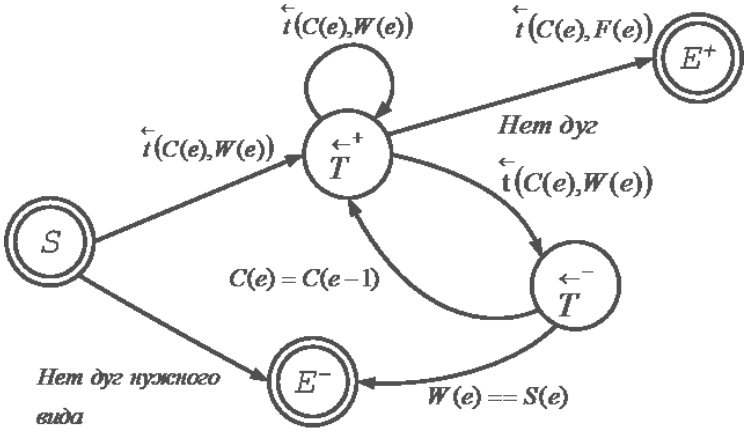

Рисунок 2 – Распознаватель моста $\overleftarrow{t}^*$

Для поиска моста $\vec{t}^*\overleftarrow{g}\overleftarrow{t}^*$ можно воспользоваться описанным распознавателем, если заменить состояние $\vec{G}^+$ на состояние $\overleftarrow{G}^+$.

Распознаватель моста $\overleftarrow{t}^*$ строится аналогичным образом. Графическая схема работы распознавателя для поиска моста $\overleftarrow{t}^*$ представлена на рисунке 2. Работу распознавателя можно оценить как $O(N^3)$.

Задача нахождения начального и конечного пролетов мостов похожа на задачу нахождения мостов, поэтому для ее решения можно воспользоваться теми же методами, то есть построить распознаватели начального и конечного пролета мостов. Графические схемы работы распознавателей приведены на рисунке 3. Оценка работы обоих распознавателей, будет равняться $O(N^3)$.

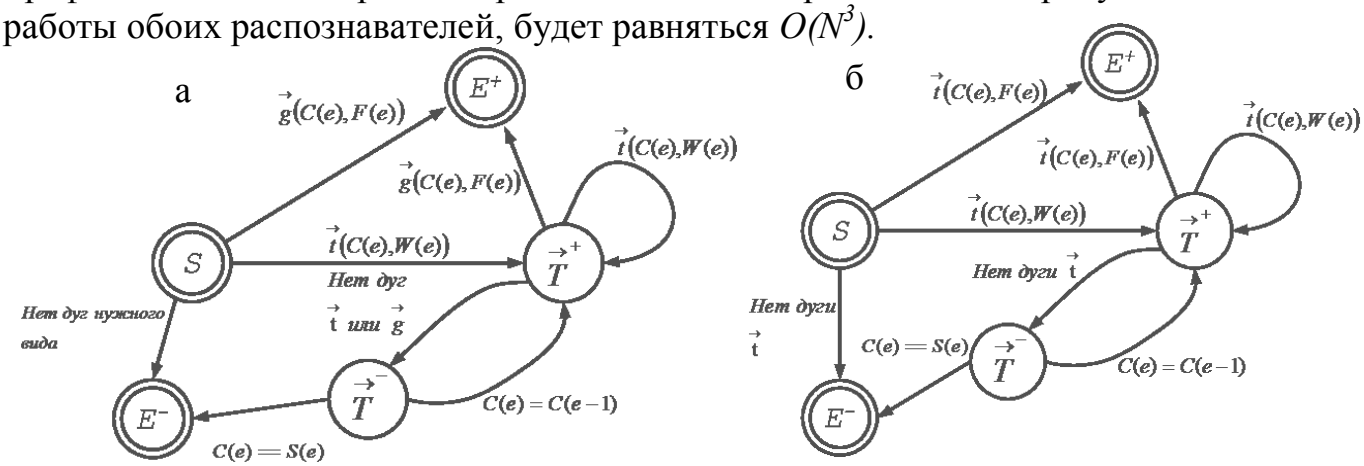

Рисунок 3 – Распознаватели пролетов моста: а – распознаватель начального пролета; б – распознаватель конечного пролета

Опишем еще один способ, нахождения мостов в произвольном графе доступов. Этот способ основан на алгоритме поиска в ширину. Начальную вершину моста

будем обозначать через *s*, а конечную – через *f*. Множество всех вершин-объектов исходного графа обозначим через *O*. Так как существует четыре типа моста, можно построить четыре разных алгоритма для каждого типа соответственно.

Рассмотрим мост $\vec{t}*$.Формально алгоритм будет состоять из трех основных этапов.

Перед началом работы алгоритма разобьем множество *O* на два подмножества O_r и O_i, причем $O = O_r \cup O_i$.

Этап 1. Вершина *s* заносится в множество O_r, все остальные вершины заносятся в O_i.

Этап 2. Просматриваются все дуги графа, началом которых являются вершины из O_r, если существует $\vec{t}(e_r, e_i)$, e_i заносится в множество O_r и удаляется из O_i. Здесь $e_r \in O_r$, $e_i \in O_i$. Когда все дуги инцидентные вершинам из O_r множества будут просмотрены - переходим на третий этап.

Этап 3. Если после выполнения этапа 2 вершина *f* оказалась во множестве O_r, то алгоритм заканчивает свою работу: мост заданного вида в графе существует. Если после выполнения этапа 2 мощности множеств O_r и O_i не изменились, алгоритм также заканчивает работу: моста заданного вида в графе не существует. В противном случае – возвращаемся на этап 2.

Пусть исходный граф содержит *N* вершин. Количество повторений этапа 2 можно ограничить количеством вершин графа, так как в случае если после каждого выполнения этапа в множество O_r будет заноситься по одной вершине, то мост в графе будет найден за *N* шагов. В общем случае во множество O_r за каждое выполнение этапа 2 будет заноситься более одной вершины, то есть мост будет найден меньше чем за *N* шагов. Если же моста не существует, то на шаге $i \le N$ не найдется дуги нужного вида и алгоритм ничего не занесет в множество O_r, то есть мощность множества останется неизменной, тогда алгоритм прервется с выдачей соответствующего сообщения. Таким образом, сложность работы алгоритма можно оценить как $O(N \cdot N \cdot (N-1))$ или $O(N^3)$.

Для моста типа $\overleftarrow{t}*$ очевидно можно использовать описанный выше алгоритм, если на втором этапе вместо дуг типа $\vec{t}$ искать дуги типа $\overleftarrow{t}$. При этом все сказанное выше будет справедливо и для этого случая, в том числе трудоемкость алгоритма также будет оцениваться как $O(N^3)$.

Рассмотрим мост типа $\vec{t}*\vec{g}\overleftarrow{t}*$. Разобьем множество *O* на четыре подмножества O_i, O_{tr}, O_{gr} и O_{tl} следующим образом: $O = O_i \cup (O_{tr} \cap O_{gr} \cap O_{tl})$. В множество O_{tr} заносятся вершины, до которых существует путь $\vec{t}$, в множество O_{gr} заносятся вершины, до которых существует путь $\vec{g}$, в множество O_{tl} заносятся те вершины, до которых существует путь $\overleftarrow{t}$, множество O_i изначально содержит все вершины кроме *s*. Вершина может находиться сразу в нескольких из этих подмножеств, но если она находится в подмножествах O_{tr}, O_{gr} и O_{tl} одновременно, то из O_i она исключается.

Опишем формально алгоритм поиска моста $\vec{t}*\vec{g}\overleftarrow{t}*$.

Этап 1. Вершина *s* заносится во множество O_{tr}, при этом в остальные множества стартовую вершину можно не включать. Все прочие вершины заносятся в O_i.

Этап 2. Перебираем все дуги графа, началом которых является вершина, принадлежащая одному или нескольким множествам O_{tr}, O_{gr} и O_{tl}. Если дуга найдена, то в зависимости от ее типа, конечная вершина этой дуги заносится в одно из множеств O_{tr}, O_{gr} или O_{tl}, согласно следующим правилам:

а) $O_{tr} = O_{tr} \cup \{e_i\}$ если $\vec{t}(e_m, e_i)$ и $e_m \in O_{tr}$;

б) $O_{gr} = O_{gr} \cup \{e_i\}$ если $\vec{g}t(e_m, e_i)$ и $e_m \in O_{tr}$;

в) $O_{tl} = O_{tl} \cup \{e_i\}$ если $\overleftarrow{t}(e_m, e_i)$ и $e_m \in O_{gr}$ или $e_m \in O_{tl}$;

г) $O_i = O_i \setminus \{e_i\}$ если $e_i \in O_{tr} \cap O_{gr} \cap O_{tl}$.

Здесь e_m может принимать значения e_{tr}, e_{gr}, e_{tl}, где $e_{tr} \in O_{tr}$, $e_{gr} \in O_{gr}$, $e_{tl} \in O_{tl}$; $e_i \in O_i$.

Этап 3. Если после выполнения этапа 2 вершина *f* оказалась в одном из множеств O_{tr}, O_{gr} или O_{tl}, то алгоритм заканчивает свою работу: мост указанного вида в графе существует. Если после выполнения этапа 2 не изменилась мощность ни одного множества, то алгоритм также останавливается: моста заданного вида не существует. В ином случае возвращаемся на этап 2.

При оценке трудоемкости алгоритма надо учитывать, что по сравнению с вышеописанным случаем число множеств, в которые заносятся вершины, возросло втрое. В худшем случае на каждом шаге второго этапа одна вершина попадает в одно из множеств, однако потенциально каждая вершина может попасть во все три множества. Таким образом, количество повторений второго этапа следует увеличить до *3N*. Количество дуг в графе остается неизменным, потому общую трудоемкость алгоритма можно оценить как $O(3 \cdot N \cdot N \cdot (N-1))$ или $O(N^3)$.

Очевидно, что для моста $\vec{t}*\overleftarrow{g}\overleftarrow{t}*$ можно использовать описанный выше алгоритм, если на втором этапе в множество O_{gr} включать дуги вида $\overleftarrow{g}$ вместо $\vec{g}$. Все сказанное относительно моста $\vec{t}*\vec{g}\overleftarrow{t}*$ будет справедливо и для моста $\vec{t}*\overleftarrow{g}\overleftarrow{t}*$, включая трудоемкость, которую также можно оценить как $O(N^3)$.

Начальный пролет моста имеет вид $\vec{t}*\vec{g}$, поэтому, перед началом работы необходимо выбрать вершину $a \in O$ такую, что $\vec{g}(a,x)$, где $x \in O$ - конечная вершина начального пролета моста. При выборе вершины *a* возможно будет необходимо рассмотреть все вершины из *O*, поэтому поиск вершины *a* можно оценить как $O(N^3)$. После того, как подходящая вершина найдена, для поиска начального пролета моста можно применить алгоритм, использовавшийся для поиска моста вида $\vec{t}*$.

Для поиска конечного пролета моста, аналогично необходимо сначала найти вершину *b* такую что $\alpha(b,y)$, где α - необходимый доступ или набор доступов, *y* – вершина, возможность доступа до которой выясняется. Для поиска вершины *b* точно также необходимо просмотреть все вершины из *O*. Далее, для поиска конечного пролета можно также применить алгоритм, аналогичный алгоритму для поиска моста вида $\vec{t}*$, вершина *s* будет принадлежать какому-либо острову, а *f=b*.

Для того чтобы применение модели Take-Grant стало возможным, необходимо выяснить, присутствуют ли в Windows аналоги прав *Take* и *Grant*. В системе Windows возможность передавать права на объект другим субъектам имеют администратор, владелец объекта, либо любой пользователь с полными правами на объект. Информация о владельце объекта и правах субъектов на объект содержится в дескрипторе безопасности объекта, потому, анализируя дескриптор безопасности можно однозначно выявить субъекты, имеющие право *Grant* на объект.

Помимо прочих прав доступа в Windows присутствует право *Take Ownership* (смена владельца). Субъект, обладающий этим правом на объект может стать владельцем объекта и получить любые права на этот объект. Таким образом, можно говорить, что субъект, обладающий правом *Take Ownership*, обладает правом *Take* на объект.

В заключении сформулированы основные результаты и выводы.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

1. Проведено развитие модели безопасности компьютерных систем с дискреционным разделением доступа HRU.

1.1 Возможно построение минимального набора операций на множестве матриц доступа (базиса).

1.2 Использование базиса позволяет выбирать наиболее удобное представление команд преобразования матрицы доступов.

1.3 Использование базисного подхода позволяет выявить новые классы безопасности компьютерных систем с дискреционным разделением доступа.

2. Исследовано дискреционное разграничение доступа в операционных системах семейства Windows.

2.1 Применен базисный подход к исследованию системы защиты от несанкционированного доступа в компьютерных системах под управлением ОС Windows.

2.2 Сформулированы рекомендации по разработке дополнительной системы защиты, делающей объекты ОС Windows защищенными от несанкционированных доступов.

3. Построены алгоритмы проверки безопасности компьютерных систем с дискреционным разделением доступа в рамках модели Take-Grant.

3.1 Разработан полиномиальный алгоритм проверки отсутствия несанкционированных доступов в системах с дискреционным разделением доступов с трудоемкостью $O(N^3)$.

3.2 Исследована применимость построенных алгоритмов к компьютерным системам под управлением ОС Windows. Показана возможность программной реализации дополнительных систем защиты, исследующих возможность несанкционированного доступа заданного пользователя к определенному объекту.

Основное содержание диссертации опубликовано в следующих работах:

В научных журналах, рекомендованных ВАК:

1. *Бречка Д.М., Белим С.В.* Базисный подход в модели безопасности HRU //Проблемы информационной безопасности. Компьютерные системы, 2010, В.2., С.7-13.

2. *Белим С.В., Бречка Д.М.* Выявление новых классов безопасности в рамках модели HRU//Безопасность информационных технологий, 2010, В. 3. С. 10-15

В других изданиях:

3. *Бречка Д.М.* Дискреционная политика безопасности и ее моделирование // Проблемы обработки и защиты информации. Книга 1. Модели политик безопасности компьютерных систем. Коллективная монография / Под общей редакцией С.В. Белима. – Омск: ООО «Полиграфический центр КАН», 2010.

4. *Белим С.В., Бречка Д.М.* Классы безопасности модели ХРУ // Проблемы обработки и защиты информации. Книга 1. Модели политик безопасности компьютерных систем. Коллективная монография / Под общей редакцией С.В. Белима. – Омск: ООО «Полиграфический центр КАН», 2010

5. *Бречка Д.М.* Алгоритмы проверки безопасности состояний компьютерной системы в модели Take-Grant//Проблемы обработки и защиты информации. Книга 1. Модели политик безопасности компьютерных систем. Коллективная монография/ Под общей редакцией С.В.Белима–Омск:ООО«Полиграфический центр КАН», 2010.

6. *Бречка Д.М., Белим С.В.* Исследование безопасности компьютерных систем в модели дискреционного разделения доступа HRU // Математические структуры и моделирование., 2009., №19. С.97-103

7. *Бречка Д.М.* Алгоритмы анализа безопасности состояний компьютерной системы для модели Take-Grant // Математические структуры и моделирование., 2009., №20. С.160-172

8. *Бречка Д. М., Белим С. В.* Расширение классов безопасности систем в модели дискреционного доступа HRU // Современные проблемы гуманитарных и естественных наук: материалы второй международной научно-практической конференции, Том II, Москва: ООО «Открытое право», 2010, С.63-67

9. *Бречка Д. М.* Исследование безопасности систем в модели дискреционного разделения доступа HRU // Информационные технологии автоматизации и управления: материалы межвузовской научно-практической конференции, 20-24 апреля 2009 г., Омск: Изд-во ОмГТУ, 2009., С.164.

10. *Бречка Д. М.*, *Белим С.В.* Построение алгоритмов проверки безопасности систем с дискреционным разделением доступа // Решетневские чтения: Материалы XIII Международной научно-практической конференции, Ч.2, 10 - 12 ноября 2009 г., Красносярск.: Сиб. гос. аэрокосмич. ун-т., 2009., С.573-574

11. *Белим С.В., Бречка Д.М.* Расширение класса безопасных систем в модели HRU // В мире научных открытий., 2010, № 4(10) Часть 4, Красноярск: Научно-информационный центр, С. 9-11.

12. *Бречка Д.М.* Анализ возможности доступа в модели Take-Grant // В мире научных открытий., 2010, № 4(10) Часть 4, Красноярск: Научно-информационный центр, С. 11-13.

13. *Бречка Д.М.* Применение алгоритмов на графах для поиска безопасных состояний компьютерной системы //Наука в современном мире: Материалы II Международной научно-практической конференции (30 июля 2010): сборник научных трудов / Под ред. Г.Ф. Гребенщикова. – М.: Издательство «Спутник+», 2010

14. *Бречка Д.М.* Поиск tg-путей и островов для модели безопасности Take-Grant// Прикладная дискретная математика №3. Приложение. Тезисы докладов IX

Сибирской научной школы-семинара с международным участием «Компьютерная безопасность и криптография» - SIBECRYPT'10 (Тюмень, ТюмГУ, 7-10 сентября 2010 г.) / ред. Н.И. Шидловская. – Томск: ООО «Издательство научно-технической литературы», 2010. С. 46-47.

Список цитируемой литературы

1. Гайдамакин Н.А. Разграничение доступа к информации в компьютерных системах. Е.: Издательство Уральского Университета, 2003. 328 с.
2. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. Учебник. СПб.: Питер, 2001. 736 с.
3. Грушо А.А., Тимонина Е.Е. Теоретические основы защиты информации. М.: Яхтсмен, 1996. 192с.
4. Девянин П.Н. Модели безопасности компьютерных систем: Учебное пособие для студентов высших учебных заведений. М.: Издательский центр «Академия», 2005. 144 с.
5. Майника Э. Алгоритмы оптимизации на сетях и графах. М.: Мир, 1981. 323 с.
6. Руссинович М. Соломон Д. Внутреннее устройство Microsoft Windows: Windows Server 2003, Windows XP и Windows 2000. Мастер-класс. Пер. с англ. 4-е изд. М.: Издательско-торговый дом «Русская редакция», 2005. 992 с.
7. Теоретические основы компьютерной безопасности: Учебное пособие для вузов / Девянин П.Н. и др. М.: Радио и связь, 2001. 192 с.
8. Department of Defense Trusted Computer System Evaluation Criteria, TCSEC, DoD 5200.28-STD, December 26, 1985
9. Harrison M.A., Ruzzo W.L. Monotonic protection systems // Foundation of Secure Computation. New York: Academic Press, 1978. - P. 337-365.
10. Harrison M.A., Ruzzo W.L., Ulman J.D. Protection in Operating Systems // Communications of the ACM, 1975. p. 14-25.
11. Lipton R.J., Snayder L. A linear time algorithm for deciding subject security // Journal of ACM (Addison-Wesley). N.3, 1977. p.455-464
12. Microsoft Corporation Microsoft Windows XP Professional. Учебный курс MCSA/MCSE. Пер. с англ. 2-е изд. Испр. М.: Издательско-торговый дом «Русская редакция», 2003. 1008 с.

Подписано в печать. Формат бумаги 69х86 1/16. Печ. л. 4,5. Тираж 100 экз. Заказ 153

Отпечатано на полиграфической базе ОмГУ
644077, г.Омск, пр. Мира 55-а