

УДК 65.012.810(075.8)  
ББК 32.973.202я73  
Д259

## ПРЕДИСЛОВИЕ

### Рецензенты:

канд. техн. наук, зав. кафедрой Информационной безопасности  
МИЭМ А. Б. Лось; канд. техн. наук, доцент МИФИ А. И. Толстой;  
канд. техн. наук, доцент МИФИ В. А. Петров

### Девягин П. Н.

Д259    Модели безопасности компьютерных систем: Учеб. пособие для студ. высш. учеб. заведений / Петр Николаевич Девягин. — М.: Издательский центр «Академия», 2005. — 144 с.  
ISBN 5-7695-2053-1

Рассмотрены с полными доказательствами положения основных моделей безопасности компьютерных систем: дискреционного, мандатного, ролевого разграничений доступа, безопасности информационных потоков, изолированной программной среды. Приведен используемый в рассматриваемых моделях математический аппарат.

Учебное пособие разработано при содействии Академии криптографии Российской Федерации.

Для студентов высших учебных заведений. Может быть полезно специалистам в области защиты информации.

УДК 65.012.810(075.8)  
ББК 32.973.202я73

Оригинал-макет данного издания является собственностью  
Издательского центра «Академия», и его воспроизведение любым способом  
без согласия правообладателя запрещается

В настоящее время имеется достаточно много литературы, в том числе учебной, в которой приводятся обзоры и описания различных моделей безопасности компьютерных систем. В то же время их изложение, как правило, носит фрагментарный характер. Основное внимание в имеющейся литературе уделяется лишь общей формулировке основных определений и результатов моделей безопасности либо краткому их перечислению обзорного характера (без подробного рассмотрения, применяемого математического аппарата и приведения доказательств). В то же время в книгах, где доказательства приводятся, они, как правило, даются в общих чертах.

В данном учебном пособии рассмотрены с полными доказательствами положения ряда основных моделей безопасности компьютерных систем. Приведен используемый в рассматриваемых моделях математический аппарат.

Учебное пособие разработано при содействии Академии криптографии Российской Федерации.

Учебное пособие предназначено для дисциплин «Теоретические основы компьютерной безопасности», «Безопасность операционных систем», «Безопасность вычислительных систем» по специальностям 075200 «Компьютерная безопасность» и 075500 «Комплексное обеспечение информационной безопасности автоматизированных систем» и для дисциплины «Основы компьютерной безопасности» для непрофилирующих специальностей.

Учебное пособие написано на основе десятилетнего опыта преподавания перечисленных дисциплин в Институте криптографии, связи и информатики (ИКСИ).

# ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ, ИСПОЛЬЗУЕМЫЕ ПРИ ОПИСАНИИ МОДЕЛЕЙ БЕЗОПАСНОСТИ КОМПЬЮТЕРНЫХ СИСТЕМ

## 1.1. ЭЛЕМЕНТЫ ТЕОРИИ ЗАЩИТЫ ИНФОРМАЦИИ

### 1.1.1. Объект, субъект, доступ

Дадим основные определения на основе [2].

Используем следующие обозначения:

$A$  — конечный алфавит;

$A^*$  — множество слов конечной длины в алфавите  $A$ ;

$\mathcal{A} \subset A^*$  — язык описания информации, являющийся множеством слов, выделенных по определенным правилам из  $A$ .

**Аксиома 1.1.** Любая информация в компьютерной системе представляется словом в некотором языке  $\mathcal{A}$ .

**Определение 1.1.** Объект относительно языка  $\mathcal{A}$  (или просто объект) — произвольное конечное множество слов языка  $\mathcal{A}$ .

**Определение 1.2.** Преобразование информации — отображение, заданное на множестве слов языка  $\mathcal{A}$ .

Описание преобразования также является словом.

Для выполнения преобразования информации в компьютерных системах требуется время, поэтому каждое преобразование может или выполняться, или храниться. В первом случае описание преобразования взаимодействует с другими ресурсами компьютерной системы. Во втором случае, как правило, речь идет о хранении описания преобразования в некотором файле.

**Определение 1.3.** Субъект — объект, описывающий преобразование, которое выполняется в компьютерной системе.

Используем следующие обозначения:

$O$  — множество объектов системы;

$S$  — множество субъектов системы ( $S \subseteq O$ ).

**Определение 1.4.** Информационным потоком от объекта  $o$  (источника) к объекту  $o'$  (приемнику) назовем преобразование информации в объекте  $o'$ , зависящее от информации в объекте  $o$ .

Субъект для выполнения преобразования использует информацию, содержащуюся в объектах компьютерной системы, т. е. осуществляя к ним доступ. Основными видами доступа являются:

- доступ субъекта к объекту на чтение (*read*);
- доступ субъекта к объекту на запись (*write*), при этом возможно уничтожение информации, имевшейся в объекте;

- доступ субъекта к объекту на активизацию (*execute*), при этом инициируется выполнение (производится активизация) в компьютерной системе преобразования информации, описанного в объекте.

**Определение 1.5.** В каждом состоянии компьютерной системы на множестве субъектов введем бинарное отношение  $a$  активизации: выполняется  $s_1 a s_2$ , где  $s_1, s_2 \in S$ , если субъект  $s_1$ , выполняя описанное в нем преобразование информации, инициирует выполнение преобразования информации, описанного в субъекте  $s_2$ .

**Определение 1.6.** В каждом состоянии компьютерной системы определим ориентированный граф активизации  $G = (S, E)$ , где  $S$  — вершины графа;  $E$  — ребра графа, соединяющие субъектов, связанных бинарным отношением активизации.

Очевидно, что граф активизации представляет собой множество деревьев, или лес (см. подразд. 1.2).

**Определение 1.7.** Пользователи — субъекты компьютерной системы, соответствующие корням деревьев графа активизации.

В критериях оценки защищенности компьютерных систем *TCSEC* [23] («Оранжевая книга») приводится основная аксиома теории защиты информации, позволяющая выделить элементы компьютерной системы, необходимые для анализа ее безопасности.

**Аксиома 1.2 (основная аксиома теории защиты информации).** Все вопросы безопасности информации описываются доступами субъектов к объектам.

### 1.1.2. Классификация угроз безопасности информации

**Определение 1.8.** Угроза безопасности информации (компьютерной системы) — потенциально возможное воздействие на информацию (компьютерную систему), которое прямо или косвенно может нанести урон пользователям или владельцам информации (компьютерной системы).

При классификации угроз выделяют три основных свойства безопасности информации [6].

**Определение 1.9.** Конфиденциальность информации — субъективно определяемая характеристика информации, указывающая на необходимость введения ограничений на круг субъектов, имеющих доступ к данной информации.

**Определение 1.10.** Целостность информации — свойство информации, заключающееся в ее существовании в неискаженном виде (неизменном по отношению к некоторому фиксированному ее состоянию).

**Определение 1.11.** Доступность информации — свойство компьютерной системы (среды, средств и технологии обработки), в

которой циркулирует информация, характеризующееся способностью обеспечивать своевременный беспрепятственный доступ субъектов к интересующей их информации и готовность соответствующих автоматизированных служб к обслуживанию поступающих от субъектов запросов всегда, когда в обращении к ним возникает необходимость.

В соответствии с тремя основными свойствами безопасности информации различают три классические угрозы безопасности информации.

**Определение 1.12.** Угроза конфиденциальности информации состоит в нарушении установленных ограничений на доступ к информации.

**Определение 1.13.** Угроза целостности информации — несанкционированное изменение информации, случайное или преднамеренное.

**Определение 1.14.** Угроза доступности информации осуществляется, когда несанкционированно блокируется доступ к информации (блокирование может быть постоянным или на некоторое время, достаточное, чтобы информация стала бесполезной).

Кроме перечисленных угроз выделяют еще одну угрозу, реализация которой, как правило, предшествует реализации любой из классических угроз.

**Определение 1.15.** Угроза раскрытия параметров компьютерной системы — преодоление защиты компьютерной системы, выявление параметров, функций и свойств ее системы безопасности.

Осуществление угрозы конфиденциальности информации нередко происходит с использованием неблагоприятных информационных потоков (каналов утечки информации). Выделяют два основных вида неблагоприятных информационных потоков: по памяти и по времени.

Информационный поток по памяти описывается схемой, представленной на рис. 1.1.

Информационный поток по времени описывается схемой, представленной на рис. 1.2.

Опасность неблагоприятного информационного потока по времени определяется долей ценной информации, модулируемой на процесс  $s_1$ . Наиболее распространенными случаями реализации неблагоприятного информационного потока по времени являются:

- перехват информации в канале связи;
- побочные каналы утечки информации по излучению, сети питания или акустике.

При анализе угрозы целостности информации следует иметь в виду, что язык ее описания аналогичен языку описания угрозы конфиденциальности. Используя при описании требований защиты информации от угрозы целостности доступы субъектов к объектам, можно сделать выводы, аналогичные выводам, полученным

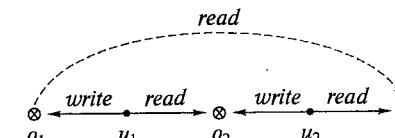


Рис. 1.1. Информационный поток по памяти:

$u_1$  — нарушитель;  $u_2$  — пользователь, обрабатывающий ценную информацию;  $o_1$  — объект, доступный нарушителю на запись;  $o_2$  — общедоступный объект;  $o_3$  — ценный объект

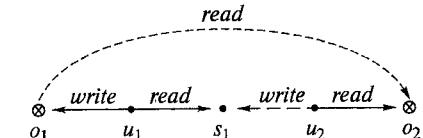


Рис. 1.2. Информационный поток по времени:

$u_1$  — нарушитель;  $u_2$  — пользователь, обрабатывающий ценную информацию;  $s_1$  — процесс, на который отражается по времени информация о работе пользователя  $u_2$ ;  $o_1$  — объект, доступный нарушителю на запись;  $o_2$  — ценный объект

при описании требований защиты от угрозы конфиденциальности; при этом следует заменить доступы на чтение информации доступами на запись, и наоборот.

Угроза доступности информации, как правило, описывается с использованием параметра, называемого максимальным временем ожидания ответа на запрос на доступ к ресурсу ( $MWT$  — *maximum wait time*). Таким образом, для каждого ресурса компьютерной системы определяется время, приемлемое для ожидания его получения.

Следует считать, что реализовалась угроза доступности информации, если ресурс с соответствующим максимальным временем ожидания, санкционировано запрашиваемый пользователем в момент времени  $t$ , не предоставлен ему к моменту времени  $t + MWT$ .

Можно использовать несколько подходов для определения  $MWT$ :

- определив  $MWT = \infty$  для всех ресурсов, можно исключить рассмотрение угрозы доступности информации при анализе безопасности компьютерной системы;
- можно определить  $MWT$  только для некоторых доступов субъектов к ресурсам компьютерной системы, действительно критичных с точки зрения ее безопасности.

### 1.1.3. Основные виды политик безопасности

#### Дискреционная политика безопасности

**Определение 1.16.** Дискреционная политика безопасности — политика безопасности, основанная на дискреционном управлении доступом (*Discretionary Access Control*), которое определяется двумя свойствами:

- все субъекты и объекты идентифицированы;

- права доступа субъектов на объекты системы определяются на основании некоторого внешнего по отношению к системе правила.

Основным элементом систем дискреционного разграничения доступа является матрица доступов.

**Определение 1.17.** Матрица доступов — матрица размером  $|S| \times |O|$ , строки которой соответствуют субъектам, а столбцы соответствуют объектам. При этом каждый элемент матрицы доступов  $M[s, o] \subseteq R$  определяет права доступа субъекта  $s$  на объект  $o$ , где  $R$  — множество прав доступа.

К достоинствам дискреционной политики безопасности можно отнести относительно простую реализацию системы разграничения доступа. Этим обусловлен тот факт, что большинство распространенных в настоящее время компьютерных систем обеспечивают выполнение требований именно данной политики безопасности.

К недостаткам дискреционной политики безопасности относится статичность определенных в ней правил разграничения доступа. Данная политика безопасности не учитывает динамику изменений состояний компьютерной системы. Кроме того, при использовании дискреционной политики безопасности возникает вопрос определения правил распространения прав доступа и анализа их влияния на безопасность компьютерной системы. В общем случае при использовании данной политики безопасности перед системой защиты, которая при санкционировании доступа субъекта к объекту руководствуется некоторым набором правил, стоит алгоритмически неразрешимая задача — проверить, приведут ли его действия к нарушению безопасности или нет. Данный факт доказывается в подразд. 2.1.

В то же время имеются модели компьютерных систем, реализующих дискреционную политику безопасности (например, рассматриваемая в подразд. 2.2 модель *Take-Grant*), которые предоставляют алгоритмы проверки безопасности.

Тем не менее, в общем случае дискреционная политика разграничения доступа не позволяет реализовать ясную и четкую систему защиты информации в компьютерной системе. Этим обуславливается поиск других, более совершенных политик безопасности.

## Мандатная политика безопасности

**Определение 1.18.** Мандатная (полномочная) политика безопасности — политика безопасности, основанная на мандатном разграничении доступа (*Mandatory Access Control*), которое определяется четырьмя условиями:

- все субъекты и объекты системы однозначно идентифицированы;

- задана решетка уровней конфиденциальности информации;
- каждому объекту системы присвоен уровень конфиденциальности, определяющий ценность содержащейся в нем информации;

- каждому субъекту системы присвоен уровень доступа, определяющий уровень доверия к нему в компьютерной системе.

Основная цель мандатной политики безопасности — предотвращение утечки информации от объектов с высоким уровнем доступа к объектам с низким уровнем доступа, т. е. противодействие возникновению в компьютерной системе неблагоприятных информационных потоков сверху вниз.

Чаще всего мандатную политику безопасности описывают в терминах, понятиях и определениях свойств модели Белла—Лападула, которая рассмотрена в подразд. 3.1. Для систем мандатного разграничения доступа задача проверки безопасности является алгоритмически разрешимой. Кроме того, по сравнению с компьютерными системами, построенными на основе дискреционной политики безопасности, для систем, реализующих мандатную политику, характерна более высокая степень надежности. Правила мандатной политики безопасности более ясны и просты для понимания разработчиками и пользователями, что также является фактором, положительно влияющим на уровень безопасности системы.

С другой стороны, реализация систем с политикой безопасности данного типа довольно сложна и требует значительных ресурсов компьютерной системы.

## Политика безопасности информационных потоков

Политика безопасности информационных потоков основана на разделении всех возможных информационных потоков между объектами системы на два непересекающихся множества: множество благоприятных информационных потоков и множество неблагоприятных информационных потоков. Цель реализации политики безопасности информационных потоков состоит в том, чтобы обеспечить невозможность возникновения в компьютерной системе неблагоприятных информационных потоков.

Политика безопасности информационных потоков в большинстве случаев используется в сочетании с политикой другого вида, например с политикой дискреционного или мандатного разграничения доступа. Реализация политики безопасности информационных потоков, как правило, на практике является трудной для решения задачей, особенно, если необходимо обеспечить защиту компьютерной системы от возникновения неблагоприятных информационных потоков по времени.

Некоторые из видов моделей политик безопасности информационных потоков рассмотрены в гл. 4.

## Политика ролевого разграничения доступа

Ролевое разграничение доступа является развитием политики дискреционного разграничения доступа; при этом права доступа субъектов системы на объекты группируются с учетом специфики их применения, образуя роли.

Задание ролей позволяет определить более четкие и понятные для пользователей компьютерной системы правила разграничения доступа. Ролевое разграничение доступа позволяет реализовать гибкие, изменяющиеся динамически в процессе функционирования компьютерной системы правила разграничения доступа. На основе ролевого разграничения доступа, в том числе, может быть реализовано мандатное разграничение доступа. Ролевое разграничение доступа рассмотрено в гл. 5.

## Политика изолированной программной среды

Целью реализации политики изолированной программной среды является определение порядка безопасного взаимодействия субъектов системы, обеспечивающего невозможность воздействия на систему защиты и модификации ее параметров или конфигурации, результатом которых могло бы стать изменение реализуемой системой политики разграничения доступа.

Политика изолированной программной среды реализуется путем изоляции субъектов системы друг от друга и путем контроля порождения новых субъектов таким образом, чтобы в системе могли активизироваться только субъекты из предопределенного списка. При этом должна контролироваться целостность объектов системы, влияющих на функциональность активизируемых субъектов.

Подробно политика изолированной программной среды рассмотрена в гл. 6.

# 1.2. МАТЕМАТИЧЕСКИЕ ОСНОВЫ МОДЕЛЕЙ БЕЗОПАСНОСТИ

## 1.2.1. Основные понятия

Большая часть используемых при описании свойств моделей безопасности компьютерных систем сведений из математики рассмотрены в тех подразделах, в которых они непосредственно ис-

пользуются. Особое внимание необходимо уделить следующим наиболее важным математическим понятиям, используемым в нескольких моделях безопасности:

- автомат;
- граф;
- алгоритмически разрешимые и алгоритмически неразрешимые проблемы;
- решетка.

### 1.2.2. Элементы теории автоматов

В математических моделях безопасности компьютерных систем понятие автомата часто используется для описания свойств системы защиты, ее отдельных элементов или компьютерной системы в целом.

**Определение 1.19.** Автоматом  $A(X, S, Y, h, f)$  называется совокупность пяти элементов:

- $X \neq \emptyset$  — конечный входной алфавит;  
 $S \neq \emptyset$  — конечное множество состояний;  
 $Y \neq \emptyset$  — конечный выходной алфавит;  
 $h: X \times S \rightarrow S$  — функция переходов автомата;  
 $f: X \times S \rightarrow Y$  — функция выходов автомата.

При этом  $A_{s_0}: X^* \rightarrow Y^*$  — автоматное отображение, где  $s_0$  — начальное состояние автомата;  $X^*$  и  $Y^*$  — конечные последовательности элементов соответственно входного и выходного алфавитов.

**Определение 1.20.** Гомоморфизмом  $\varphi$  автомата  $A(X, S, Y, h, f)$  в автомат  $A'(X', S', Y', h', f')$  называется совокупность трех функций  $\varphi = (\alpha, \beta, \gamma)$ , где

- $\alpha: X \rightarrow X'$ ;  
 $\beta: S \rightarrow S'$ ;  
 $\gamma: Y \rightarrow Y'$ .

При этом для  $x \in X, s \in S, y \in Y$  справедливы равенства:  
 $\beta(h(x, s)) = h'(\alpha(x), \beta(s))$ ;  
 $\gamma(f(x, s)) = f'(\alpha(x), \beta(s))$ .

Таким образом, коммутативная диаграмма, представленная на рис. 1.3.

**Определение 1.21.** Суммой двух автоматов  $A_1(X, S_1, Y_1, h_1, f_1)$  и  $A_2(X, S_2, Y_2, h_2, f_2)$ , где  $S_1 \cap S_2 = \emptyset$ , называется автомат  $(A_1 + A_2)(X, S_1 \cup S_2, Y_1 \cup Y_2, h, f)$ ; при этом для  $x \in X$  выполняются условия (рис. 1.4):

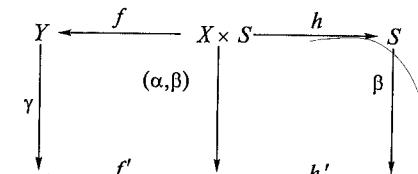


Рис. 1.3. Гомоморфизм автоматов

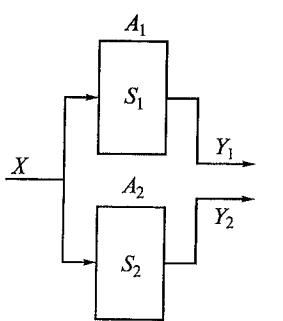


Рис. 1.4. Сумма двух автоматов  
 $(A_1 + A_2)(X, S_1 \cup S_2, Y_1 \cup Y_2, h, f)$

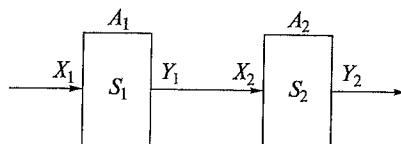


Рис. 1.5. Последовательное соединение двух автоматов  $(A_1 \rightarrow A_2)(X_1, S_1 \times S_2, Y_2, h, f)$

$$\begin{aligned} h(x, s) &= h_i(x, s), \text{ если } s \in S_i, \text{ для } i \in \{1, 2\}; \\ f(x, s) &= f_i(x, s), \text{ если } s \in S_i, \text{ для } i \in \{1, 2\}. \end{aligned}$$

**Определение 1.22.** Последовательным соединением двух автоматов  $A_1(X_1, S_1, Y_1, h_1, f_1)$  и  $A_2(X_2, S_2, Y_2, h_2, f_2)$ , где  $Y_1 \subseteq X_2$ , называется автомат  $(A_1 \rightarrow A_2)(X_1, S_1 \times S_2, Y_2, h, f)$ ; при этом для  $x \in X_1, s_1 \in S_1, s_2 \in S_2$  справедливы равенства (рис. 1.5):

$$\begin{aligned} h(x, (s_1, s_2)) &= (h_1(x, s_1), h_2(f_1(x, s_1), s_2)); \\ f(x, (s_1, s_2)) &= f_2(f_1(x, s_1), s_2). \end{aligned}$$

### 1.2.3. Элементы теории графов

Элементы теории графов широко используются в математических моделях безопасности компьютерных систем, например для представления траекторий состояний функционирования системы, матрицы доступов систем дискреционного разграничения доступа.

**Определение 1.23.** Графом  $G = (V, E)$  называется совокупность двух элементов:

$V$  — конечное множество вершин графа;

$E \subseteq V \times V$  — множество ребер графа.

При этом, если ребра  $(v_1, v_2)$  и  $(v_2, v_1)$  считаются одинаковыми, то граф называется неориентированным, в противном случае граф называется ориентированным.

**Определение 1.24.** Вершины  $v, v' \in V$  соединены путем в графе  $G = (V, E)$ , если существует последовательность ребер  $(v_1, v_2), \dots, (v_{n-1}, v_n) \in E$ , где  $v_1 = v$ ,  $v_n = v'$  и  $n \geq 1$ . Если  $v = v'$ , то такой путь называется циклом.

**Определение 1.25.** Подграфом  $G' = (V', E')$  графа  $G = (V, E)$  называется граф, где  $E' \subset E$  и  $V' \subset V$  — множество вершин, являющихся концами ребер из  $E'$ .

**Определение 1.26.** Вершины  $v, v' \in V$  являются связными в графе  $G = (V, E)$ , если  $v$  соединена путем с  $v'$  или  $v'$  соединена путем с  $v$ .

**Определение 1.27.** Вершины  $v, v' \in V$  являются сильно связными в графе  $G = (V, E)$ , если  $v$  соединена путем с  $v'$  и  $v'$  соединена путем с  $v$ .

**Определение 1.28.** Подграф, состоящий из связных (сильно связных) между собой вершин, называется компонентой связности (сильной связности) графа.

Сильно связные вершины являются связными. В неориентированном графе связные вершины являются сильно связными.

**Определение 1.29.** Ориентированный граф  $G = (V, E)$  называется деревом, если:

- в графе не содержатся циклы;
- существует единственная вершина (корень дерева), в которую не входит ни одно ребро;
- из корня в любую другую вершину графа существует единственный путь;
- в любую вершину, кроме корня, входит только одно ребро.

**Определение 1.30.** Граф, состоящий из совокупности деревьев, называется лесом.

Наиболее часто в математических моделях безопасности компьютерных систем используются следующие близкие по идее реализации алгоритмы на графах [4]:

- алгоритм обхода вершин графа;
- алгоритм поиска кратчайшего пути между вершинами графа;
- алгоритм поиска компонент связности;
- алгоритм обхода вершин дерева.

### 1.2.4. Алгоритмически разрешимые и алгоритмически неразрешимые проблемы

Понятие алгоритмически неразрешимой проблемы рассмотрим на основе [5].

Рассмотрим так называемые массовые проблемы. Массовая проблема представляет собой бесконечную серию индивидуальных задач.

Без ограничения общности рассмотрим массовые задачи, все индивидуальные задачи которых имеют двузначный ответ («да» и «нет»).

**Определение 1.31.** Характеристической функцией для массовой проблемы  $P$  называется функция  $f: P \rightarrow \{0, 1\}$ , где для индивидуальной задачи  $\pi \in P$  выполняются условия:

- $f(\pi) = 1$ , если  $\pi$  имеет ответ «да»;
- $f(\pi) = 0$ , если  $\pi$  имеет ответ «нет».

**Определение 1.32.** Массовая задача является алгоритмически разрешимой, если существует алгоритм, вычисляющий ее характеристическую функцию. В противном случае массовая проблема является алгоритмически неразрешимой.

Решая конкретную массовую проблему, следует считаться с возможностью, что она может оказаться алгоритмически неразрешимой, поэтому необходимо иметь представление о технике доказательства неразрешимости. Основной метод, применяемый для доказательства алгоритмической неразрешимости, базируется на следующем рассуждении. Пусть имеются две массовые проблемы  $P_1$  и  $P_2$ . Пусть имеется алгоритм  $A$ , который по всякой индивидуальной задаче  $p_1 \in P_1$  строит индивидуальную задачу  $p_2 \in P_2$  такую, что  $p_1$  имеет ответ «да» тогда и только тогда, когда  $p_2$  имеет ответ «да».

В этом случае говорят, что проблема  $P_1$  сводится к проблеме  $P_2$ . Если проблема  $P_1$  неразрешима, то проблема  $P_2$  также неразрешима.

Примером алгоритмически неразрешимой проблемы является проблема останова машины Тьюринга.

## 1.2.5. Модель решетки

Пусть  $X$  — конечное множество.

**Определение 1.33.** Бинарное отношение « $<$ » на множестве  $X$  называется отношением строгого порядка, если для любых  $a, b, c \in X$  выполняются три свойства:

- антирефлексивность: не выполняется  $a < a$ ;
- транзитивность:  $(a < b, b < c) \Rightarrow (a < c)$ ;
- антисимметричность:  $(a < b, b < a) \Rightarrow (a = b)$ .

**Определение 1.34.** Бинарное отношение « $\leq$ » на множестве  $X$  называется отношением частичного порядка, если для любых  $a, b, c \in X$  выполняются три свойства:

- рефлексивность:  $a \leq a$ ;
- транзитивность:  $(a \leq b, b \leq c) \Rightarrow (a \leq c)$ ;
- антисимметричность:  $(a \leq b, b \leq a) \Rightarrow (a = b)$ .

**Определение 1.35.** Для  $a, b \in X$  элемент  $c = a \oplus b \in X$  называется наименьшей верхней границей, если:

- $a \leq c, b \leq c$ ;
- для  $d \in X$  истинно  $(a \leq d, b \leq d) \Rightarrow (c \leq d)$ .

**Определение 1.36.** Для  $a, b \in X$  элемент  $c = a \otimes b \in X$  называется наибольшей нижней границей, если:

- $c \leq a, c \leq b$ ;
- для  $d \in X$  истинно  $(d \leq a, d \leq b) \Rightarrow (d \leq c)$ .

Для пары элементов частично упорядоченного множества  $X$  не обязательно существует наименьшая верхняя (наибольшая ниж-

няя) граница, но если она существует, то из антисимметричности следует ее единственность.

**Определение 1.37.** Пусть  $X$  — частично упорядоченное множество.  $(X, \leq)$  называется решеткой, если для любых  $a, b \in X$  существуют  $a \oplus b \in X$  и  $a \otimes b \in X$ .

**Лемма 1.1.** Для любого набора  $S = \{a_1, a_2, \dots, a_n\}$  элементов решетки  $(X, \leq)$  существуют единственные элементы:

$\oplus S = a_1 \oplus a_2 \oplus \dots \oplus a_n$  — наименьшая верхняя граница  $S$ ;

$\otimes S = a_1 \otimes a_2 \otimes \dots \otimes a_n$  — наибольшая нижняя граница  $S$ .

Для решетки  $(X, \leq)$  существует максимальный элемент  $high = \oplus X$  и минимальный элемент  $low = \otimes X$ .

**Определение 1.38.** Линейная решетка (линейная шкала) из  $n$  элементов — это линейное упорядоченное множество; можно всегда считать  $X = \{0, 1, \dots, n\}$ .

Как правило, решетки представляют с помощью ориентированных графов (рис. 1.6). При этом вершинами графа являются элементы множества  $X$  и для  $a_1, a_2 \in X$  справедливо неравенство  $a_1 \leq a_2$ , если в графе существует путь из  $a_1$  в  $a_2$ .

Частным важным случаем решеток является решетка подмножеств некоторого конечного множества  $U$ .

**Определение 1.39.** Пусть  $U$  — конечное множество,  $X = 2^U$  — множество всех подмножеств множества  $U$ . Определим решетку  $(X, \leq)$  с бинарным отношением частичного порядка « $\leq$ », где для  $a, b \subseteq U, a, b \in X$  выполняется условие

$a \leq b$  тогда и только тогда, когда  $a \subseteq b$ .

При этом

$$a \oplus b = a \cup b, a \otimes b = a \cap b.$$

Другим распространенным случаем решеток является решетка многоуровневой безопасности (*MLS*). Данная решетка строится как прямое произведение линейной решетки  $L$  и решетки  $X$  подмножеств множества  $U$ .

**Определение 1.40.** Пусть  $(L, \leq)$  — линейная решетка,  $(X, \leq)$  — решетка подмножеств  $U$ . Определим решетку многоуровневой безопасности  $(X \times L, \leq)$  с бинарным отношением частичного порядка « $\leq$ », где для  $(a, \alpha), (b, \beta) \in X \times L$  выполняется условие

$(a, \alpha) \leq (b, \beta)$  тогда и только тогда, когда  $a \subseteq b, \alpha \leq \beta$ .

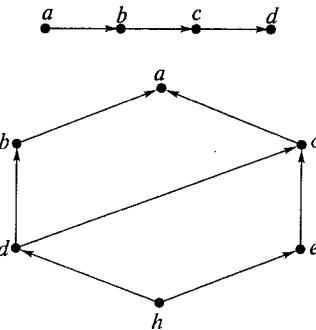


Рис. 1.6. Примеры решеток

При этом

$$(a, \alpha) \oplus (b, \beta) = (a \cup b, \max\{\alpha, \beta\});$$

$$(a, \alpha) \otimes (b, \beta) = (a \cap b, \min\{\alpha, \beta\}).$$

На практике при использовании многоуровневых решеток безопасности решетка  $(L, \leq)$  является линейной шкалой уровней конфиденциальности, а  $(X, \leq)$  — решеткой подмножеств множества неиерархических категорий информации.

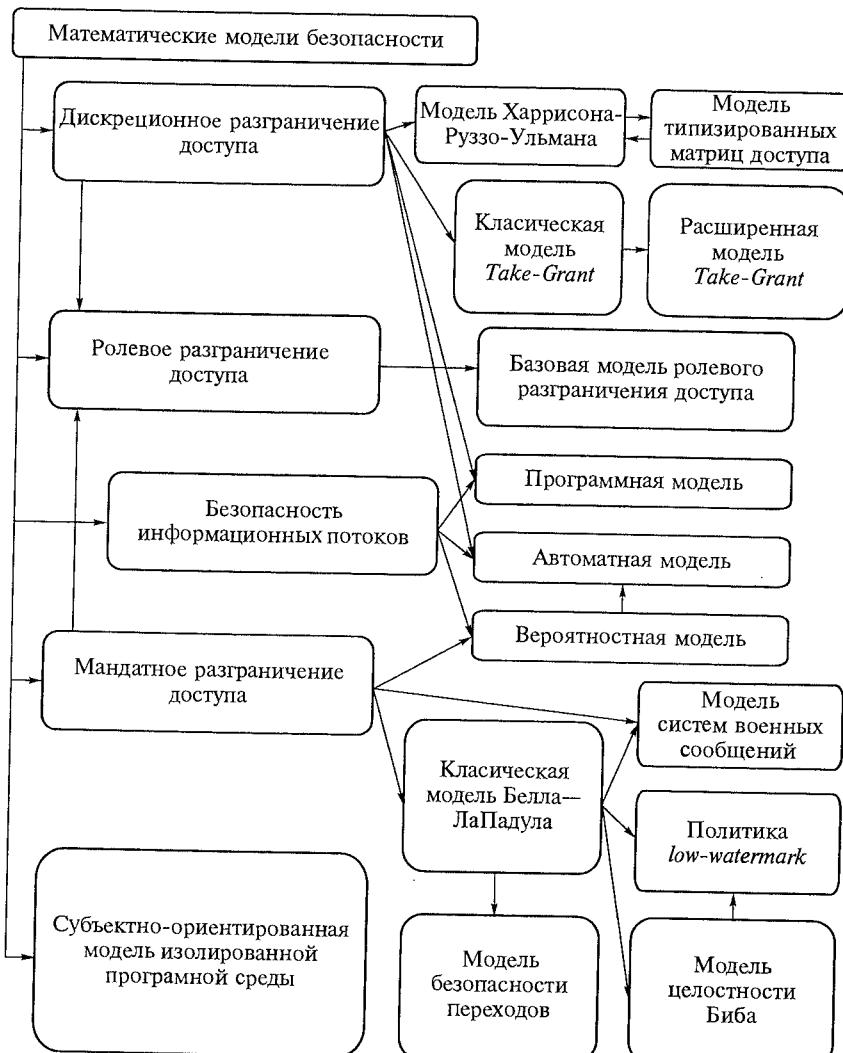


Рис. 1.7. Схема классификации и взаимосвязи положений математических моделей безопасности компьютерных систем

### 1.3. ОСНОВНЫЕ ВИДЫ МОДЕЛЕЙ БЕЗОПАСНОСТИ

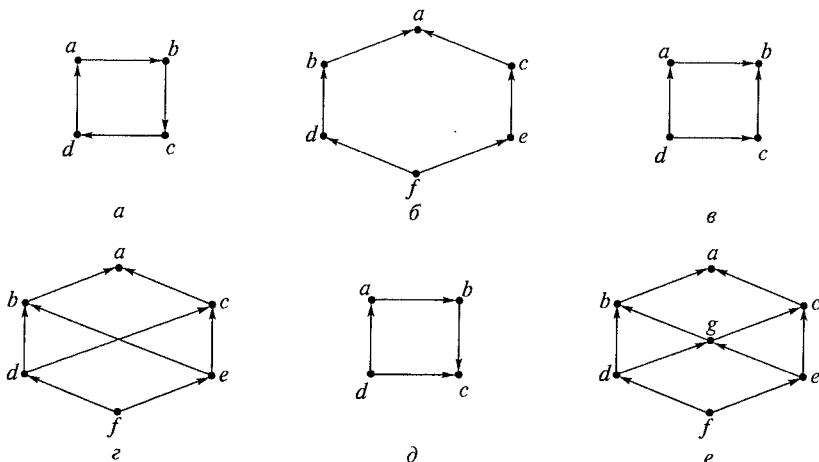
Все математические модели безопасности компьютерных систем, рассматриваемые в данном пособии, классифицируются по пяти основным видам:

- модели систем дискреционного разграничения доступа;
- модели систем мандатного разграничения доступа;
- модели безопасности информационных потоков;
- модели ролевого разграничения доступа;
- субъектно-ориентированная модель изолированной программной среды.

Схема классификации и взаимосвязи положений рассматриваемых математических моделей безопасности компьютерных систем представлены на рис. 1.7.

#### Контрольные вопросы

1. В чем состоит важность основной аксиомы теории защиты информации?
2. Какие основные угрозы безопасности информации рассматриваются в теории защиты информации?
3. Приведите примеры наиболее распространенных в современных операционных системах и системах управления базами данных неблагоприятных информационных потоков по памяти и по времени.
4. Какие основные виды политик безопасности рассматриваются в теории защиты информации?
5. Задают ли решетку следующие графы?



6. Нарисуйте график, соответствующий решетке многоуровневой безопасности  $(X \times L, \leq)$ , для решетки  $(L, \leq) = \{Low, High\}$  и  $(X, \leq)$  — решетки подмножеств множества  $U = \{Political, Economic, Military\}$ .

# МОДЕЛИ СИСТЕМ ДИСКРЕЦИОННОГО РАЗГРАНИЧЕНИЯ ДОСТУПА

## 2.1. МОДЕЛЬ МАТРИЦЫ ДОСТУПОВ ХРУ

### 2.1.1. Элементы модели ХРУ

Модель Харрисона — Руззо — Ульмана (ХРУ) [6, 11, 13] используется для анализа систем защиты, реализующих дискреционную политику безопасности.

Элементами модели ХРУ являются:

$O$  — множество объектов системы;

$S$  — множество субъектов системы ( $S \subseteq O$ );

$R$  — множество видов прав доступа субъектов на объекты, например права на чтение (*read*), на запись (*write*), владения (*own*);

$M$  — матрица доступов, строки которой соответствуют субъектам, а столбцы соответствуют объектам.  $M[s, o] \subseteq R$  — права доступа субъекта  $s$  на объект  $o$ .

**Определение 2.1.** Автомат, построенный согласно положениям модели ХРУ, называется системой ХРУ.

Функционирование системы рассматривается только с точки зрения изменений в матрице доступа. Возможные изменения определяются шестью видами примитивных операторов, представленных в табл. 2.1.

В результате выполнения примитивного оператора  $\alpha$  осуществляется переход из состояния  $q = (S, O, M)$  в результирующее состояние  $q' = (S', O', M')$ . Данный переход обозначим через  $q \vdash_{\alpha} q'$ .

Из примитивных операторов составляются команды. Каждая команда состоит из двух частей:

- условия, при котором выполняется команда;
- последовательности примитивных операторов.

Таким образом, запись команды имеет следующий вид:

```
command c( $x_1, \dots, x_k$ )
  if ( $r_1 \in M[x_{s_1}, x_{o_1}]$ ) and ... and ( $r_m \in M[x_{s_m}, x_{o_m}]$ ) then
     $\alpha_1;$ 
    ...
     $\alpha_n;$ 
  endif
end,
```

где  $r_1, \dots, r_m \in R$  — права доступа;  $\alpha_1, \dots, \alpha_n$  — последовательность примитивных операторов, параметрами которых являются параметры команды  $x_1, \dots, x_k$ . Следует отметить, что наличие условия в теле команды не является обязательным.

При выполнении команды  $c(x_1, \dots, x_k)$  система осуществляет переход из состояния  $q$  в новое состояние  $q'$ . Данный переход обозначим

$$q \vdash_{c(x_1, \dots, x_k)} q'.$$

При этом

$q' = q$ , если одно из условий команды  $c(x_1, \dots, x_k)$  не выполнено;

Таблица 2.1. Примитивные операторы модели ХРУ

Примитивный оператор	Исходное состояние $q = (S, O, M)$	Результирующее состояние $q' = (S', O', M')$
«Внести» право $r \in R$ в $M[s, o]$	$s \in S$ $o \in O$	$S' = S, O' = O,$ $M'[s, o] = M[s, o] \cup \{r\},$ для $(s', o') \neq (s, o)$ справедливо равенство $M'[s', o'] = M[s', o']$
«Удалить» право $r \in R$ из $M[s, o]$	$s \in S$ $o \in O$	$S' = S, O' = O,$ $M'[s, o] = M[s, o] \setminus \{r\},$ для $(s', o') \neq (s, o)$ справедливо равенство $M'[s', o'] = M[s', o']$
«Создать» субъект $s'$	$s' \notin S$	$S' = S \cup \{s'\}, O' = O \cup \{s'\},$ для $(s, o) \in S \times O$ справедливо равенство $M'[s, o] = M[s, o],$ для $o \in O'$ справедливо равенство $M'[s', o] = \emptyset,$ для $s \in S'$ справедливо равенство $M'[s, s'] = \emptyset$
«Создать» объект $o'$	$o' \notin O$	$S' = S, O' = O \cup \{o'\},$ для $(s, o) \in S \times O$ справедливо равенство $M'[s, o] = M[s, o],$ для $s \in S'$ справедливо равенство $M'[s, o'] = \emptyset$
«Уничтожить» субъект $s'$	$s' \in S$	$S' = S \setminus \{s'\}, O' = O \setminus \{s'\},$ для $(s, o) \in S' \times O'$ справедливо равенство $M'[s, o] = M[s, o]$
«Уничтожить» объект $o'$	$o' \in O$ $o' \notin S$	$S' = S, O' = O \setminus \{o'\},$ для $(s, o) \in S' \times O'$ справедливо равенство $M'[s, o] = M[s, o]$

$q' = q_n$ , если условие команды  $c(x_1, \dots, x_k)$  выполнено и существуют состояния  $q_1, \dots, q_n$  такие, что  $q = q_0 \vdash_{\alpha_1} q_1 \vdash_{\alpha_2} \dots \vdash_{\alpha_n} q_n$ .

**Пример 2.1.** Команда создания субъектом  $s$  личного файла  $f$ .

*command* «создать файл»  $(s, f)$ :

- «создать» объект  $f$ ;
- «внести» право владения  $own$  в  $M[s, f]$ ;
- «внести» право на чтение  $read$  в  $M[s, f]$ ;
- «внести» право на запись  $write$  в  $M[s, f]$ ;

*end.*

**Пример 2.2.** Команда передачи субъекту  $s'$  права  $read$  на файл  $f$  его владельцем субъектом  $s$ .

*command* «передать право чтения»  $(s, s', f)$ :

```
if ( $own \in M[s, f]$ ) then
    «внести» право  $read$  в  $M[s', f]$ ;
endif
```

*end.*

## 2.1.2. Анализ безопасности систем ХРУ

Согласно требованиям основных критериев оценки безопасности компьютерных систем [10, 23], их системы защиты должны строиться на основе математических моделей. С использованием математических моделей должно быть теоретически обосновано соответствие системы защиты требованиям заданной политики безопасности. Для решения этой задачи необходим алгоритм, позволяющий осуществлять данную проверку. Однако, как показывают результаты анализа модели ХРУ, задача построения алгоритма проверки безопасности систем, реализующих дискреционную политику разграничения прав доступа, не может быть решена в общем случае.

**Определение 2.2.** Будем считать, что возможна утечка права  $r \in R$  в результате выполнения команды  $c(x_1, \dots, x_k)$ , если при переходе системы  $q \vdash_{c(x_1, \dots, x_k)} q'$  выполняется примитивный оператор, вносящий  $r$  в элемент матрицы доступов  $M$ , до этого  $r$  не содержавший.

**Определение 2.3.** Начальное состояние  $q_0$  называется безопасным по отношению к некоторому праву  $r \in R$ , если не возможен переход системы в такое состояние  $q$ , в котором возможна утечка права  $r$ .

**Определение 2.4.** Система называется монооперационной, если каждая команда системы содержит один примитивный оператор.

**Теорема 2.1.** Существует алгоритм, проверяющий: является ли исходное состояние монооперационной системы безопасным по отношению к праву  $r \in R$ .

**Доказательство.** Для доказательства достаточно показать, что число последовательностей команд монооперационной системы, которые необходимо проверить, ограничено и сами последовательности команд имеют конечную длину. В этом случае алгоритм проверки безопасности есть алгоритм тотального перебора всех последовательностей команд и проверки конечного состояния для каждой из них на отсутствие утечки права  $r$ .

Заметим, что нет необходимости рассматривать в последовательностях команды, содержащие примитивные операторы вида «удалить»... и «уничтожить»..., так как необходимо проверить наличие права доступа, а не его отсутствие. Заметим также, что нет необходимости рассматривать последовательности команд, содержащих более одного примитивного оператора вида «создать»... Это обусловлено тем, что все последовательности команд, которые проверяют или вносят права доступа в новые элементы матрицы доступов  $M$ , могут быть заменой параметров в командах представлены последовательностями, действующими с существующими субъектами и объектами. Тем не менее, необходимо сохранить одну команду создания субъекта (которая должна быть выполнена первой) на случай, если в начальном состоянии системы  $q_0 = (S_0, O_0, M_0)$  множество  $S_0 = \emptyset$ .

Таким образом, надо рассматривать только те последовательности команд, которые содержат примитивные операторы вида «внести»... и максимум один оператор «создать» субъект.

Число различных примитивных операторов «внести»...

$$n = |R|(|S_0| + 1)(|O_0| + 1).$$

Все команды, содержащие один и тот же примитивный оператор, но разные условия, объединяются в алгоритме проверки безопасности в одну команду с общим составным условием (с применением логических операторов *and* и *or*). Таким образом, число последовательностей команд, которые необходимо рассмотреть, равняется  $n!$ ; при этом длина каждой последовательности равна  $n$ . Теорема доказана.

**Следствие 2.1.** Алгоритм проверки безопасности монооперационных систем имеет экспоненциальную сложность.

**Теорема 2.2.** Задача проверки безопасности произвольных систем ХРУ алгоритмически неразрешима.

**Доказательство.** Для доказательства теоремы воспользуемся фактом, доказанным в теории машин Тьюринга [5]: не существует алгоритма проверки для произвольной машины Тьюринга и произвольного начального слова — остановится ли машина Тьюринга в конечном состоянии или нет.

Под машиной Тьюринга понимается способ переработки слов в конечных алфавитах. Слова записываются на бесконечную в обе стороны ленту, разбитую на ячейки.

Для элементов и команд машины Тьюринга используем следующие обозначения:

$A = \{a_0, a_1, \dots, a_m\}$  — внешний алфавит, где  $a_0 = \lambda$  — пустой символ;

$Q = \{q_0, q_1, \dots, q_k\}$  — внутренний алфавит, где  $q_1$  — начальное состояние;  $q_0$  — конечное состояние;

$D = \{r, l, e\}$  — множество действий, где  $r$  — шаг вправо управляющей головки;  $l$  — шаг влево управляющей головки;  $e$  — управляющая головка не перемещается;

$C: Q \times A \rightarrow Q \times A \times D$  — функция, задающая команды машины Тьюринга. Если  $C(q_i, a_i) = (q_{i'}, a_{i'}, d)$ , то это означает, что когда машина находится в состоянии  $q_i$  и управляющая головка указывает на ячейку ленты, содержащую символ  $a_i$ , тогда выполняется шаг машины, в результате которого в эту ячейку записывается символ  $a_{i'}$ , машина переходит в состояние  $q_{i'}$ , а управляющая головка смещается по ленте согласно действию  $d$ .

Для того чтобы воспользоваться указанным выше фактом, представим все элементы и команды машины Тьюринга в виде элементов и команд системы модели ХРУ.

Пусть машина Тьюринга выполнила некоторое число шагов. Пронумеруем все ячейки, пройденные считывающей головкой (включая те, в которые была изначально занесена информация), числами от 1 до  $n$ . Тогда  $(a_{s_1}, \dots, a_{s_n})$  — заполнение ленты, где  $a_{s_i} \in A$ ,  $s_i \in \{0, 1, \dots, m\}$  для  $i = 1, \dots, n$ . Пусть считывающая головка указывает на ячейку с номером  $i \in \{1, \dots, n\}$ , содержащую символ  $a_i \in A$ , где  $i \in \{0, 1, \dots, m\}$ , состояние машины  $q_i \in Q$ , где  $i_j \in \{1, \dots, k\}$ , и должна быть выполнена команда  $C(q_{i'}, a_{i'}) = (q_{i''}, a_{i''}, d)$ , где  $q_{i'} \in Q$ ,  $a_{i''} \in A$ ,  $i_{i'} \in \{0, 1, \dots, m\}$ ,  $i_{i''} \in \{1, \dots, k\}$ ,  $d \in D$ .

Каждой ячейке ленты поставим в соответствие субъект модели ХРУ; при этом будем считать, что  $O = S = \{s_1, \dots, s_n\}$ . Зададим матрицу доступов  $M$  для текущего состояния. Пусть множество прав доступа  $R = Q \cup A \cup \{own, left, right\}$  и в матрицу доступов внесены права:

- $a_{s_j} \in M[s_j, s_j]$  для  $j = 1, \dots, n$  — заполнение ленты;
- $own \in M[s_j, s_{j+1}]$  для  $j = 1, \dots, n - 1$  — упорядочивание субъектов, соответствующих ячейкам ленты;
- $q_i \in M[s_i, s_i]$  — управляющая головка указывает на ячейку с номером  $i$ ;
- $left \in M[s_1, s_1]$  — признак крайней левой из пройденных ячеек на ленте;
- $right \in M[s_n, s_n]$  — признак крайней правой из пройденных ячеек на ленте.

	$s_1$	$s_2$	$s_3$	...	$s_i$	...	$s_{n-1}$	$s_n$
$s_1$	$a_{s_1},$ <i>left</i>	<i>own</i>						
$s_2$		$a_{s_2}$	<i>own</i>					
$s_3$			$a_{s_3}$					
...								
$s_i$						$a_{s_p},$ $q_{i_j}$		
...								
$s_{n-1}$							$a_{s_{n-1}}$	<i>own</i>
$s_n$								$a_{s_n},$ <i>right</i>

Рис. 2.1. Заполнение матрицы доступов модели ХРУ

Таким образом, текущему состоянию машины Тьюринга соответствует матрица доступов  $M$  системы модели ХРУ, представленная на рис. 2.1.

Построим гомоморфизм машины Тьюринга в систему ХРУ. Для каждой команды машины Тьюринга  $C(q_i, a_i) = (q_{i'}, a_{i'}, d)$  зададим соответствующую ей команду модели ХРУ:

- если  $d = e$ , то

*command Eq<sub>i</sub>a<sub>i</sub>q<sub>i</sub>a<sub>i'</sub>(s):*

```
if ( $q_i \in M[s, s]$ ) and ( $a_i \in M[s, s]$ ) then
    «удалить» право  $q_i$  из  $M[s, s]$ ;
    «удалить» право  $a_i$  из  $M[s, s]$ ;
    «внести» право  $q_{i'}$  в  $M[s, s]$ ;
    «внести» право  $a_{i'}$  в  $M[s, s]$ ;
```

*endif*

*end;*

- если  $d = r$ , то необходимо задать две команды для случаев, когда считывающая головка указывает или не указывает на самую правую ячейку ленты:

*command R1q<sub>i</sub>a<sub>i</sub>q<sub>i</sub>a<sub>i'</sub>(s, s'):*

```
if ( $q_i \in M[s, s]$ ) and ( $a_i \in M[s, s]$ ) and ( $own \in M[s, s']$ ) then
    «удалить» право  $q_i$  из  $M[s, s]$ ;
    «удалить» право  $a_{i'}$  из  $M[s, s]$ ;
    «внести» право  $a_{i'}$  в  $M[s, s]$ ;
    «внести» право  $q_{i'}$  в  $M[s', s']$ ;
```

*endif*

*end;*

*command R2q<sub>i</sub>a<sub>i</sub>q<sub>i</sub>a<sub>i</sub>,(s, s'):*

```

if (qi ∈ M[s, s]) and (ai ∈ M[s, s]) and (right ∈ M[s, s]) then
    «удалить» право qi из M[s, s];
    «удалить» право ai из M[s, s];
    «удалить» право right из M[s, s];
    «внести» право ai в M[s, s];
    «создать» субъект s';
    «внести» право own в M[s, s'];
    «внести» право a0 в M[s', s'];
    «внести» право qi в M[s', s'];
    «внести» право right в M[s', s'];
endif
end;
```

- если  $d = l$ , то две команды для этого случая задаются аналогично командам для случая  $d = r$ .

Если машина Тьюринга останавливается в своем конечном состоянии  $q_0$ , то в соответствующей системе ХРУ происходит утечка права доступа  $q_0$ . Из алгоритмической неразрешимости задачи проверки: остановится ли машина Тьюринга в конечном состоянии, — следует аналогичный вывод для задачи проверки безопасности соответствующей ей системе ХРУ. Таким образом, в общем случае для систем дискреционного разграничения доступа, построенных на основе модели ХРУ, задача проверки безопасности алгоритмически неразрешима. Теорема доказана.

Приведенные выше теоремы 2.1 и 2.2 указывают на имеющуюся проблему выбора у разработчиков систем защиты. С одной стороны, общая модель ХРУ может выражать большое разнообразие политик дискреционного разграничения доступа, но при этом не предоставляет алгоритма проверки их безопасности. С другой стороны, можно использовать монооперационные системы, для которых алгоритм проверки безопасности существует, но данный класс систем является слишком узким. Например, монооперационные системы не могут выразить политику, дающую субъектам право владения на созданные ими объекты, так как не существует одного примитивного оператора, который одновременно и создает объект, и помечает его как принадлежащий создающему субъекту.

Дальнейшие исследования модели ХРУ велись в основном в направлении определения условий, которым должна удовлетворять система, чтобы для нее задача проверки безопасности была алгоритмически разрешимой. Так, в 1976 г. было доказано [13], что задача проверки безопасности алгоритмически разрешима для систем, в которых нет примитивных операторов вида «создать»... В 1978 г. было установлено [14], что таковыми могут быть системы монотонные и моноусловные, т. е. системы, команды которых не содержат операторов вида «уничтожить»... или «удалить»... и содержат условия, состоящие из не более одной проверки вида

$r \in M[s, o]$ . В том же году было установлено [17], что задача проверки безопасности для систем с конечным множеством субъектов разрешима, но вычислительно сложна.

### 2.1.3. Модель типизированной матрицы доступов

Другая дискреционная модель, получившая название «Типизированная матрица доступов» (ТМД), представляет собой развитие модели ХРУ, дополненной концепцией типов, что позволяет несколько смягчить те условия, для которых возможно доказательство безопасности системы. Рассмотрим модель ТМД на основе [3] и [22].

Формальное описание модели ТМД включает в себя следующие элементы:

$O$  — множество объектов системы;  
 $S$  — множество субъектов системы ( $S \subseteq O$ );  
 $R$  — множество прав доступа субъектов к объектам;  
 $M$  — матрица доступов;  
 $C$  — множество команд;  
 $T$  — множество типов объектов;  
 $t: O \rightarrow T$  — функция, ставящая в соответствие каждому объекту некоторый тип;  
 $q = (S, O, t, M)$  — состояние системы;  
 $Q$  — множество состояний системы.

Состояние системы изменяется с помощью команд из множества  $C$ . Команды ТМД имеют тот же формат, что и в модели ХРУ, но всем параметрам приписывается определенный тип:

```

command c(x1; t1, ..., xk; tk)
if (r1 ∈ M[xs1, xo1]) and ... and (rm ∈ M[xsm, xom]) then
    α1;
    ...
    αn;
endif
end.
```

Перед выполнением команды происходит проверка типов фактических параметров. Если они не совпадают с указанными в определении, то команда не выполняется. В модели используются шесть видов примитивных операторов, отличающихся от аналогичных операторов модели ХРУ только использованием типизированных параметров (табл. 2.2).

Таким образом, ТМД является обобщением модели ХРУ, которую можно рассматривать как частный случай ТМД с одним единственным типом для всех объектов и субъектов. С другой стороны, любую систему ТМД можно выразить через систему ХРУ, введя для обозначения типов специальные права доступа, а про-

Таблица 2.2. Примитивные операторы модели ТМД

Оператор	Исходное состояние $q = (S, O, M)$	Результатирующее состояние $q' = (S', O', M')$
«Внести» право $r \in R$ в $M[s, o]$	$s \in S$ $o \in O$	$S' = S, O' = O, t'(o) = t(o)$ для $o \in O, M'[s, o] = M[s, o] \cup \{r\}$ , для $(s', o') \neq (s, o)$ справедливо равенство $M'[s', o'] = M[s', o']$
«Удалить» право $r \in R$ из $M[s, o]$	$s \in S$ $o \in O$	$S' = S, O' = O, t'(o) = t(o)$ для $o \in O, M'[s, o] = M[s, o] \setminus \{r\}$ , для $(s', o') \neq (s, o)$ справедливо равенство $M'[s', o'] = M[s', o']$
«Создать» субъект $s'$ с типом $t_s$	$s' \notin S$	$S' = S \cup \{s'\}, O' = O \cup \{s'\}$ , для $o \in O$ справедливы равенства $t'(o) = t(o), t'(s') = t_s$ , для $(s, o) \in S \times O$ справедливо равенство $M'[s, o] = M[s, o]$ , для $o \in O'$ справедливо равенство $M'[s', o] = \emptyset$ , для $s \in S'$ справедливо равенство $M'[s, s'] = \emptyset$
«Создать» объект $o'$ с типом $t_o$	$o' \notin O$	$S' = S, O' = O \cup \{o'\}, t'(o') = t_o$ , для $(s, o) \in S \times O$ справедливо равенство $M'[s, o] = M[s, o]$ , для $s \in S'$ справедливо равенство $M'[s, o'] = \emptyset$
«Уничтожить» субъект $s'$	$s' \in S$	$S' = S \setminus \{s'\}, O' = O \setminus \{s'\}$ , для $o \in O'$ справедливо равенство $t'(o) = t(o)$ , для $(s, o) \in S' \times O'$ справедливо равенство $M'[s, o] = M[s, o]$
«Уничтожить» объект $o'$	$o' \in O$ $o' \notin S$	$S' = S, O' = O \setminus \{o'\}$ , для $o \in O'$ справедливо равенство $t'(o) = t(o)$ , для $(s, o) \in S' \times O'$ справедливо равенство $M'[s, o] = M[s, o]$

верку типов в командах заменив проверкой наличия соответствующих прав доступа.

Появление в каждой команде дополнительных неявных условий, ограничивающих область применения команды только объектами соответствующих типов, позволяет несколько смягчить же-

сткие условия классической модели, при которых критерий безопасности является разрешимым.

**Определение 2.5.** Модель монотонной типизированной матрицы доступов (МТМД) — модель ТМД, в командах которой отсутствуют немонотонные примитивные операторы вида «удалить»... и «уничтожить»...

**Определение 2.6.** Каноническая форма модели МТМД (КФМТМД) — модель МТМД, в которой команды, содержащие примитивные операторы вида «создать»..., не содержат условий и примитивных операторов вида «внести»...

**Теорема 2.3.** Любая система МТМД эквивалента системе КФМТМД.

**Доказательство.** Пусть задана система МТМД, в которой определены множества  $R, T, Q, C$ . Построим эквивалентную ей систему КФМТМД, определив множества  $R^*, T^*, Q^*, C^*$ .

Пусть

$$R^* = R \cup \{\text{active}\};$$

$$T^* = T \cup \{t_{\text{active}}\}.$$

В каждом состоянии  $q^* = (S^*, O^*, t^*, M^*)$ , соответствующем состоянию  $q = (S, O, t, M)$ , справедливы равенства:

$$S^* = S \cup \{s_{\text{active}}\};$$

$$O^* = O \cup \{s_{\text{active}}\}.$$

Пусть также для каждого  $o \in O$  справедливо равенство  $t^*(o) = t(o)$  и  $s_{\text{active}}$  — единственный субъект такой, что  $t^*(s_{\text{active}}) = t_{\text{active}}$ . Кроме того, для  $s \in S, o \in O$  справедливо равенство  $M^*[s, o] = M[s, o]$  и в начальном состоянии системы для  $o \in O_0$  справедливо равенство  $M^*[s_{\text{active}}, o] = \{\text{active}\}$ .

Таким образом, право доступа *active* обозначает активизированные субъекты и объекты КФМТМД.

Каждую команду  $c(x_1; t_1, \dots, x_k; t_k)$  системы МТМД, не содержащую примитивные операторы «создать»..., представим командой  $c(x_1; t_1, \dots, x_k; t_k, s; t_{\text{active}})$  системы КФМТМД, полученной из исходной команды добавлением условий проверки  $\text{active} \in M[s, x_i]$ , для  $i = 1, \dots, k$ .

Каждую команду  $c(x_1; t_1, \dots, x_k; t_k)$  системы МТМД, содержащую примитивные операторы «создать»..., представим двумя монотонными командами КФМТМД:

- $c'(x_1; t_1, \dots, x_k; t_k)$  — команда без проверки условий, содержащая все примитивные операторы «создать»... команды  $c(x_1; t_1, \dots, x_k; t_k)$ ;

- $c''(x_1; t_1, \dots, x_k; t_k, s; t_{\text{active}})$  — команда, содержащая условия и примитивные операторы «внести»... команды  $c(x_1; t_1, \dots, x_k; t_k)$ , условия проверки  $\text{active} \in M[s, x_i]$ , для всех родительских (не создаваемых в команде  $c(x_1; t_1, \dots, x_k; t_k)$ ) объектов  $x_i$ , примитивные операторы «внести» право *active* в  $M[s, x_i]$  для всех объектов  $x_i$ , создаваемых в команде  $c(x_1; t_1, \dots, x_k; t_k)$ .

Таким образом, только «активизированные» объекты (в том числе и субъекты) системы КФМТМД соответствуют объектам системы МТМД, а все преобразования над ними в системе КФМТМД соответствуют преобразованиям системы МТМД. Теорема доказана.

Для того чтобы сформулировать ограничения, необходимые для алгоритмической разрешимости задачи проверки безопасности в системах МТМД, определим отношения между типами.

**Определение 2.7.** Пусть  $c(x_1: t_1, \dots, x_k: t_k)$  — некоторая команда ТМД. Будем говорить, что  $t_i$  является дочерним типом в  $c(x_1: t_1, \dots, x_k: t_k)$ , если в ее теле имеется один из следующих примитивных операторов:

- «создать» субъект  $s$  с типом  $t_i$ ;
- «создать» объект  $o$  с типом  $t_i$ .

В противном случае будем говорить, что  $t_i$  является родительским типом в  $c(x_1: t_1, \dots, x_k: t_k)$ .

Заметим, что в одной команде тип может быть одновременно и родительским, и дочерним. Например:

```
command foo(s1: u, s2: u, s3: v, o1: w, o2: b)
    «создать» субъект  $s_2$  с типом  $u$ ;
    «создать» субъект  $s_3$  с типом  $v$ ;
end.
```

Здесь  $u$  является родительским типом относительно  $s_1$  и дочерним типом относительно  $s_2$ . Кроме того,  $w$  и  $b$  являются родительскими типами, а  $v$  — дочерним типом.

Тогда можно описать взаимосвязи между различными типами с помощью графа, определяющего отношение «наследственности» между типами, устанавливаемые через операции порождения объектов.

**Определение 2.8.** Граф создания — ориентированный граф с множеством вершин  $T$ , в котором ребро от  $u$  к  $v$  существует тогда и только тогда, когда в системе имеется команда, в которой  $u$  является родительским типом, а  $v$  — дочерним типом.

Граф создания для каждого типа позволяет определить:

- объекты каких типов должны существовать в системе, чтобы в ней мог появиться объект или субъект заданного типа;
- объекты каких типов могут быть порождены при участии объектов заданного типа.

**Определение 2.9.** Система МТМД (КФМТМД) называется ациклической (АМТМД или, соответственно, АКФМТМД) тогда и только тогда, когда ее граф создания не содержит циклов; в противном случае говорят, что система является циклической.

Например, граф создания для приведенной выше команды  $foo()$ , содержит следующие ребра:  $\{(u, u), (u, v), (w, u), (w, v), (b, u), (b, v)\}$ . Система МТМД, содержащая эту команду, будет циклической, поскольку тип  $u$  является для нее одновременно и

родительским, и дочерним, что приводит к появлению в графе цикла  $(u, u)$ .

Алгоритм проверки безопасности систем АМТМД будет строиться для эквивалентных им систем АКФМТМД. Он состоит из двух шагов.

**Шаг 1.** Используя команды, содержащие только примитивные операторы «создать»... и не содержащие условий, перейти из начального состояния системы в некоторое развернутое состояние, обеспечивающее минимально необходимый и достаточный для распространения прав доступа состав объектов.

**Шаг 2.** Используя команды, не содержащие примитивные операторы «создать»..., перейти из развернутого состояния в замкнутое состояние, в котором дальнейшее применение таких команд не приведет к изменениям в матрице доступов.

Второй шаг алгоритма, очевидно, всегда будет иметь конечную сложность. Так как множество объектов не изменяется, то необходимо перебрать все последовательности различных команд (по аналогии с доказательством теоремы 2.1), которых конечное число.

Наибольшую трудность в общем случае представляет разработка алгоритма построения развернутого состояния. Однако для АМТМД или АКФМТМД такой алгоритм существует. Перед его рассмотрением дадим определение.

**Определение 2.10.** Пусть  $\alpha$  и  $\beta$  — две различные команды системы МТМД, содержащие примитивные операторы «создать»... Будем считать, что  $\alpha < \beta$  тогда и только тогда, когда для каждого дочернего типа в  $\alpha$  в графе создания найдется путь в некоторый родительский тип в  $\beta$ .

**Лемма 2.1.** В АМТМД отношение « $<$ » на множестве команд является отношением строгого порядка (обладает свойствами антирефлексивности, антисимметричности и транзитивности).

*Доказательство.* Выполняется по определению отношения « $<$ » и АМТМД.

**Алгоритм 2.1.** Алгоритм построения развернутого состояния для АКФМТМД.

**Шаг 1.** Упорядочить линейно в списке все команды, содержащие примитивные операторы вида «создать»... (команда  $\alpha$  следует в списке перед командой  $\beta$  тогда и только тогда, когда или  $\alpha < \beta$ , или  $\alpha$  и  $\beta$  несравнимы).

**Шаг 2.** Начиная с начального состояния, применять по созданному на шаге 1 списку все команды; при этом каждая команда применяется со всеми возможными для нее наборами родительских объектов.

**Лемма 2.2.** Алгоритм 2.1 заканчивает работу за конечное время на любом начальном состоянии произвольной АКФМТМД.

*Доказательство.* Число команд в списке конечно. Для каждой команды из списка на каждом шаге работы алгоритма существует

конечный набор объектов, которые могут являться ее параметрами. Следовательно, алгоритм 2.1 всегда заканчивает работу за конечное время. Лемма доказана.

**Определение 2.11.** Пусть  $(q_0, q_1, \dots, q_i, \dots)$  — некоторая история АКФМТМД. Для каждого состояния  $q_i = (S_i, O_i, t_i, T)$ ,  $i = 0, 1, \dots$ , на множестве  $O_i$  рекурсивно определим функцию порождения объектов  $\pi()$ :

- для каждого  $o \in O_0 \subset O_i$  зададим  $\pi(o) = o$ ,
- если объект  $o \in O_k \subset O_i$  создан на шаге  $0 < k \leq i$  истории командой  $\alpha_k$ , где  $o_1, o_2, \dots, o_m \in O_{k-1}$  — последовательность входящих в нее объектов родительских типов, то зададим  $\pi(o) = \alpha_k(\pi(o_1), \pi(o_2), \dots, \pi(o_m))$ .

**Пример 2.3.** Пусть система АКФМТМД с двумя командами:

*command cv(x: u, y: v)*  
 «создать» субъект  $y$  с типом  $v$ ;  
*end,*  
*command cw(x: u, y: v, z: w)*  
 «создать» объект  $z$  с типом  $w$ ;

*end,*

и история имеет вид  $q_0 \vdash_{cv(x, y)} q_1 \vdash_{cw(x, y, z)} q_2$ , где  $q_0 = (S_0, O_0, t_0, T) = (\{x\}, \{x\}, \{(x, u)\}, \{u, v, w\})$ . Тогда в состоянии  $q_2 = (S_2, O_2, t_2, T) = (\{x, y\}, \{x, y, z\}, \{(x, u), (y, v), (z, w)\}, \{u, v, w\})$  функция  $\pi()$  будет иметь следующие значения:

$$\begin{aligned}\pi(x) &= x, \\ \pi(y) &= cv(x), \\ \pi(z) &= cw(x, cv(x)).\end{aligned}$$

**Лемма 2.3.** В развернутом состоянии АКФМТМД для каждого возможного значения функции порождения объектов  $\pi()$  существует только один объект.

**Доказательство.** Развернутое состояние для АКФМТМД строится по алгоритму 2.1 за конечное число шагов; при этом, по определению шага 2, оно содержит объекты для всех возможных значений функции  $\pi()$ . В то же время, на каждом шаге построения по алгоритму 2.1 получается новый объект с новым значением функции  $\pi()$ . Лемма доказана.

**Теорема 2.4.** Существует алгоритм проверки безопасности системы АКФМТМД.

**Доказательство.** Без ограничения общности по теореме 2.3 будем рассматривать АКФМТМД.

Пусть  $f$  — начальное состояние системы;  $q$  — развернутое состояние системы, полученное из  $f$  по алгоритму 2.1;  $m$  — замкнутое состояние системы, полученное из  $q$  с использованием всех команд, не содержащих примитивные операторы вида «создать»...; при этом дальнейшее применение таких команд к  $m$  не приводит к изменениям в матрице доступов.

Так как система монотонная, то для  $(s, o) \in S_q \times O_q$  выполняется условие

$$M_q[s, o] \subseteq M_m[s, o]. \quad (2.1)$$

Покажем, что для систем АКФМТМД для любой истории  $(f, \dots, h, \dots)$  может быть найдена история  $(q, \dots, g, \dots)$  без команд, содержащих примитивные операторы вида «создать»..., где для  $(s, o) \in S_h \times O_h$  выполняется условие

$$M_h[s, o] \subseteq M_g[\pi(s), \pi(o)]. \quad (2.2)$$

Если условие (2.2) выполняется, то из (2.1) и (2.2) следует, что для  $(s, o) \in S_f \times O_f$  выполняется условие  $M_h[s, o] \subseteq M_g[\pi(s), \pi(o)] \subseteq M_m[\pi(s), \pi(o)]$ . Так как  $f$  — начальное состояние системы, то заменяем  $\pi(s)$  на  $s$ ,  $\pi(o)$  на  $o$  и получаем, что для  $(s, o) \in S_f \times O_f$  выполняется условие

$$M_h[s, o] \subseteq M_g[s, o] \subseteq M_m[s, o],$$

что и означает алгоритмическую разрешимость задачи проверки безопасности систем АКФМТМД.

Обоснуйм (2.2), для чего построим алгоритм преобразования последовательности команд истории  $(f, \dots, h, \dots)$  в последовательность команд истории  $(q, \dots, g, \dots)$ .

**Шаг 1.** В командах истории  $(f, \dots, h, \dots)$  удалить примитивные операторы вида «создать»...

**Шаг 2.** Команды  $c(x_1: t_1, \dots, x_k: t_k)$  истории  $(f, \dots, h, \dots)$  заменить на команды  $c(\pi(x_1): t_1, \dots, \pi(x_k): t_k)$  истории  $(q, \dots, g, \dots)$ .

По индукции по длине истории  $(f, \dots, h, \dots)$  легко показать, что выполняются два условия.

**Условие 1.** Условие каждой команды истории  $(q, \dots, g, \dots)$  истинно тогда и только тогда, когда истинно условие соответствующей ей команды истории  $(f, \dots, h, \dots)$ .

**Условие 2.** Для состояния  $h$  истории  $(f, \dots, h, \dots)$  и соответствующего ему состояния  $g$  истории  $(q, \dots, g, \dots)$  выполняется условие для  $(s, o) \in S_h \times O_h$  истинно  $(r \in M_h[s, o]) \Rightarrow (r \in M_g[\pi(s), \pi(o)])$ .

Таким образом, (2.2) обосновано. Теорема доказана.

**Следствие 2.2.** Алгоритм проверки безопасности систем АКФМТМД имеет экспоненциальную сложность.

**Доказательство.** При построении замкнутого состояния системы из развернутого состояния используется алгоритм, аналогичный алгоритму, использованному в теореме 2.1. По следствию 2.1 получаем, что алгоритм проверки безопасности систем АКФМТМД имеет экспоненциальную сложность.

Определим подмножество АКФМТМД, называемое тернарными АКФМТМД.

**Определение 2.12.** Тернарными называются АКФМТМД, в которых каждая команда имеет не более трех параметров.

Для тернарной АМТМД доказано [8], что алгоритм проверки ее безопасности имеет полиномиальную сложность.

Таким образом, введение строгого контроля типов в дискреционную модель ХРУ позволило доказать критерий безопасности систем для более приемлемых ограничений, что существенно расширило область ее применения.

## 2.2. МОДЕЛЬ РАСПРОСТРАНЕНИЯ ПРАВ ДОСТУПА TAKE-GANT

### 2.2.1. Основные положения классической модели Take-Grant

Классическая модель *Take-Grant* ориентирована на анализ путей распространения прав доступа в системах дискреционного разграничения доступа. Классическую модель *Take-Grant* рассмотрим на основе [6].

Основные элементы модели *Take-Grant*:

$O$  — множество объектов;

$S \subseteq O$  — множество субъектов;

$R = \{r_1, r_2, \dots, r_m\} \cup \{t, g\}$  — множество видов прав доступа, где  $t$  (*take*) — право брать права доступа;  $g$  (*grant*) — право давать права доступа;

$G = (S, O, E)$  — конечный помеченный ориентированный граф без петель, представляющий текущие доступы в системе. Элементы множеств  $S$  и  $O$  являются вершинами графа, которые соответственно будем обозначать:  $\otimes$  — объекты (элементы множества  $O \setminus S$ ),  $\bullet$  — субъекты (элементы множества  $S$ ). Элементы множества  $E \subseteq O \times O \times R$  являются ребрами графа. Каждое ребро помечено непустым подмножеством множества видов прав доступа  $R$ .

Состояние системы описывается соответствующим ему графом доступов. В отличие от модели ХРУ в модели *Take-Grant* возможно наличие прав доступа не только у субъектов на объекты, но и у объектов на объекты.

Основная цель классической модели *Take-Grant* — определение и обоснование алгоритмически проверяемых условий проверки возможности утечки права доступа по исходному графу доступов, соответствующего некоторому состоянию системы.

Порядок перехода системы модели *Take-Grant* из состояния в состояние определяется правилами преобразования графа доступов, которые в классической модели носят название де-юре правил. Преобразование графа  $G$  в граф  $G'$  в результате выполнения правила  $op$  обозначим следующим образом:

$$G \vdash_{op} G'.$$

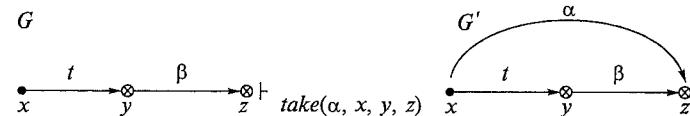


Рис. 2.2. Применение правила  $take(\alpha, x, y, z)$

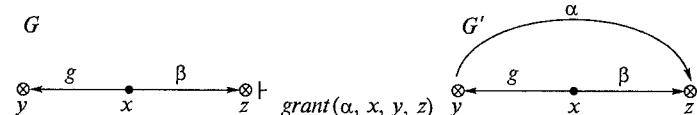


Рис. 2.3. Применение правила  $grant(\alpha, x, y, z)$

В классической модели *Take-Grant* рассматриваются четыре де-юре правила преобразования графа, выполнение каждого из которых может быть инициировано только субъектом, являющимся активной компонентой системы (рис. 2.2 — 2.5):

- *take()* — брать права доступа;
- *grant()* — давать права доступа;
- *create()* — создавать новый объект или субъект; при этом субъект-создатель может взять на созданный субъект любые права доступа (по умолчанию предполагается, что при выполнение правила *create()* создается объект; случаи, когда создается субъект, оговариваются особо);
- *remove()* — удалять права доступа.

Рассмотрим условия применения де-юре правил в исходном состоянии  $G = (S, O, E)$  и результаты их применения в результирующем состоянии  $G' = (S', O', E')$  (табл. 2.3).

В модели *Take-Grant* основное внимание уделяется определению условий, при которых в системе возможно распространение прав доступа определенным способом. Рассмотрим условия реализации:

- способа санкционированного получения прав доступа;
- способа похищения прав доступа.

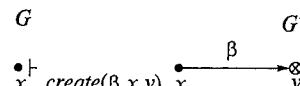


Рис. 2.4. Применение правила  $create(\beta, x, y)$

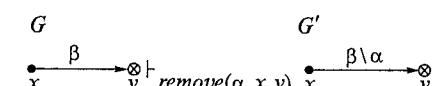


Рис. 2.5. Применение правила  $remove(\alpha, x, y)$

Таблица 2.3. Де-юре правила классической модели *Take-Grant*

Правила	Исходное состояние $G = (S, O, E)$	Результирующее состояние $G' = (S', O', E')$
<i>take</i> ( $\alpha, x, y, z$ )	$x \in S$ , $(x, y, \{t\}) \subset E$ , $(y, z, \beta) \subset E$ , $x \neq z, \alpha \subseteq \beta$	$S' = S$ , $O' = O$ , $E' = E \cup \{(x, z, \alpha)\}$
<i>grant</i> ( $\alpha, x, y, z$ )	$x \in S$ , $(x, y, \{g\}) \subset E$ , $(x, z, \beta) \subset E$ , $y \neq z$ , $\alpha \subseteq \beta$	$S' = S$ , $O' = O$ , $E' = E \cup \{(y, z, \alpha)\}$
<i>create</i> ( $\beta, x, y$ )	$x \in S$ , $y \notin O$ , $\beta \neq \emptyset$	$O' = O \cup \{y\}$ , $S' = S \cup \{y\}$ (если $y$ — субъект), $E' = E \cup \{(x, y, \beta)\}$
<i>remove</i> ( $\alpha, x, y$ )	$x \in S$ , $(x, y, \beta) \subset E$ , $\alpha \subseteq \beta$	$S' = S$ , $O' = O$ , $E' = E \setminus \{(x, y, \alpha)\}$

### Санкционированное получение прав доступа

Данный способ характеризуется тем, что при передаче прав доступа не накладывается ограничений на кооперацию субъектов системы, участвующих в этом процессе.

**Определение 2.13.** Пусть  $x, y \in O_0$ ,  $x \neq y$  — различные объекты графа доступов  $G_0 = (S_0, O_0, E_0)$ ,  $\alpha \subseteq R$ . Определим предикат «возможен доступ»( $\alpha, x, y, G_0$ ), который будет истинным тогда и только тогда, когда существуют графы  $G_1 = (S_1, O_1, E_1)$ , ...,  $G_N = (S_N, O_N, E_N)$  и правила  $op_1, \dots, op_N$  такие, что  $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \dots \vdash_{op_N} G_N$  и  $(x, y, \alpha) \subset E_N$ .

Определение истинности предиката «возможен доступ»() непосредственно по определению является в общем случае алгоритмически неразрешимой задачей, так как требует проверки всех траекторий функционирования системы. По этой причине для проверки истинности предиката «возможен доступ»() следует определить необходимые и достаточные условия, проверка которых возможна. Решение данной задачи будет выполнено в два этапа. На первом этапе будут определены и обоснованы условия истинности предиката «возможен доступ»() для графов, все вершины которых являются субъектами, на втором этапе условия истинно-

сти предиката «возможен доступ»() будут определены и обоснованы для произвольных графов.

**Определение 2.14.** Пусть  $G = (S, S, E)$  — граф доступов. Говорят, что вершины графа доступов являются *tg*-связными или что они соединены *tg*-путем, если, без учета направления ребер, в графе между ними существует путь такой, что каждое ребро этого пути помечено *t* или *g*.

**Теорема 2.5.** Пусть  $G_0 = (S_0, S_0, E_0)$  — граф доступов, содержащий только вершины-субъекты,  $x, y \in S_0$ ,  $x \neq y$ . Тогда предикат «возможен доступ»( $\alpha, x, y, G_0$ ) истинен тогда и только тогда, когда выполняются условия 1 и 2.

**Условие 1.** Существуют субъекты  $s_1, \dots, s_m \in S_0$ :

$$(s_i, y, \gamma_i) \subset E_0, \text{ где } i = 1, \dots, m \text{ и } \alpha = \gamma_1 \cup \dots \cup \gamma_m.$$

**Условие 2.** Субъекты  $x$  и  $s_i$  *tg*-связны в графе  $G_0$ , где  $i = 1, \dots, m$ .

**Доказательство.** Проведем доказательство теоремы для  $m = 1$ , так как схему доказательства для этого случая легко продолжить для случая  $m > 1$ .

При  $m = 1$  условия 1 и 2 формулируются следующим образом.

**Условие 1.** Существует субъект  $s$  такой, что выполняется включение  $(s, y, \alpha) \subset E_0$ .

**Условие 2.** Субъекты  $x$  и  $s$  соединены *tg*-путем в графе  $G_0$ .

**Достаточность.** Пусть выполнены условия 1 и 2. Доказательство проведем индукцией по длине *tg*-пути, соединяющего субъекты  $x$  и  $s$ .

Пусть  $N = 0$ . Тогда  $x = s$ ,  $(x, y, \alpha) \subset E_0$  и предикат «возможен доступ»( $\alpha, x, y, G_0$ ) истинен.

Пусть  $N = 1$ . Тогда существует  $(s, y, \alpha) \subset E_0$  и  $x$  и  $s$  соединены ребром *t* или *g* в графе  $G_0$ . Возможны четыре случая такого соединения  $x$  и  $s$ , для каждого из которых указана последовательность преобразований графа, требуемая для передачи прав доступа (рис. 2.6 — 2.9).

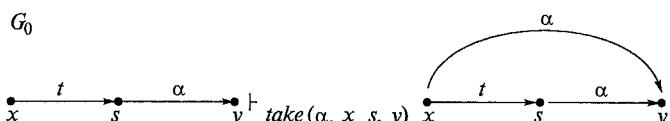


Рис. 2.6. Первый случай

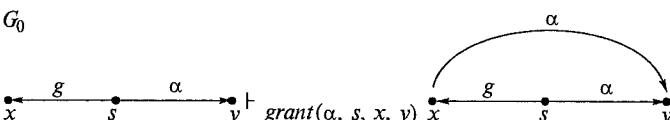


Рис. 2.7. Второй случай

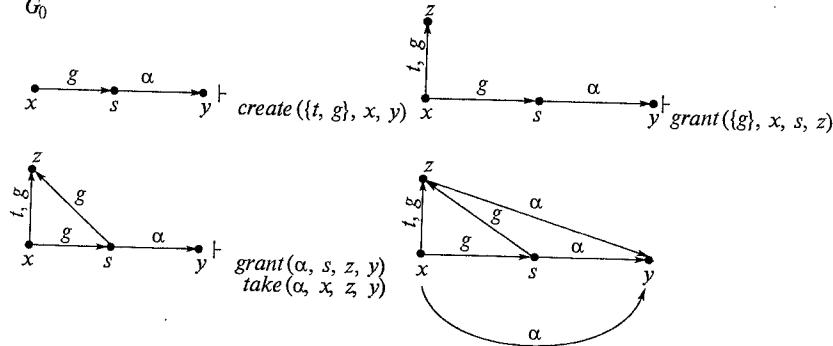


Рис. 2.8. Третий случай

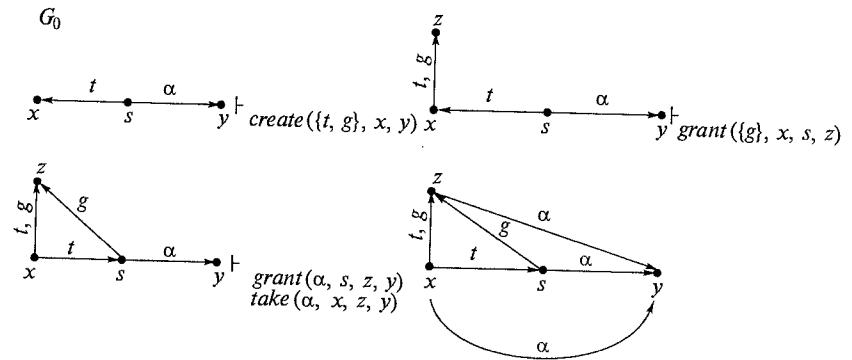


Рис. 2.9. Четвертый случай

Пусть  $N > 1$ . Пусть вершина  $z$  находится на  $tg$ -пути между  $x$  и  $s$  и является смежной с  $s$  в графе  $G_0$ . Тогда исходя из доказанного для случая  $N=1$  существует последовательность преобразований графов доступов  $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \dots \vdash_{op_k} G_K$  такая, что  $(z, y, \alpha) \subset E_K$ , и длина  $tg$ -пути между  $z$  и  $x$  равна  $N-1$ . Это позволяет применить предположение индукции. Достаточность условий теоремы доказана.

**Необходимость.** Пусть истинен предикат «возможен доступ»  $(\alpha, x, y, G_0)$ .

По определению истинности предиката существует последовательность графов доступов  $G_1 = (S_1, O_1, E_1), \dots, G_N = (S_N, O_N, E_N)$  такая, что  $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \dots \vdash_{op_N} G_N$  и  $(x, y, \alpha) \subset E_N$ . Среди всех таких последовательностей выберем ту, у которой длина  $N$  является минимальной. В этом случае  $(x, y, \alpha) \subset E_{N-1}$ .

Докажем необходимость условий 1 и 2 индукцией по  $N$ .

При  $N=0$  очевидно, что  $(x, y, \alpha) \subset E_0$ . Таким образом,  $s=x$  и условия теоремы выполнены.

При  $N=1$  из определения де-юре правил модели следует, что существует субъект  $s$  такой, что выполняется включение  $(s, y, \alpha) \subset E_0$  и или  $op_1 = take(\alpha, x, s, y)$ , или  $op_1 = grant(\alpha, s, x, y)$ . Таким образом,  $s$  и  $x$   $tg$ -связны в графе  $G_0$  и условия теоремы выполнены.

Пусть  $N > 0$  и утверждение теоремы истинно для  $k < N$ . Тогда  $(x, y, \alpha) \subset E_{N-1}$  и ребро  $(x, y, \alpha)$  появляется в графе доступов  $G_N$  в результате применения к графу  $G_{N-1}$  некоторого правила  $op_N$ . Очевидно, что это не правила  $create()$  или  $remove()$ . Следовательно, существует субъект  $s' \in S_{N-1}$  такой, что  $(s', y, \alpha) \subset E_{N-1}$  и или  $(x, s', t) \in E_{N-1}$  и  $op_N = take(\alpha, x, s', y)$ , или  $(s', x, g) \in E_{N-1}$  и  $op_N = grant(\alpha, s', x, y)$ .

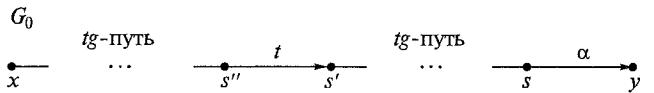
Возможны два случая:  $s' \in S_0$  и  $s' \notin S_0$ .

Рассмотрим первый случай. Пусть  $s' \in S_0$ , тогда истинен предикат «возможен доступ»  $(\alpha, s', y, G_0)$ ; при этом число преобразований графов меньше  $N$ . Следовательно, по предположению индукции существует  $s \in S_0$ :  $(s, y, \alpha) \subset E_0$  и  $s'$  соединен с  $s$   $tg$ -путем в графе  $G_0$ . Кроме того, если  $op_N = take(\alpha, x, s', y)$ , то истинен предикат «возможен доступ»  $(\{t\}, x, s', G_0)$ , а если  $op_N = grant(\alpha, s', x, y)$ , то истинен предикат «возможен доступ»  $(\{g\}, s', x, G_0)$ ; при этом число преобразований графов меньше  $N$ . Следовательно, по предположению индукции существует  $s'' \in S_0$  такой, что если  $op_N = take(\alpha, x, s', y)$ , то  $(s'', s', t) \in E_0$  и  $s''$  соединен с  $x$   $tg$ -путем в графе  $G_0$ , а если  $op_N = grant(\alpha, s', x, y)$ , то  $(s'', x, g) \in E_0$  и  $s''$  соединен с  $s'$   $tg$ -путем в графе  $G_0$ . Таким образом, существует  $s \in S_0$ :  $(s, y, \alpha) \subset E_0$  и субъекты  $x$  и  $s$  соединены  $tg$ -путем в графе  $G_0$ . Для случая  $s' \in S_0$  индуктивный шаг доказан (рис. 2.10).

Рассмотрим второй случай. Пусть  $s' \notin S_0$ . Заметим, что число преобразований графов  $N$  минимально, поэтому преобразования графов удовлетворяют следующим требованиям:

- каждый субъект графа  $G_0$  создает не более одного субъекта;
- субъект-создатель берет на созданный субъект необходимый набор прав  $\{t, g\}$ ;
- созданный субъект не создает новых субъектов.

Справедливость перечисленных требований следует из того, что если субъект графа  $G_0$  создает более одного субъекта или созданный субъект создает нового субъекта, то в преобразованиях графов доступа такие дополнительные субъекты могут быть заменены имеющимися. Следовательно, не будет являться необходимой операция создания дополнительных субъектов, что противоречит условию минимальности числа преобразований графов. Требова-

Рис. 2.10. Случай  $s' \in S_0$  и  $op_N = take(\alpha, x, s', y)$

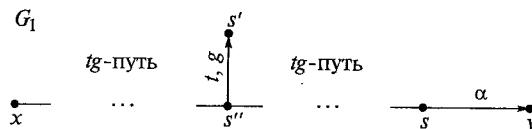


Рис. 2.11. Случай  $s' \notin S_0$  и  $op_N = \text{take}(\alpha, x, s', y)$

ние к субъекту-создателю брать на созданный субъект необходимый набор прав  $\{t, g\}$  является, очевидно, необходимым.

Из перечисленных требований следует, что существуют  $M < N - 1$ ,  $s'' \in S_0$ :  $op_M = \text{create}(\{t, g\}, s'', s')$ . Среди всех последовательностей преобразований длины  $N$  можно выбрать такую, что  $M = 1$ . Тогда рассмотрим граф  $G_1$ ; при этом  $S_1 = S_0 \cup \{s'\}$ ,  $E_1 = E_0 \cup \{(s'', s', \{t, g\})\}$ , истинен предикат «возможен доступ»( $\alpha, s', y, G_1$ ) с числом преобразований меньше  $N$  и  $s' \in S_1$ . Следовательно, существует  $s \in S_1$ :  $(s, y, \alpha) \subset E_1$ ; при этом  $s$  и  $s'$  соединены  $tg$ -путем в графе  $G_1$ . Очевидно, что  $s \in S_0$ ,  $(s, y, \alpha) \subset E_0$ ; при этом  $s$  и  $s''$  соединены  $tg$ -путем в графе  $G_0$ . Кроме того, если  $op_N = \text{take}(\alpha, x, s', G_1)$ , а если  $op_N = \text{grant}(\alpha, s', x, y)$ , то истинен предикат «возможен доступ»( $\{g\}, s', x, G_1$ ); при этом число преобразований меньше  $N$ . Следовательно, так как  $s''$  — единственный субъект в графе  $G_1$ , обладающий правами доступа на  $s'$ , то  $x$  и  $s''$  соединены  $tg$ -путем в графе  $G_1$  и, очевидно, в графе  $G_0$ .

Таким образом, существует  $s \in S_0$ :  $(s, y, \alpha) \subset E_0$ ; при этом  $x$  и  $s$  соединены  $tg$ -путем в графе  $G_0$ . Для случая  $s' \notin S_0$  индуктивный шаг доказан (рис. 2.11). Теорема доказана.

Для определения истинности предиката «возможен доступ» в произвольном графе необходимо определить ряд дополнительных понятий.

**Определение 2.15.** Островом в произвольном графе доступов  $G_0$  называется его максимальный  $tg$ -связный подграф, состоящий только из вершин субъектов.

**Определение 2.16.** Мостом в графе доступов  $G_0$  называется  $tg$ -путь, концами которого являются вершины-субъекты, проходящий через вершины-объекты, словарная запись которого имеет следующий вид:  $\bar{t}^*, \bar{t}^*, \bar{t}^*, \bar{t}^* \bar{g} \bar{t}^*, \bar{t}^* \bar{g} \bar{t}^*$ , где символ \* означает много-кратное (в том числе нулевое) повторение.

**Определение 2.17.** Начальным пролетом моста в графе доступов  $G_0$  называется  $tg$ -путь, началом которого является вершина-субъект, концом — объект, проходящий через вершины-объекты, словарная запись которого имеет следующий вид:  $\bar{t}^* \bar{g}$ .

**Определение 2.18.** Конечным пролетом моста в графе доступов  $G_0$  называется  $tg$ -путь, началом которого является вершина-субъект, концом — объект, проходящий через вершины-объекты, словарная запись которого имеет следующий вид:  $\bar{t}^*$ .

**Теорема 2.6.** Пусть  $G_0 = (S_0, O_0, E_0)$  — произвольный граф доступов,  $x, y \in O_0$ ,  $x \neq y$ . Предикат «возможен доступ»( $\alpha, x, y, G_0$ ) истинен тогда и только тогда, когда выполняются условия 3, 4 и 5.

**Условие 3.** Существуют объекты  $s_1, \dots, s_m \in O_0$ :

$(s_i, y, \gamma_i) \subset E_0$  для  $i = 1, \dots, m$  и  $\alpha = \gamma_1 \cup \dots \cup \gamma_m$ .

**Условие 4.** Существуют субъекты  $x'_1, \dots, x'_m, s'_1, \dots, s'_m \in S_0$ :

а)  $x = x'_i$  или  $x'_i$  соединен с  $x$  начальным пролетом моста в графе  $G_0$ , где  $i = 1, \dots, m$ ;

б)  $s = s'_i$  или  $s'_i$  соединен с  $s$  конечным пролетом моста в графе  $G_0$ , где  $i = 1, \dots, m$ .

**Условие 5.** В графе  $G_0$  для каждой пары  $(x'_i, s'_i)$ , где  $i = 1, \dots, m$ , существуют острова  $I_{i,1}, \dots, I_{i,u_i}$ ,  $u_i \geq 1$  такие, что  $x'_i \in I_{i,1}$ ,  $s'_i \in I_{i,u_i}$ , и существуют мосты между островами  $I_{ij}$  и  $I_{ij+1}$ , где  $j = 1, \dots, u_{i-1}$ .

**Доказательство.** Проведем доказательство теоремы для  $m = 1$ , так как схему доказательства для этого случая легко продолжить в случае  $m > 1$ .

При  $m = 1$  условия 3, 4, 5 формулируются следующим образом (рис. 2.12).

**Условие 3.** Существует  $s \in O_0$ :  $(s, y, \alpha) \subset E_0$ .

**Условие 4.** Существуют  $x', s' \in S_0$ :

а)  $x = x'$  или  $x'$  соединен с  $x$  начальным пролетом моста в графе  $G_0$ ;

б)  $s = s'$  или  $s'$  соединен с  $s$  конечным пролетом моста в графе  $G_0$ .

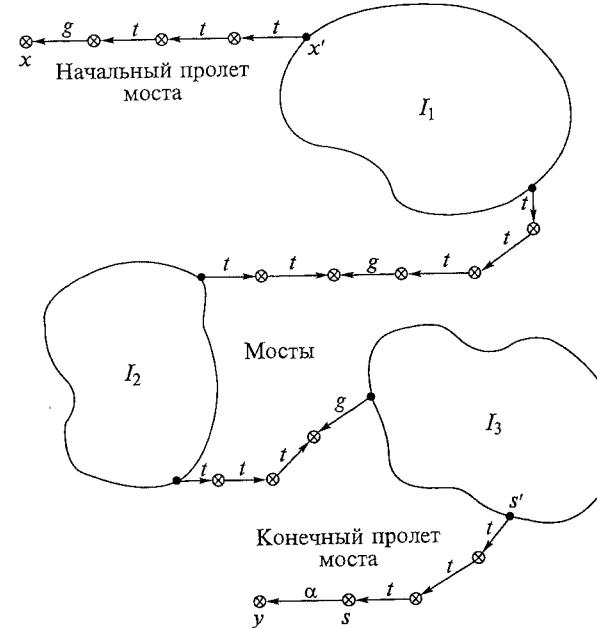


Рис. 2.12. Пример пути передачи объекту  $x$  прав доступа  $\alpha$  на объект  $y$

**Условие 5.** В графе  $G_0$  существуют острова  $I_1, \dots, I_u$ ,  $u \geq 1$ , такие что  $x' \in I_1$ ,  $s' \in I_u$ , и существуют мосты между островами  $I_j$  и  $I_{j+1}$ , где  $j = 1, \dots, u-1$ .

**Достаточность.** Является очевидным тот факт, что по мостам возможна передача прав доступа между субъектами, являющимися его концами. В качестве примера разобрана последовательность преобразований графа доступов при передаче прав по мосту вида  $t \overline{g} t$  (рис. 2.13).

Также очевидно, что по конечному пролету моста субъект может забрать права доступа у объекта, а по начальному пролету моста — передать его объекту.

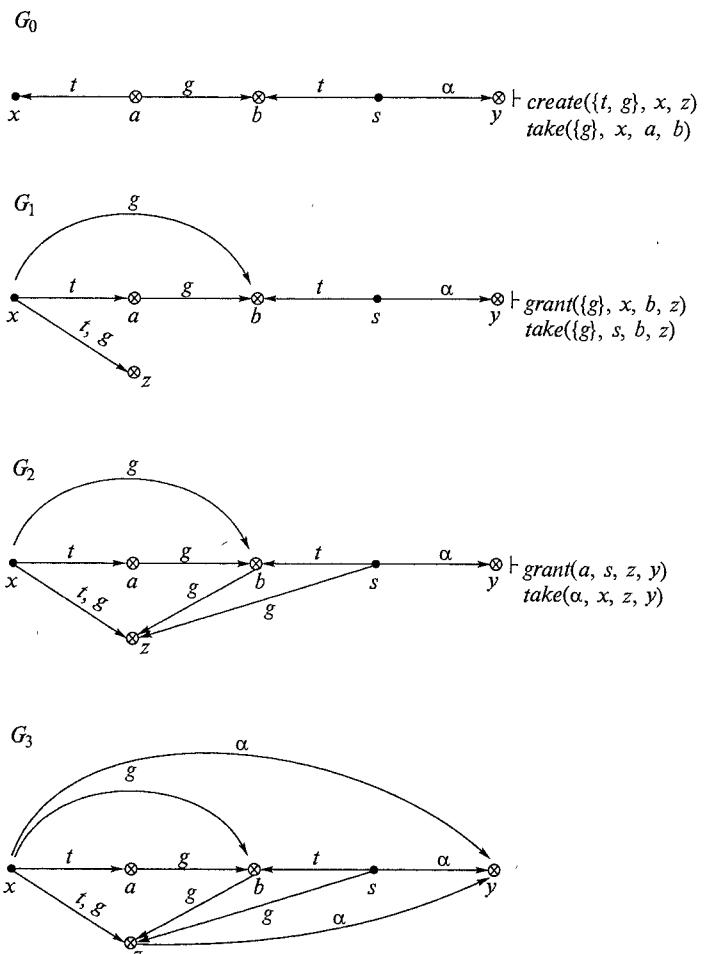


Рис. 2.13. Пример передачи прав доступа по мосту

По условию 3 существует объект  $s$ , который обладает правами  $\alpha$  на объект  $y$ . По условию 4,  $b$  существует субъект  $s'$ , который либо совпадает с  $s$ , либо по конечному пролету моста может забрать у субъекта  $s$  права  $\alpha$  на объект  $y$ .

По теореме 2.1 права доступа, полученные одним субъектом, принадлежащим острову, могут быть переданы любому другому субъекту острова. По условию 5 между островами существуют мосты, по которым возможна передача прав доступа. По условию 4,  $a$  существует субъект  $x'$ , который или совпадает с  $x$ , или, получив права доступа, может передать их  $x$  по начальному пролету моста.

**Необходимость.** Пусть истинен предикат «возможен доступ»( $\alpha$ ,  $x$ ,  $y$ ,  $G_0$ ). По определению истинности предиката существует последовательность графов доступов  $G_1 = (S_1, O_1, E_1), \dots, G_N = (S_N, O_N, E_N)$  такая, что  $G_0 \models_{op_1} G_1 \models_{op_2} \dots \models_{op_N} G_N$  и  $(x, y, \alpha) \subset E_N$ ; при этом  $N$  выбирается минимальным. Тогда  $(x, y, \alpha) \notin E_{N-1}$ . Докажем необходимость условий 3, 4, 5 индукцией по  $N$ .

При  $N=0$  очевидно, что  $(x, y, \alpha) \subset E_0$ . Следовательно, условия 3, 4, 5 выполнены.

При  $N=1$  из определения де-юре правил модели следует, что:

- либо существует субъект  $s \in S_0$  такой, что справедливо условие  $(s, y, \alpha) \in E_0$  и  $op_1 = grant(\alpha, s, x, y)$ ;
- либо существует объект  $s \in O_0$  такой, что справедливо условие  $(s, y, \alpha) \in E_0$  и  $op_1 = take(\alpha, x, s, y)$ .

Таким образом, для  $N=1$  условия теоремы выполняются.

Пусть  $N>0$  и утверждение теоремы истинно для  $k < N$ . Тогда  $(x, y, \alpha) \notin E_{N-1}$  и ребро  $(x, y, \alpha)$  появляется в графе доступов  $G_N$  в результате применения к графу  $G_{N-1}$  некоторого правила  $op_N$ . Возможны два случая:  $x \notin S_0$  и  $x \in S_0$ .

Если  $x \notin S_0$ , то существует  $x_1 \in S_{N-1}$ :  $op_N = grant(\alpha, x_1, x, y)$ . Также возможны два случая:  $x_1 \in S_0$  или  $x_1 \notin S_0$ .

Если  $x_1 \in S_0$ , то истинен предикат «возможен доступ»( $\{g\}$ ,  $x_1$ ,  $x$ ,  $G_0$ ) с числом преобразований графов, меньшим  $N$ . Следовательно, по предположению индукции существуют:

- $x_2 \in O_0$ :  $(x_2, x, g) \in E_0$ ;
- $x' \in E_0$ , соединенный с  $x_2$  конечным пролетом моста;
- острова  $I_1, \dots, I_t$ ,  $t \geq 1$ , такие что  $x_1 \in I_t$ ,  $x' \in I_1$ , и существуют мосты между островами  $I_j$  и  $I_{j+1}$  для  $j = 1, \dots, t-1$ .

Кроме того, истинен предикат «возможен доступ»( $\alpha$ ,  $x_1$ ,  $y$ ,  $G_0$ ) с числом преобразований графов, меньшим  $N$ . Тогда по предположению индукции существуют:

- $s \in O_0$ :  $(s, y, \alpha) \subset E_0$ ;
- $s' \in E_0$ :  $s = s'$  или  $s'$  соединен с  $s$  конечным пролетом моста;
- острова  $I_1, \dots, I_u$ ,  $t-u \geq 1$ , такие что  $x_1 \in I_t$ ,  $s' \in I_u$ , и существуют мосты между островами  $I_j$  и  $I_{j+1}$  для  $j = t, \dots, u-1$ .

Заметим, что путь, соединяющий вершины  $x'$ ,  $x_2$ ,  $x$ , есть начальный пролет моста.

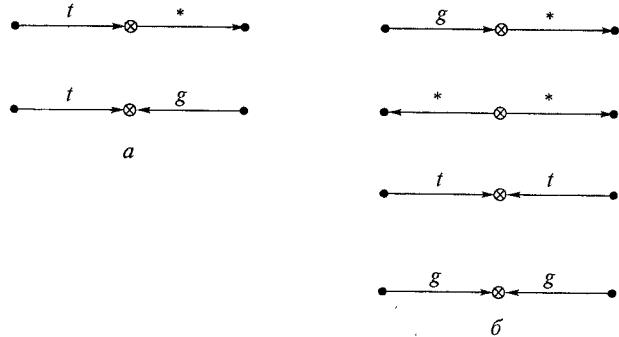


Рис. 2.14. Виды путей длины 2 (\* — или  $t$ , или  $g$ ):

*a, b* — пути, по которым соответственно возможна и невозможна передача прав доступа

Если  $x_1 \notin S_0$ , то с учетом замечаний, сделанных при доказательстве теоремы 2.5, получаем, что существуют  $M < N - 1$ ,  $x_2 \in S_0$  такие, что  $opr_M = create(\{t, g\}, x_2, x_1)$ . Среди всех последовательностей преобразований длины  $N$  можно выбрать такую, что  $M = 1$ . Далее используем технику доказательства теоремы 2.1 и доказательства индуктивного шага для случая  $x_1 \in S_0$ . Таким образом, для случая  $x \notin S_0$  условия 3, 4, 5 выполняются и индуктивный шаг доказан.

Если  $x \in S_0$ , то п. 1 условия 4, а выполняется. Многократно применяя технику доказательства, использованную выше и в теореме 2.1, доказываем индуктивный шаг и в данном случае. Теорема доказана.

**Замечание.** При доказательстве теоремы 2.2 можно показать, что не существует путей, отличных от мостов между двумя субъектами, проходящих через вершины-объекты, по которым возможна передача прав доступа.

**Доказательство.** Рассмотрим все пути (с учетом симметрии) длины 2 между двумя субъектами, проходящие через вершины-объекты, по которым возможна передача прав доступа (рис. 2.14, *a*) и невозможна передача прав доступа (рис. 2.14, *b*).

Очевидно, что любой путь, соединяющий двух субъектов и проходящий через объекты, по которому возможна передача прав доступа, не должен содержать фрагменты, приведенные на рис. 2.14, *b*. В то же время очевидно, что любой такой путь, состоящий только из фрагментов, приведенных на рис. 2.14, *a*, является мостом. Утверждение замечания доказано.

### Похищение прав доступа

Способ санкционированной передачи прав доступа предполагает идеальное сотрудничество субъектов. В случае похищения прав

доступа передача прав доступа на объект осуществляется без содействия субъекта, изначально обладавшего передаваемыми правами.

**Определение 2.19.** Пусть  $x, y \in O_0$ ,  $x \neq y$  — различные объекты графа доступов  $G_0 = (S_0, O_0, E_0)$ ,  $\alpha \subseteq R$ . Определим предикат «возможно похищение»( $\alpha, x, y, G_0$ ), который будет истинным тогда и только тогда, когда  $(x, y, \alpha) \cap E_0 = \emptyset$  и существуют графы  $G_1 = (S_1, O_1, E_1)$ , ...,  $G_N = (S_N, O_N, E_N)$  такие, что  $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \dots \vdash_{op_N} G_N$  и  $(x, y, \alpha) \subset E_N$ ; при этом, если существует  $(s, y, \gamma) \subset E_0$ , где  $\gamma \subseteq \alpha$ , то справедливо неравенство  $opr_K \neq grant(\gamma, s, z, y)$ , где  $z \in O_{K-1}$ , для  $K = 1, \dots, N$ .

**Теорема 2.7.** Пусть  $G_0 = (S_0, O_0, E_0)$  — произвольный граф доступов,  $x, y \in O_0$ ,  $x \neq y$ . Предикат «возможно похищение»( $\alpha, x, y, G_0$ ) истинен тогда и только тогда, когда выполняются условия 6, 7, 8, 9.

**Условие 6.**  $(x, y, \alpha) \cap E_0 = \emptyset$ .

**Условие 7.** Существуют объекты  $s_1, \dots, s_m \in O_0$ :

$(s_i, y, \gamma_i) \subset E_0$  для  $i = 1, \dots, m$  и  $\alpha = \gamma_1 \cup \dots \cup \gamma_m$ .

**Условие 8.** Являются истинными предикаты «возможен доступ»( $\{t\}, x, s_i, G_0$ ), где  $i = 1, \dots, m$ .

**Условие 9.** Граф доступов  $G_0$  не является графиком указанного вида (рис. 2.15).

**Доказательство.** Доказательство осуществляется аналогично доказательству теоремы 2.6.

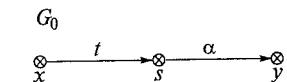


Рис. 2.15. Условие 9 теоремы 2.7

## 2.2.2. Расширенная модель Take-Grant

### Направления развития модели Take-Grant

Рассмотренные в классической модели *Take-Grant* способы анализа путей распространения прав доступа в системах дискретционного разграничения доступа имеют в большей степени теоретическое значение, так как, как правило, в реальных системах не реализуются столь сложные графы доступов, для анализа которых необходимо использовать теоремы 2.5 — 2.7. В то же время на основе классической модели были разработаны ее расширения [12], которые развивают идеи классической модели, предлагая новые механизмы анализа, в большей степени применимые к современным системам защиты информации.

Рассмотрим три расширения модели.

1. Правила де-факто, предназначенные для поиска и анализа информационных потоков.

2. Алгоритм построения замыкания графа доступов и информационных потоков.

3. Способы анализа путей распространения прав доступа и информационных потоков.

### Де-факто правила расширенной модели Take-Grant

Вместо прав доступа *take* и *grant* в расширенной модели в первую очередь рассматриваются права доступа *read* и *write*, наличие которых у субъектов системы является причиной возникновения информационных потоков.

Расширенная модель *Take-Grant* строится на основе классической модели. Ее основными элементами являются:

$O$  — множество объектов;

$S \subseteq O$  — множество субъектов;

$R = \{r_1, r_2, \dots, r_m\} \cup \{t, g\} \cup \{r, w\}$  — множество видов прав доступа и видов информационных потоков, где  $r$  (*read*) — право на чтение или информационный поток на чтение;  $w$  (*write*) — право на запись или информационный поток на запись;

$G = (S, O, E \cup F)$  — конечный помеченный ориентированный граф без петель, представляющий текущие доступы в системе и информационные потоки. Элементы множеств  $S, O$  являются вершинами графа. Элементы множества  $E \subseteq O \times O \times R$  являются «реальными» ребрами графа, соответствующими правам доступа, и в графе доступов обозначаются сплошными линиями. Элементы множества  $F \subseteq O \times O \times \{r, w\}$  являются «мнимыми» ребрами, соответствующими информационным потокам, и в графе доступов обозначаются пунктирными линиями. Каждое «реальное» ребро помечено непустым подмножеством множества видов прав доступа  $R$ , каждое «мнимое» ребро помечено непустым подмножеством множества  $\{r, w\}$ .

Состояние системы описывается соответствующим ему графом доступов и информационных потоков.

Порядок перехода системы расширенной модели *Take-Grant* из состояния в состояние определяется де-юре и де-факто правилами преобразования графа доступов и информационных потоков. Преобразование графа  $G$  в граф  $G'$  в результате выполнения правила *op* обозначим следующим образом:

$$G \vdash_{op} G'.$$

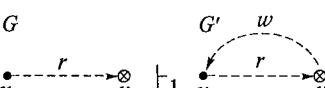


Рис. 2.16. Применение первого де-факто правила

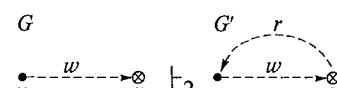


Рис. 2.17. Применение второго де-факто правила

Определения де-юре правил *take()*, *grant()*, *create()*, *remove()* совпадают с определениями этих правил в классической модели *Take-Grant*. Де-юре правила применяются только к «реальным» ребрам (элементам множества  $E$ ).

Де-факто правила применяются к «реальным» или «мнимым» ребрам (элементам множества  $E \cup F$ ), помеченным  $r$  или  $w$ . Результатом применения де-факто правил является добавление новых «мнимых» ребер в множество  $F$ . Рассматриваются шесть де-факто правил: два вспомогательных и четыре основных.

Рассмотрим порядок применения де-юре правил преобразования графа доступов. В отличие от де-юре правил для применения трех из шести де-факто правил требуется участие двух субъектов (рис. 2.16—2.21).

Рассмотрим условия применения де-факто правил в исходном состоянии  $G = (S, O, E)$  и результаты их применения в результирующем состоянии  $G' = (S', O', E')$  (табл. 2.4).

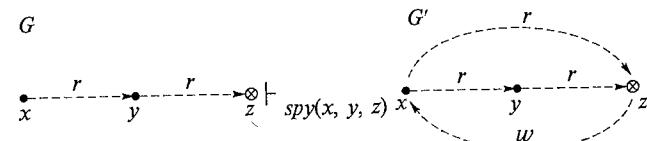


Рис. 2.18. Применение правила *spy(x, y, z)*

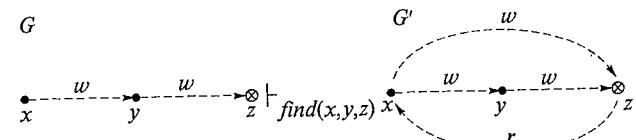


Рис. 2.19. Применение правила *find(x, y, z)*

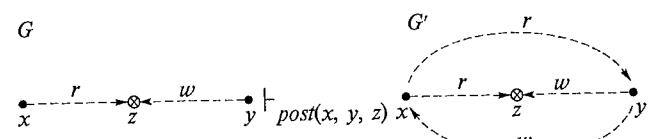


Рис. 2.20. Применение правила *post(x, y, z)*

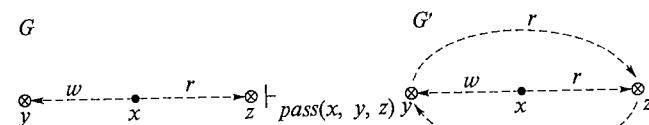


Рис. 2.21. Применение правила *pass(x, y, z)*

Таблица 2.4. Де-факто правила расширенной модели *Take-Grant*

Правила	Исходное состояние $G = (S, O, E \cup F)$	Результирующее состояние $G' = (S, O, E \cup F')$
Первое правило	$x \in S$ , $(x, y, r) \in E \cup F$	$F' = F \cup \{(y, x, w), (x, y, r)\}$
Второе правило	$x \in S$ , $(x, y, w) \in E \cup F$	$F' = F \cup \{(y, x, r), (x, y, w)\}$
$spy(x, y, z)$	$x, y \in S, x \neq z$ , $\{(x, y, r), (y, z, r)\} \subset E \cup F$	$F' = F \cup \{(x, z, r), (z, x, w)\}$
$find(x, y, z)$	$x, y \in S, x \neq z$ , $\{(x, y, w), (y, z, w)\} \subset E \cup F$	$F' = F \cup \{(x, z, w), (z, x, r)\}$
$post(x, y, z)$	$x, y \in S, x \neq y$ , $\{(x, z, r), (y, z, w)\} \subset E \cup F$	$F' = F \cup \{(x, y, r), (y, x, w)\}$
$find(x, y, z)$	$x \in S, y \neq z$ , $\{(x, y, w), (x, z, r)\} \subset E \cup F$	$F' = F \cup \{(y, z, r), (z, y, w)\}$

Из определения де-факто правил следует, что для анализа информационных потоков достаточно рассматривать потоки одного вида: либо на чтение, либо на запись. В дальнейшем будем рассматривать только информационные потоки на запись. Будем также предполагать, что при возникновении информационного потока не накладывается ограничений на кооперацию субъектов системы, участвующих в этом процессе.

**Определение 2.20.** Пусть  $x, y \in O_0$ ,  $x \neq y$  — различные объекты графа доступов и информационных потоков  $G_0 = (S_0, O_0, E_0 \cup F_0)$ . Определим предикат «возможна запись»( $x, y, G_0$ ), который будет истинным тогда и только тогда, когда существуют графы  $G_1 = (S_1, O_1, E_1)$ , ...,  $G_N = (S_N, O_N, E_N)$  и де-юре или де-факто правила  $op_1$ , ...,  $op_N$  такие, что  $G_0 \vdash_{op_1} G_1 \vdash_{op_2} \dots \vdash_{op_N} G_N$  и  $(x, y, w) \in F_N$ .

Для проверки истинности предиката «возможна запись»() также следует определить необходимые и достаточные условия, задача проверки которых алгоритмически разрешима.

**Теорема 2.8.** Пусть  $G_0 = (S_0, O_0, E_0 \cup F_0)$  — граф доступов и информационных потоков,  $x, y \in O_0$ ,  $x \neq y$ . Тогда предикат «возможна запись»( $x, y, G_0$ ) истинен только тогда, когда существуют объекты  $o_1, \dots, o_m \in O_0$ , где  $o_1 = x$ ,  $o_m = y$ , такие что или  $m = 2$  и  $(x, y, w) \in F_0$ , или для  $i = 1, \dots, m - 1$  выполняется одно из условий:

- $o_i \in S_0$  и или истинен предикат «возможен доступ»( $\{w\}, o_i, o_{i+1}, G_0$ ), или  $(o_i, o_{i+1}, w) \in E_0 \cup F_0$ ;
- $o_{i+1} \in S_0$  и или истинен предикат «возможен доступ»( $\{r\}, o_{i+1}, o_i, G_0$ ), или  $(o_{i+1}, o_i, r) \in E_0 \cup F_0$ ;
- $o_i, o_{i+1}$  и или истинен предикат «возможен доступ»( $\alpha, o_i, o_{i+1}, G_0$ ), или истинен предикат «возможен доступ»( $\alpha, o_{i+1}, o_i, G_0$ ), где  $\alpha \in \{t, g\}$ .

**Доказательство.** Доказательство осуществляется аналогично доказательству теорем 2.5 и 2.6.

### Построение замыкания графа доступов и информационных потоков

Для проверки истинности предикатов «возможен доступ»(), «возможно похищение»() или «возможна запись»() для многих пар вершин неэффективно использовать алгоритмы проверки условий теорем 2.5 — 2.8. Эффективнее использовать алгоритмы, позволяющие осуществлять проверку истинности указанных предикатов сразу для всех пар вершин. Такие алгоритмы реализуют преобразование графа доступов и информационных потоков в его замыкание.

**Определение 2.21.** Пусть  $G = (S, O, E \cup F)$  — граф доступов и информационных потоков такой, что для каждого  $s \in S$  существует  $o \in O$  и при этом  $(s, o, \{t, g\}) \subset E$ . Тогда замыканием (или де-факто-замыканием) графа  $G$  называется граф доступов и информационных потоков  $G^* = (S, O, E^* \cup F^*)$ , полученный из  $G$  применением последовательности правил *take()*, *grant()* и де-факто правил. При этом применение к графу  $G^*$  указанных правил не приводит к появлению в нем новых ребер.

Алгоритм построения замыкания графа доступов состоит из трех этапов.

1. Построение *tg*-замыкания.
2. Построение де-юре-замыкания.
3. Построение замыкания.

**Определение 2.22.** Пусть  $G = (S, O, E \cup F)$  — граф доступов и информационных потоков такой, что для каждого  $s \in S$  существует  $o \in O$  и при этом  $(s, o, \{t, g\}) \subset E$ . Тогда *tg*-замыканием графа  $G$  называется граф доступов и информационных потоков  $G^{tg} = (S, O, E^{tg} \cup F)$ , полученный из  $G$  применением последовательности правил *take()* или *grant()*. При этом каждое ребро  $(s, o, \alpha) \in E^{tg} \setminus E$  имеет вид  $(s, o, t)$  или  $(s, o, g)$  и применение к графу  $G^{tg}$  правил *take()* или *grant()* не приводит к появлению в нем новых ребер указанного вида.

**Определение 2.23.** Пусть  $G = (S, O, E \cup F)$  — граф доступов и информационных потоков такой, что для каждого  $s \in S$  существует

ет  $o \in O$  и при этом  $(s, o, \{t, g\}) \subset E$ . Тогда де-юре-замыканием графа  $G$  называется граф доступов и информационных потоков  $G^{\text{де-юре}} = (S, O, E^{\text{де-юре}} \cup F)$ , полученный из  $G$  применением последовательности правил *take()* или *grant()*. При этом применение к графу  $G^{\text{де-юре}}$  правил *take()* или *grant()* не приводит к появлению в нем новых ребер.

**Алгоритм 2.2.** Алгоритм построения *tg*-замыкания графа доступов и информационных потоков  $G = (S, O, E \cup F)$  состоит из пяти шагов.

*Шаг 1.* Для каждого  $s \in S$  выполнить правило *create*( $\{t, g\}, s, o$ ); при этом создаваемые объекты занести в множество  $O$ .

*Шаг 2.* Инициализировать:  $L = \{(x, y, \alpha) \in E : \alpha \in \{t, g\}\}$  — список ребер графа доступов и информационных потоков — и  $N = \emptyset$  — множество вершин.

*Шаг 3.* Выбрать из списка  $L$  первое ребро  $(x, y, \alpha)$ . Занести  $x, y$  в множество  $N$ . Удалить ребро  $(x, y, \alpha)$  из списка  $L$ .

*Шаг 4.* Для всех вершин  $z \in N$  проверить возможность применения правил *take()* или *grant()* на тройке вершин  $x, y, z$  с использованием ребра  $(x, y, \alpha)$ . Если в результате применения правил *take()* или *grant()* появляются новые ребра вида  $(a, b, \beta)$ , где  $\{a, b\} \subset \{x, y, z\}$  и  $\beta \in \{t, g\}$ , то занести их в конец списка  $L$  и множество  $E$ .

*Шаг 5.* Пока список  $L$  не пуст, перейти на шаг 3.

Очевидно, что вычислительная сложность данного алгоритма пропорциональна  $|O|^3$ .

**Теорема 2.9.** Алгоритм 2.2 корректно строит *tg*-замыкание графа доступов и информационных потоков.

**Доказательство.** Пусть  $G' = (S, O, E' \cup F)$  — график доступов и информационных потоков, полученный из  $G = (S, O, E \cup F)$  после применения алгоритма 2.2. Пусть  $G^{\text{tg}} = (S, O, E^{\text{tg}} \cup F)$  — *tg*-замыкание графа  $G$ . Необходимо доказать, что  $G' = G^{\text{tg}}$ .

Докажем от противного. Пусть существует ребро  $(x, y, \alpha) \in E^{\text{tg}} \setminus E'$ , где  $\alpha \in \{t, g\}$ . Тогда существуют два ребра  $(a, b, \beta), (c, d, \gamma) \in E^{\text{tg}}$ , где  $\beta, \gamma \in \{t, g\}$ , использованные в правилах *take()* или *grant()*, применение которых привело к появлению ребра  $(x, y, \alpha)$ .

Если  $(a, b, \beta), (c, d, \gamma) \in E'$ , то, согласно описанию алгоритма 2.2, ребро  $(x, y, \alpha) \in E'$  — противоречие. Следовательно, или  $(a, b, \beta) \in E^{\text{tg}} \setminus E'$ , или  $(c, d, \gamma) \in E^{\text{tg}} \setminus E'$ . В то же время график  $G^{\text{tg}}$  получен из  $G$  в результате применения конечной последовательности правил *take()* и *grant()*. Ребра  $(a, b, \beta), (c, d, \gamma)$  должны быть получены раньше, чем ребро  $(x, y, \alpha)$ . Таким образом, повторяя рассуждения для ребер, с использованием которых получены ребра  $(a, b, \beta), (c, d, \gamma)$ , придем к противоречию, так как все ребра графа  $G$  изначально содержатся в графике  $G'$ . Следовательно,  $E^{\text{tg}} = E'$  и  $G' = G^{\text{tg}}$ . Теорема доказана.

**Алгоритм 2.3.** Алгоритм построения де-юре-замыкания графа доступов и информационных потоков  $G = (S, O, E \cup F)$  состоит из четырех шагов.

*Шаг 1.* Выполнить алгоритм 2.2.

*Шаг 2.* Для каждой пары ребер вида  $(x, y, t), (y, z, \alpha) \in E^{\text{tg}}$ , где  $x \in S$ , применить правило *take*( $\alpha, x, y, z$ ) и, если полученное ребро  $(x, z, \alpha) \notin E^{\text{tg}}$ , занести его в множество  $E^{\text{tg}}$ .

*Шаг 3.* Для каждой пары ребер вида  $(x, y, g), (x, z, \alpha) \in E^{\text{tg}}$ , где  $x \in S$ , применить правило *grant*( $\alpha, x, y, z$ ) и, если полученное ребро  $(y, z, \alpha) \notin E^{\text{tg}}$ , занести его в множество  $E^{\text{tg}}$ .

*Шаг 4.* Для каждой пары ребер вида  $(x, y, t), (y, z, \alpha) \in E^{\text{tg}}$ , где  $x \in S$ , применить правило *take*( $\alpha, x, y, z$ ) и, если полученное ребро  $(x, z, \alpha) \notin E^{\text{tg}}$ , занести его в множество  $E^{\text{tg}}$ .

**Теорема 2.10.** Алгоритм 2.3 корректно строит де-юре-замыкание графа доступов и информационных потоков.

**Доказательство.** После построения *tg*-замыкания на первом шаге алгоритма 2.3 всем субъектам графа доступов необходимо выполнить следующую последовательность действий:

- используя правило *grant()*, раздать права доступа и, используя правило *take()*, забрать права доступа (рис. 2.22, а);

- используя правило *take()*, забрать права доступа и, используя правило *grant()*, раздать права доступа (рис. 2.22, б).

Объединяя указанные последовательности, получаем шаги 2 — 4 алгоритма. Теорема доказана.

**Алгоритм 2.4.** Алгоритм построения де-факто-замыкания графа доступов и информационных потоков  $G = (S, O, E \cup F)$  состоит из шести шагов.

*Шаг 1.* Выполнить алгоритм 2.3.

*Шаг 2.* Для всех ребер  $(x, y, \alpha) \in E^{\text{де-юре}} \cup F$ , где  $x \in S, \alpha \in \{w, r\}$ , применить первые два де-факто правила. Если будут получены новые ребра, то занести их в множество  $F$ .

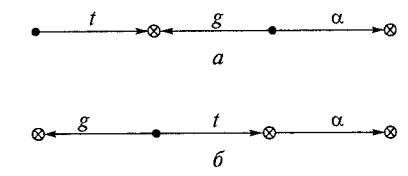
*Шаг 3.* Инициализировать:  $L = \{(x, y, \alpha) \in E^{\text{де-юре}} \cup F : \alpha \in \{w, r\}\}$  — список ребер графа доступов и информационных потоков — и  $N = \emptyset$  — множество вершин.

*Шаг 4.* Выбрать из списка  $L$  первое ребро  $(x, y, \alpha)$ . Занести  $x, y$  в множество  $N$ . Удалить ребро  $(x, y, \alpha)$  из списка  $L$ .

*Шаг 5.* Для всех вершин  $z \in N$  проверить возможность применения де-факто правил на тройке вершин  $x, y, z$  с использованием ребра  $(x, y, \alpha)$ . Если в результате применения де-факто правил

Рис. 2.22. Примеры для последовательностей шагов алгоритма 2.3:

а — график доступов для последовательности действий *grant()*, *take()*; б — график доступов для последовательности действий *take()*, *grant()*



*spy()*, *find()*, *post()*, *pass()* появятся новые ребра вида  $(a, b, \beta)$ , где  $\{a, b\} \subset \{x, y, z\}$  и  $\beta \in \{r, w\}$ , то занести их в конец списка  $L$  и множество  $F$ .

Шаг 6. Пока список  $L$  не пуст, перейти на шаг 4.

**Теорема 2.11.** Алгоритм 2.4 корректно строит де-факто-замыкание графа доступов и информационных потоков.

**Доказательство.** Алгоритм 2.4 аналогичен алгоритму 2.2. Следовательно, доказательство теоремы 2.11 аналогично доказательству теоремы 2.9.

### Анализ путей распространения прав доступа и информационных потоков

Можно рассмотреть проблему поиска и анализа информационных потоков в ином свете. Допустим, факт нежелательной передачи прав или информации уже состоялся. Каков наиболее вероятный путь его осуществления? В классической модели *Take-Grant* не дается прямого ответа на этот вопрос. Можно говорить, что есть возможность передачи прав доступа или возникновения информационного потока, но нельзя определить, какой путь при этом использовался.

Рассмотрим подходы к решению задачи определения возможных путей передачи прав доступа или возникновения информационных потоков.

Предположим, что чем больше узлов на пути между вершинами, по которому произошла передача прав доступа или возник информационный поток, тем меньше вероятность использования этого пути.

Рассмотрим пример, представленный на рис. 2.23. Интуитивно ясно, что наиболее вероятный путь передачи информации от субъекта  $z$  к субъекту  $x$  лежит через объект  $y$ . В то же время нарушитель для большей скрытности может специально использовать более длинный путь через  $a, b, c$ .

Особенно, если предположить, что информация в объекте  $y$  контролируется администратором безопасности системы.

Рассмотрим другой пример. Какой из двух путей возникновения информационного потока, представленных на рис. 2.24, более вероятный?

Путь, представленный на рис. 24,  $b$ , реализуется за счет активных действий субъекта  $x$ , заин-

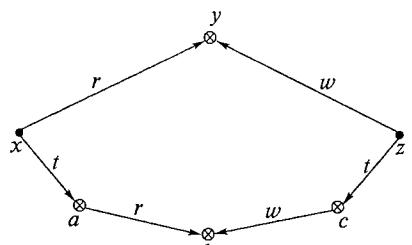


Рис. 2.23. Пути возникновения информационного потока на запись от  $z$  к  $x$

тересованного в возникновении информационного потока. По этой причине наиболее вероятным, как правило, будет именно этот путь.

Таким образом, в расширенную модель *Take-Grant* можно включить понятие вероятности или стоимости пути передачи прав доступа или информации. Путем меньшей стоимости соответствует наибольшая вероятность, и их надо исследовать в первую очередь. Есть два основных подхода к определению стоимости путей.

Первый подход основан на присваивании стоимости ребрам графа доступов, находящимся на пути передачи прав доступа или возникновения информационного потока. В этом случае стоимость ребра определяется в зависимости от прав доступа, которыми оно помечено, а стоимость пути есть сумма стоимостей проходимых ребер.

Второй подход основан на присваивании стоимости каждому используемому де-юре или де-факто правилу. Стоимость правила при этом может быть выбрана исходя из условий функционирования системы *Take-Grant* и может:

- являться константой;
- зависеть от специфики правила;
- зависеть от числа и состава участников при применении правила;
- зависеть от степени требуемого взаимодействия субъектов.

Стоимость пути в этом случае определяется как сумма стоимостей примененных правил.

### 2.2.3. Представление систем *Take-Grant* системами ХРУ

Решение задачи представления систем классической модели *Take-Grant* системами ХРУ позволяет лучше изучить основные свойства двух моделей систем дискреционногограничения доступа.

Построим гомоморфизм системы *Take-Grant* и системы ХРУ.

Пусть состояние системы *Take-Grant* описывается графом  $G = (S_{tg}, O_{tg}, E)$ , а  $R_{tg}$  — множество прав доступа системы *Take-Grant*.

Примем для системы ХРУ:

$R = R_{tg} \cup \{own\}$  — множество прав доступа;

$S = O = O_{tg}$  — множество субъектов и объектов системы ХРУ;

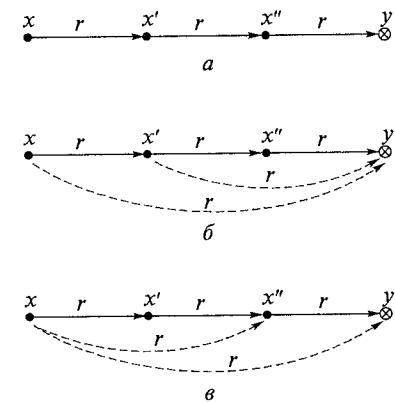


Рис. 2.24. Исходный граф (а) и пути (б, в) возникновения информационного потока на чтение от  $x$  к  $y$

$M_{|S| \times |S|}$  — матрица доступов, где для  $x, y \in O_{tg}$ , если  $(x, y, r) \in E$ , то  $r \in M[x, y]$ , и для всех  $s \in S_{tg}$  выполняется условие  $own \in M[s, s]$ ;  $q = (S, O, M)$  — состояние системы.

Переход системы ХРУ в соответствии с правилами модели *Take-Grant* осуществляется в результате применения команд пяти видов для каждого  $\alpha = \{r_1, \dots, r_k\} \subset R_{tg}$ .

1. command *take\_* $\alpha(x, y, z)$ :

```
if (own ∈ M[x, x]) and (t ∈ M[x, y]) and (r1 ∈ M[y, z]) and ...
and (rk ∈ M[y, z]) then
    «внести» право r1 в M[x, z];
    ...
    «внести» право rk в M[x, z];
endif
end.
```

2. command *grant\_* $\alpha(x, y, z)$ :

```
if (own ∈ M[x, x]) and (g ∈ M[x, y]) and (r1 ∈ M[x, z]) and ...
and (rk ∈ M[x, z]) then
    «внести» право r1 в M[y, z];
    ...
    «внести» право rk в M[y, z];
endif
end.
```

3. command *create\_object\_* $\alpha(x, y)$ :

```
if (own ∈ M[x, x]) then
    «создать» субъект y;
    «внести» право r1 в M[x, y];
    ...
    «внести» право rk в M[x, y];
endif
end.
```

4. command *create\_subject\_* $\alpha(x, y)$ :

```
if (own ∈ M[x, x]) then
    «создать» субъект y;
    «внести» право own в M[y, y];
    «внести» право r1 в M[x, y];
    ...
    «внести» право rk в M[x, y];
endif
end.
```

5. command *remove\_* $\alpha(x, y)$ :

```
if (own ∈ M[x, x]) then
    «удалить» право r1 из M[x, y];
    ...
    «удалить» право rk из M[x, y];
endif
end.
```

## Контрольные вопросы

1. Для команды  $C(q_{i_0}, a_{i_0}) = (q_i, a_i, l)$  машины Тьюринга выпишите две представляющие ее команды модели ХРУ.

2. Как с использованием команд и элементов описания системы ХРУ можно реализовать команду, содержащую в условии проверку на отсутствие в ячейке матрицы доступов некоторого права доступа?

3. Постройте граф создания для систем МТМД со следующими наборами команд:

a) command *a1*( $x:\alpha, y:\beta, z:\beta$ )  
«создать» субъект x с типом  $\alpha$ ;

command *a2*( $x:\alpha, y:\gamma, z:\beta, s:\delta$ )  
«создать» объект y с типом  $\gamma$ ;  
«создать» субъект s с типом  $\delta$ ;

command *a3*( $x:\varepsilon, y:\delta, z:\beta, s:\gamma, o:\delta$ )  
«создать» субъект o с типом  $\delta$ ;  
«создать» объект x с типом  $\varepsilon$ ;

end;

command *a1*( $x:\alpha, y:\beta, z:\gamma$ )  
«создать» субъект y с типом  $\beta$ ;

«создать» субъект x с типом  $\alpha$ ;

end;

command *a2*( $x:\beta, y:\delta, z:\delta$ )  
«создать» субъект z с типом  $\delta$ ;

end;

command *a3*( $x:\varepsilon, y:\alpha, z:\delta$ )  
«создать» объект x с типом  $\varepsilon$ ;

end.

Являются ли данные системы ациклическими?

4. Покажите, что для общего случая систем ХРУ не существует алгоритма проверки возможности утечки права доступа  $r$  для заданной пары: субъект  $s$  и объект  $o$ .

5. Как произвольную систему ТМД представить системой ХРУ?

6. Как по аналогии с определением безопасного начального состояния в модели ХРУ и с использованием определения предиката «возможен доступ»() определить безопасное начальное состояние в модели *Take-Grant*?

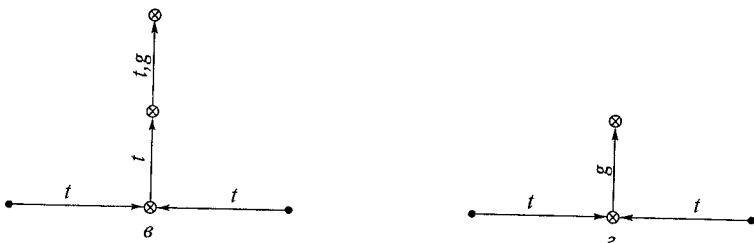
7. Как представить систему, построенную на основе модели *Take-Grant*, системой ТМД? При каких условиях система *Take-Grant* может быть представлена системой АМТМД?

8. Рассмотрите подробнее доказательство замечания в подразд. 2.2.1.

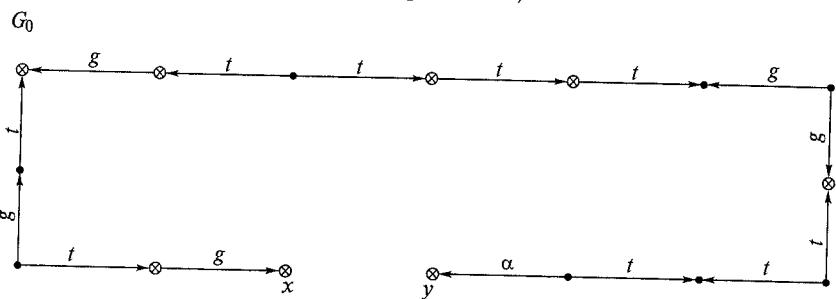
9. Являются ли мостами следующие графы доступов?



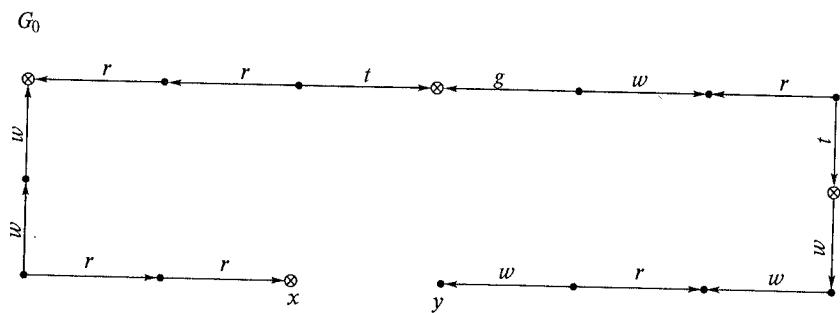
# МОДЕЛИ СИСТЕМ МАНДАТНОГО РАЗГРАНИЧЕНИЯ ДОСТУПА



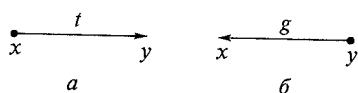
10. Истирен ли предикат «возможен доступ»( $\alpha, x, y, G_0$ ) для следующего графа доступов  $G_0$ ? (Решение задачи должно быть получено путем проверки выполнения условий теоремы 2.6.)



11. Истирен ли предикат «возможна запись»( $x, y, G_0$ ) для следующего графа доступов  $G_0$ ? (Решение задачи должно быть получено двумя способами: путем проверки выполнения условий теоремы 2.8 и путем непосредственного применения де-юре и де-факто правил.)



12. Как реализовать информационный поток на чтение от  $x$  к  $y$  и от  $y$  к  $x$  для систем со следующими графиками доступов?



## 3.1. МОДЕЛЬ БЕЛЛА—ЛАПАДУЛА

### 3.1.1. Классическая модель Белла—Лападула

Классическая модель Белла—Лападула была описана в 1975 г. [9] и в настоящее время является основной моделью, предназначенной для анализа систем защиты, реализующих мандатное разграничение доступа.

В классической модели Белла—Лападула [6] анализируются условия, при выполнении которых в компьютерной системе невозможно возникновение информационных потоков от объектов с большим уровнем конфиденциальности к объектам с меньшим уровнем конфиденциальности. Основными элементами классической модели Белла—Лападула являются:

$S$  — множество субъектов;

$O$  — множество объектов;

$R = \{read, write, append\}$  (доступ на запись в конец объекта),  $execute\}$  — множество видов доступа и видов прав доступа;

$B = \{b \subseteq S \times O \times R\}$  — множество возможных множеств текущих доступов в системе;

$(L, \leq)$  — решетка уровней конфиденциальности, например:  $L = \{U (unclassified), C (confidential), S (secret), TS (top secret)\}$ , где  $U < C < S < TS$ ;

$M_{|S| \times |O|}$  — матрица доступов, где  $M[s, o] \subseteq R$  — права доступа субъекта  $s$  на объект  $o$ ;

$(f_s, f_o, f_c) \in F = L^s \times L^o \times L^c$  — тройка функций  $(f_s, f_o, f_c)$ , определяющих:  $f_s: S \rightarrow L$  — уровень доступа субъекта;  $f_o: O \rightarrow L$  — уровень конфиденциальности объекта;  $f_c: S \rightarrow L$  — текущий уровень доступа субъекта, при этом для любого  $s \in S$  справедливо неравенство  $f_c(s) \leq f_s(s)$ ;

$V = B \times M \times F$  — множество состояний системы;

$Q$  — множество запросов системе;

$D$  — множество ответов по запросам, например:  $D = \{yes, no, error\}$ ;

$W \subseteq Q \times D \times V \times V$  — множество действий системы, где четверка  $(q, d, v^*, v) \in W$  означает, что система по запросу  $q$  с ответом  $d$  перешла из состояния  $v$  в состояние  $v^*$ ;

$N_0 = \{0, 1, 2, \dots\}$  — множество значений времени;

$X$  — множество функций  $x: N_0 \rightarrow Q$ , задающих все возможные последовательности запросов к системе;

$Y$  — множество функций  $y: N_0 \rightarrow D$ , задающих все возможные последовательности ответов системы по запросам;

$Z$  — множество функций  $z: N_0 \rightarrow V$ , задающих все возможные последовательности состояний системы.

**Определение 3.1.**  $\Sigma(Q, D, W, z_0) \subseteq X \times Y \times Z$  называется системой, если для каждого  $(x, y, z) \in \Sigma(Q, D, W, z_0)$  выполняется условие: для  $t \in N_0$ ,  $(x_t, y_t, z_{t+1}, z_t) \in W$ , где  $z_0$  — начальное состояние системы. При этом каждый набор  $(x, y, z) \in \Sigma(Q, D, W, z_0)$  называется реализацией системы, а  $(x_t, y_t, z_{t+1}, z_t) \in W$  называется действием системы в момент времени  $t \in N_0$ .

**Замечание.** В классической модели Белла—ЛаПадула рассматриваются следующие запросы, входящие в множество  $Q$ :

1. Запросы изменения множества текущих доступов  $b$ .
2. Запросы изменения функций  $f$ .
3. Запросы изменения текущей структуры разрешения доступа в матрице  $M$ .

Следующий список описывает изменения каждого элемента состояния системы в данной модели. Конкретное решение по запросу включает в себя возможность производить следующие изменения в состоянии системы.

1. Изменение текущего доступа:

- получить доступ (добавить тройку (субъект, объект, вид доступа) в текущее множество  $b$ );

• отменить доступ (удалить тройку из текущего множества  $b$ ).

2. Изменение функций уровня конфиденциальности:

- изменить уровень конфиденциальности объекта;
- изменить текущий уровень конфиденциальности субъекта.

3. Изменение разрешения доступа:

- дать разрешение на доступ (добавить право доступа в соответствующий элемент матрицы доступов  $M$ );

- отменить разрешение на доступ (удалить право доступа из соответствующего элемента матрицы доступов  $M$ ).

Безопасность системы определяется с помощью трех свойств:

$ss$  — свойства простой безопасности (*simple security*);

$*$  — свойства звезды;

$ds$  — свойства дискреционной безопасности (*discretionary security*).

**Определение 3.2.** Доступ  $(s, o, r) \in S \times O \times R$  обладает  $ss$ -свойством относительно  $f = (f_s, f_o, f_c) \in F$ , если выполняется одно из условий:

- $r \in \{execute, append\}$ ;
- $r \in \{read, write\}$  и  $f_s(s) \geq f_o(o)$ .

**Определение 3.3.** Состояние системы  $(b, M, f) \in V$  обладает  $ss$ -свойством, если каждый элемент  $(s, o, r) \in b$  обладает  $ss$ -свойством относительно  $f$ .

**Определение 3.4.** Доступ  $(s, o, r) \in S \times O \times R$  обладает  $*$ -свойством относительно  $f = (f_s, f_o, f_c) \in F$ , если выполняется одно из условий:

- $r = execute$ ;
- $r = append$  и  $f_o(o) \geq f_c(s)$ ;
- $r = read$  и  $f_c(s) \geq f_o(o)$ ;
- $r = write$  и  $f_c(s) = f_o(o)$ .

**Определение 3.5.** Состояние системы  $(b, M, f) \in V$  обладает  $*$ -свойством, если каждый элемент  $(s, o, r) \in b$  обладает  $*$ -свойством относительно  $f$ .

**Определение 3.6.** Состояние системы  $(b, M, f) \in V$  обладает  $*$ -свойством относительно подмножества  $S' \subseteq S$ , если каждый элемент  $(s, o, r) \in b$ , где  $s \in S'$ , обладает  $*$ -свойством относительно  $f$ . При этом  $S \setminus S'$  называется множеством доверенных субъектов, т. е. субъектов, имеющих право нарушать требования  $*$ -свойства.

**Определение 3.7.** Состояние системы  $(b, M, f) \in V$  обладает  $ds$ -свойством, если для каждого элемента  $(s, o, r) \in b$  выполняется условие  $r \in M[s, o]$ .

**Определение 3.8.** Состояние системы  $(b, M, f)$  называется безопасным, если оно обладает  $*$ -свойством относительно  $S'$ ,  $ss$ -свойством и  $ds$ -свойством.

**Определение 3.9.** Реализация системы  $(x, y, z) \in \Sigma(Q, D, W, z_0)$  обладает  $ss$ -свойством ( $*$ -свойством,  $ds$ -свойством), если в последовательности  $(z_0, z_1, \dots)$  каждое состояние обладает  $ss$ -свойством ( $*$ -свойством,  $ds$ -свойством).

**Определение 3.10.** Система  $\Sigma(Q, D, W, z_0)$  обладает  $ss$ -свойством ( $*$ -свойством,  $ds$ -свойством), если каждая ее реализация обладает  $ss$ -свойством ( $*$ -свойством,  $ds$ -свойством).

**Определение 3.11.** Система  $\Sigma(Q, D, W, z_0)$  называется безопасной, если она обладает  $ss$ -свойством,  $*$ -свойством,  $ds$ -свойством одновременно.

Прокомментируем введенные выше свойства безопасности системы.

Во-первых, из обладания доступом  $*$ -свойством относительно  $f$  следует обладание этим доступом  $ss$ -свойством относительно  $f$ .

Во-вторых, из обладания системой  $ss$ -свойством следует, что в модели Белла—ЛаПадула выполняется запрет на чтение вверх, требуемый мандатной политикой безопасности. Кроме того,  $ss$ -свойство не допускает модификацию с использованием доступа  $write$ , если  $f_s(s) < f_o(o)$ . Таким образом, функция  $f_s(s)$  определяет для субъекта  $s$  верхний уровень конфиденциальности объектов, к которым он потенциально может получить доступ  $read$  или  $write$ .

В-третьих, поясним  $*$ -свойство. Если субъект  $s$  может понизить свой текущий доступ до  $f_c(s) = f_o(o)$ , то он может получить доступ  $write$  на объект  $o$ , но не доступ  $read$  на объекты  $o'$ , чей уровень  $f_o(o') > f_c(s)$ . Хотя при этом, возможно, справедливо неравенство

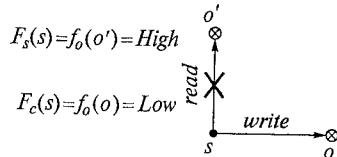


Рис. 3.1. Иллюстрация \*-свойства

$f_s(s) \geq f_o(o')$  и в каких-то других состояниях системы субъект  $s$  может получить доступ *read* на объект  $o'$ . Таким образом, \*-свойство исключает появление в системе неблагоприятного информационного потока сверху вниз и соответствует требованиям мандатной политики безопасности (рис. 3.1).

Проверка безопасности системы по определению в большинстве случаев не может быть реализована на практике в связи с тем, что при этом требуется проверка безопасности всех реализаций системы, а их бесконечно много. Следовательно, необходимо определить и обосновать иные условия безопасности системы, которые можно проверять на практике. В классической модели Белла—Лападула эти условия определяются для множества действий системы  $W$ .

**Теорема 3.1 (A1).** Система  $\Sigma(Q, D, W, z_0)$  обладает *ss*-свойством для любого начального состояния  $z_0$ , обладающего *ss*-свойством, тогда и только тогда, когда для каждого действия  $(q, d, (b^*, M^*, f^*), (b, M, f)) \in W$  выполняются условия 1, 2.

**Условие 1.** Каждый доступ  $(s, o, r) \in b^* \setminus b$  обладает *ss*-свойством относительно  $f^*$ .

**Условие 2.** Если  $(s, o, r) \in b$  и не обладает *ss*-свойством относительно  $f^*$ , то  $(s, o, r) \notin b^*$ .

**Доказательство.** Сначала докажем достаточность условий.

**Достаточность.** Пусть выполнены условия 1 и 2 и пусть  $(x, y, z) \in \Sigma(Q, D, W, z_0)$  — произвольная реализация системы. Тогда  $(x_t, y_t, (b_{t+1}, M_{t+1}, f_{t+1}), (b_t, M_t, f_t)) \in W$ , где  $z_{t+1} = (b_{t+1}, M_{t+1}, f_{t+1})$ ,  $z_t = (b_t, M_t, f_t)$  для  $t \in N_0$ .

Для  $(s, o, r) \in b_{t+1}$  выполняется одно из условий: или  $(s, o, r) \in b_{t+1} \setminus b_t$ , или  $(s, o, r) \in b_t$ . Из условия 1 следует, что состояние системы  $z_{t+1}$  пополнилось доступами, которые обладают *ss*-свойством относительно  $f^*$ . Из условия 2 следует, что доступы из  $b_t$ , которые не обладают *ss*-свойством относительно  $f^*$ , не входят в  $b_{t+1}$ . Следовательно, каждый доступ  $(s, o, r) \in b_{t+1}$  обладает *ss*-свойством относительно  $f^*$  и по определению состояние  $z_{t+1}$  обладает *ss*-свойством для  $t \in N_0$ . Так как по условию и состояние  $z_0$  обладает *ss*-свойством, то выбранная произвольная реализация  $(x, y, z)$  также обладает *ss*-свойством. Достаточность теоремы доказана.

**Необходимость.** Пусть система  $\Sigma(Q, D, W, z_0)$  обладает *ss*-свойством. Будем считать, что в множество  $W$  входят только те действия системы, которые встречаются в ее реализациях. Тогда для каждого  $(q, d, (b^*, M^*, f^*), (b, M, f)) \in W$  существует реализация  $(x, y, z) \in \Sigma(Q, D, W, z_0)$  и существует  $t \in N_0$ :  $(q, d, (b^*, M^*, f^*), (b_t, M_t, f_t)) \in W$ .

$(b, M, f)) = (x_t, y_t, z_{t+1}, z_t)$ . Так как реализация системы  $(x, y, z)$  обладает *ss*-свойством, то и состояние  $z_{t+1} = (b^*, M^*, f^*)$  обладает *ss*-свойством по определению. Следовательно, условия 1 и 2 выполняются. Необходимость теоремы доказана.

**Теорема 3.2 (A2).** Система  $\Sigma(Q, D, W, z_0)$  обладает \*-свойством относительно  $S' \subseteq S$  для любого начального состояния  $z_0$ , обладающего \*-свойством относительно  $S'$ , тогда и только тогда, когда для каждого действия  $(q, d, (b^*, M^*, f^*), (b, M, f)) \in W$  выполняются условия 1 и 2.

**Условие 1.** Для  $s \in S'$  доступ  $(s, o, r) \in b^* \setminus b$  обладает \*-свойством относительно  $f^*$ .

**Условие 2.** Для  $s \in S'$ , если доступ  $(s, o, r) \in b$  и не обладает \*-свойством относительно  $f^*$ , то  $(s, o, r) \notin b^*$ .

**Доказательство.** Доказательство осуществляется аналогично доказательству теоремы 3.1.

**Теорема 3.3 (A3).** Система  $\Sigma(Q, D, W, z_0)$  обладает *ds*-свойством для любого начального состояния  $z_0$ , обладающего *ds*-свойством, тогда и только тогда, когда для каждого действия  $(q, d, (b^*, M^*, f^*), (b, M, f)) \in W$  выполняются условия 1 и 2.

**Условие 1.** Для каждого  $(s, o, r) \in b^* \setminus b$  выполняется условие  $r \in M^*[s, o]$ ;

**Условие 2.** Если доступ  $(s, o, r) \in b$  и  $r \notin M^*[s, o]$ , то  $(s, o, r) \notin b^*$ .

**Доказательство.** Доказательство осуществляется аналогично доказательству теоремы 3.1.

**Теорема 3.4 (базовая теорема безопасности (БТБ)).** Система  $\Sigma(Q, D, W, z_0)$  безопасна для безопасного  $z_0$  тогда и только тогда, когда множество действий системы  $W$  удовлетворяет условиям теорем 3.1 — 3.3.

**Доказательство.** Базовая теорема безопасности следует из теорем 3.1 — 3.3.

Описанная классическая модель Белла—Лападула предоставляет общий подход к построению систем, реализующих мандатную политику безопасности. В модели определяется, какими свойствами должны обладать состояния и действия системы, чтобы система была безопасной согласно данным определениям. В модели не определяется точный порядок действий системы разграничения доступа по запросам на доступ субъектов к объектам.

**Пример 3.1.** Пусть субъект  $s$  запрашивает доступ *read* на объект  $o'$  (см. рис. 3.1). В данной ситуации система может выбрать один из двух возможных ответов по запросу.

1. Запретить субъекту  $s$  запрашиваемый им доступ *read* на объект  $o'$ .

2. Закрыть доступ *write* субъекта  $s$  на объект  $o$ . Повысить текущий уровень конфиденциальности  $f_s(s)$  до *High*. Разрешить субъекту  $s$  запрашиваемый им доступ *read* на объект  $o'$ .

Каждый из описанных путей соответствует требованиям безопасности модели Белла—ЛаПадула.

В реальных системах возможны более сложные ситуации, чем ситуация, описанная в примере 3.1. Кроме того, возможно использование в системе каких-то других видов доступа субъектов к объектам, которые потребуют дополнительного определения свойств безопасности, что не всегда легко сделать. В связи с этим большое значение имеет корректное определение свойств безопасности.

### 3.1.2. Пример некорректного определения свойств безопасности

При использовании классической модели Белла—ЛаПадула важно учитывать тот факт, что в ней отсутствует логическая связь условий выполнения системой свойств безопасности, данных в их определениях, с заложенными в модель условиями их проверки, необходимость и достаточность которых доказывается в теореме 3.4. Приведем пример некорректного определения основного свойства безопасности в модели Белла—ЛаПадула. Вместо  $ss$ -свойства используем абсурдное с точки зрения здравого смысла  $ds$ -свойство. Однако при этом не возникает никаких противоречий в логике доказательства теорем, определяющих условия проверки безопасности системы.

**Определение 3.12.** Доступ  $(s, o, r) \in S \times O \times R$  обладает  $ds$ -свойством относительно  $f = (f_s, f_o, f_c) \in F$ , если выполняется одно из условий:

- $r \in \{read, append, execute\}$ ;
- $r = write$  и  $f_c(s) \geq f_o(o)$ .

**Определение 3.13.** Состояние системы  $(b, M, f) \in V$  обладает  $ss$ -свойством, если каждый доступ  $(s, o, r) \in b$  обладает  $ss$ -свойством относительно  $f$ .

**Определение 3.14.** Состояние системы  $(b, M, f) \in V$  называется  $ds$ -безопасным, если обладает  $ss$ -свойством,  $ds$ -свойством одновременно.

**Определение 3.15.** Реализация  $(x, y, z)$  системы  $\Sigma(Q, D, W, z_0)$  обладает  $ss$ -свойством, если в последовательности  $(z_0, z_1, \dots)$  каждое состояние обладает  $ss$ -свойством.

**Определение 3.16.** Система  $\Sigma(Q, D, W, z_0)$  обладает  $ds$ -свойством, если каждая ее реализация обладает  $ds$ -свойством.

**Определение 3.17.** Система  $\Sigma(Q, D, W, z_0)$  называется  $ds$ -безопасной, если она обладает  $ss$ -свойством,  $ds$ -свойством,  $ds$ -свойством одновременно.

**Теорема 3.5.** Система  $\Sigma(Q, D, W, z_0)$  обладает  $ds$ -свойством для любого начального состояния  $z_0$ , обладающего  $ss$ -свойством,

тогда и только тогда, когда для каждого  $(q, d, (b^*, M^*, f^*), (b, M, f)) \in W$  выполняются условия 1 и 2.

**Условие 1.** Каждый доступ  $(s, o, r) \in b^* \setminus b$  обладает  $ss$ -свойством относительно  $f^*$ .

**Условие 2.** Если доступ  $(s, o, r) \in b$  и не обладает  $ss$ -свойством относительно  $f^*$ , то  $(s, o, r) \notin b^*$ .

**Доказательство.** Доказательство осуществляется аналогично доказательству теоремы 3.1.

**Теорема 3.6.** Система  $\Sigma(Q, D, W, z_0)$   $ss$ -безопасна для  $z_0$   $ss$ -безопасного состояния тогда и только тогда, когда множество действий системы  $W$  удовлетворяет условиям теорем 3.1, 3.3, 3.5.

**Доказательство.** Данная теорема следует из теорем 3.1, 3.3, 3.5.

### 3.1.3. Политика *low-watermark* в модели Белла—ЛаПадула

Среди проблем использования модели Белла—ЛаПадула особенно выделяется проблема отсутствия в модели определения порядка действий системы при переходе из состояния в состояние. Данная проблема иллюстрируется на примере политики *low-watermark* [2, 11].

Политика *low-watermark* реализуется в рассматриваемой далее модели мандатной политики целостности информации Биба. В то же время используемый в политике *low-watermark* подход к определению правил переходов системы из состояния в состояние может быть использован для демонстрации уязвимости определений безопасности в модели Белла—ЛаПадула.

При реализации политики *low-watermark* в модели Белла—ЛаПадула предлагается порядок безопасного функционирования системы в случае, когда по запросу субъекта ему всегда необходимо предоставлять доступ на запись в объект.

Основными элементами реализации политики *low-watermark* в модели Белла—ЛаПадула являются:

$S$  — множество субъектов системы;

$O$  — множество объектов системы;

$R = \{read, write\}$  — множество видов и прав доступа;

$B = \{b \subseteq S \times O \times R\}$  — множество возможных множеств текущих доступов в системе;

$(L, \leq)$  — решетка уровней конфиденциальности;

$(f_s, f_o) \in F = L^s \times L^o$  — двойка функций  $(f_s, f_o)$ , определяющих:  $f_s: S \rightarrow L$  — уровень доступа субъекта;  $f_o: O \rightarrow L$  — уровень конфиденциальности объекта;

$V = B \times F$  — множество состояний системы;

$OP = \{Read(), Write(), Reset()\}$  — множество операций (табл. 3.1).

Таблица 3.1. Условия и результаты выполнения операций

Операция	Условия выполнения	Результат выполнения
$Read(s, o)$	$f_s(s) \geq f_o(o)$	$f_s^* = f_s; b^* = b \cup \{(s, o, read)\}$
$Write(s, o)$	$f_s(s) \leq f_o(o)$	$f_s^* = f_s; f_o^*(o) = f_s(s);$ для каждого $o' \neq o$ справедливо равенство $f_o^*(o') = f_o(o')$ ; если $f_o^*(o) < f_o(o)$ , то содержимое $o$ очищается; $b^* = b \cup \{(s, o, read)\}$
$Reset(s, o)$	$f_s(s) > f_o(o)$	$f_s^* = f_s; b^* = b;$ для каждого $o' \neq o$ справедливо равенство $f_o^*(o') = f_o(o')$ ; $f_o^*(o) = \oplus L$ (максимальный уровень конфиденциальности)

$W \subseteq OP \times V \times V$  — множество действий системы, где тройка  $(op, (b^*, f^*), (b, f)) \in W$  означает, что система в результате выполнения операции  $op \in OP$  перешла из состояния  $(b, f)$  в состояние  $(b^*, f^*)$ .

В результате выполнения операции  $Write()$  уровень конфиденциальности объекта понижается до уровня доступа субъекта. Если это понижение реально происходит, то вся информация в объекте стирается. В результате выполнения операции  $Reset()$  уровень конфиденциальности объекта становится максимально возможным в системе. При этом в системах, в которых несколько субъектов одновременно могут запрашивать доступ к одному и тому же объекту, может потребоваться уточнение порядка использования операций  $Write()$  и  $Reset()$ .

Дадим определения  $ss$ -свойства и  $*$ -свойства для реализации политики *low-watermark* в модели Белла—Лападула.

**Определение 3.18.** Доступ  $(s, o, r) \in S \times O \times R$  обладает  $ss$ -свойством относительно  $f \in F$ , если  $r \in \{read, write\}$  и  $f_s(s) \geq f_o(o)$ .

**Определение 3.19.** Доступ  $(s, o, r) \in S \times O \times R$  обладает  $*$ -свойством относительно  $f \in F$ , если он удовлетворяет одному из условий:

- $r = read$  и  $f_s(s) \geq f_o(o)$ ;
- $r = write$  и  $f_s(s) = f_o(o)$ .

**Определение 3.20.** Состояние системы  $(b, f) \in V$  обладает  $ss$ -свойством ( $*$ -свойством), если каждый элемент  $(s, o, r) \in b$  обладает  $ss$ -свойством ( $*$ -свойством) относительно  $f$ .

**Определение 3.21.** Состояние системы называется безопасным, если оно обладает  $ss$ -свойством и  $*$ -свойством одновременно.

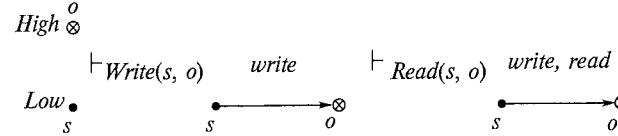


Рис. 3.2. Реализация неблагоприятного информационного потока

**Лемма 3.1.** Операции  $Read()$ ,  $Write()$ ,  $Reset()$  переводят систему из безопасного состояния в безопасное состояние.

**Доказательство.** Из описания операций  $Read()$ ,  $Write()$ ,  $Reset()$  следует, что в результате их выполнения последующее состояние системы обладает  $ss$ -свойством и  $*$ -свойством, если исходное состояние обладало  $ss$ -свойством и  $*$ -свойством. Лемма доказана.

Заметим, что условие стирания информации при выполнении операции  $Write()$  является существенным. Хотя при его отсутствии лемма 1 формально останется истинной. Однако в этом случае система не будет безопасной, так как возможно возникновение неблагоприятного информационного потока. Например, субъект, запрашивая доступ на запись в объект, понижает его уровень конфиденциальности, после чего он может запросить доступ на чтение из этого объекта (рис. 3.2).

Приведенный пример еще раз демонстрирует уязвимость определений безопасности в классической модели Белла—Лападула.

### 3.1.4. Безопасность переходов

Как уже было отмечено, в классической модели Белла—Лападула не описывается точный порядок действий системы при переходе из состояния в состояние. Частично этот недостаток устранен в [19], где предлагается оригинальное определение свойств безопасности модели Белла—Лападула с использованием вместо множества действий системы функций переходов.

При рассмотрении этого подхода будем считать  $R = \{read, write\}$  и для каждого  $s \in S$  справедливо равенство  $f_s(s) = f_c(s)$ . Исключим из рассмотрения матрицу доступов  $M$  и множество ответов системы  $D$ . Вместо множества действий системы  $W$  используем функцию переходов:

$T: Q \times V \rightarrow V$ , где  $T(q, v) = v^*$  — переход из состояния  $v$  по команде  $q$  в состояние  $v^*$ .

В этом случае будем обозначать систему через  $\Sigma(V, T, z_0)$ .

Далее переопределим  $ss$ -свойство и  $*$ -свойство. Так как основные ограничения на доступ  $write$  следуют из  $*$ -свойства, то в  $ss$ -свойстве зададим ограничения только на  $read$ .

**Определение 3.22.** Доступ  $(s, o, r) \in b$  обладает  $ss$ -свойством относительно  $f$ , если выполняется одно из условий:

- $r = \text{write}$ ;
- $r = \text{read}$  и  $f_s(s) \geq f_o(o)$ .

**Определение 3.23.** Доступ  $(s, x, r) \in b$  обладает  $*$ -свойством относительно  $f$ , если выполняется одно из условий:

- $r = \text{read}$  и не существует  $y \in O$ :  $(s, y, \text{write}) \in b$  и  $f_o(x) > f_o(y)$ ;
- $r = \text{write}$  и не существует  $y \in O$ :  $(s, y, \text{read}) \in b$  и  $f_o(y) > f_o(x)$ .

Заметим, что в  $*$ -свойстве не рассматривается уровень доступа субъекта посредника  $s$ . В этом нет необходимости, так как если требовать выполнения  $*$ -свойства и  $ss$ -свойства одновременно и считать, что субъект не может накапливать в себе информацию, то не возникает противоречий по существу с положениями мандатной политики безопасности.

Субъект может читать информацию из объектов с уровнем конфиденциальности не выше его уровня доступа, и при этом субъект не может реализовать неблагоприятный информационный поток сверху вниз.

По аналогии со свойствами классической модели Белла—Лападула определяются  $ss$ -свойства и  $*$ -свойства для состояния, реализации и системы в целом.

**Теорема 3.7 (A1-RW).** Система  $\Sigma(V, T, z_0)$  обладает  $ss$ -свойством для любого начального состояния  $z_0$ , обладающего  $ss$ -свойством, тогда и только тогда, когда функция переходов  $T(q, v) = v^*$  удовлетворяет условиям 1 и 2.

*Условие 1.* Если доступ  $(s, o, \text{read}) \in b^* \setminus b$ , то  $f_s^*(s) \geq f_o^*(o)$ .

*Условие 2.* Если доступ  $(s, o, \text{read}) \in b$  и  $f_s^*(s) < f_o^*(o)$ , то  $(s, o, \text{read}) \notin b^*$ .

*Доказательство.* Доказательство осуществляется аналогично доказательству теоремы 3.1.

**Теорема 3.8 (A2-RW).** Система  $\Sigma(V, T, z_0)$  обладает  $*$ -свойством для любого начального состояния  $z_0$ , обладающего  $*$ -свойством, тогда и только тогда, когда функция переходов  $T(q, v) = v^*$  удовлетворяет условиям 1 и 2.

*Условие 1.* Если  $\{(s, x, \text{read}), (s, y, \text{write})\} \cap (b^* \setminus b) \neq \emptyset$  и  $\{(s, x, \text{read}), (s, y, \text{write})\} \subset b^*$ , то  $f_o^*(y) \geq f_o^*(x)$ .

*Условие 2.* Если  $\{(s, x, \text{read}), (s, y, \text{write})\} \subseteq b$  и  $f_o^*(y) < f_o^*(x)$ , то  $\{(s, x, \text{read}), (s, y, \text{write})\} \not\subseteq b^*$ .

*Доказательство.* Доказательство осуществляется аналогично доказательству теоремы 3.1.

**Теорема 3.9 (БТБ-RW).** Система  $\Sigma(V, T, z_0)$  безопасна для безопасного начального состояния  $z_0$  тогда и только тогда, когда выполнены условия теорем 3.7 и 3.8.

*Доказательство.* Теорема 3.9 следует из теорем 3.7, 3.8.

В данном подходе  $*$ -свойство определено таким образом, что его смысл — предотвращение возникновения неблагоприятных информационных потоков сверху вниз становится более ясным, чем при использовании функции  $f_c()$  в классической модели Бел-

ла—Лападула. В этом заключена основная ценность рассмотренных определений основных свойств безопасности.

Далее определим основные свойства безопасности модели Белла—Лападула, используя определения свойств безопасности функции переходов  $T()$ . При этом на  $T()$  накладывается дополнительное ограничение — за один шаг работы системы вносится только одно изменение в одном из параметров, используемом при определении свойств безопасности, т. е. изменяется на один элемент либо множество текущих доступов, либо одно из значений одной из функций. При этом остальные параметры остаются неизменными.

**Определение 3.24.** Функция переходов  $T(q, (b, f)) = (b^*, f^*)$  обладает  $ss$ -свойством, если выполнены следующие условия:

- если  $(s, o, \text{read}) \in b^* \setminus b$ , то  $f_s(s) \geq f_o(o)$  и  $f^* = f$ ;
- если  $f_s(s) \neq f_s^*(s)$ , то  $f_o^* = f_o$ ,  $b^* = b$ , для  $s' \neq s$  справедливо равенство  $f_s(s') = f_s^*(s')$ , и если  $(s, o, \text{read}) \in b$ , то  $f_s^*(s) \geq f_o(o)$ ;
- если  $f_o(o) \neq f_o^*(o)$ , то  $f_s^* = f_s$ ,  $b^* = b$ , для  $o' \neq o$  справедливо равенство  $f_o(o') = f_o^*(o')$ , и если  $(s, o, \text{read}) \in b$ , то  $f_s(s) \geq f_o^*(o)$ .

**Определение 3.25.** Функция переходов  $T(q, (b, f)) = (b^*, f^*)$  обладает  $*$ -свойством, если выполнены следующие условия:

- если  $\{(s, x, \text{read}), (s, y, \text{write})\} \subseteq b^*$  и  $\{(s, x, \text{read}), (s, y, \text{write})\} \not\subseteq b$ , то  $f_o(y) \geq f_o(x)$  и  $f^* = f$ ;
- если  $f_o^*(y) \neq f_o(y)$ , то  $b = b^*$ ,  $f_s^* = f_s$ , для  $z \neq y$  справедливо равенство  $f_o(z) = f_o^*(z)$ , и если  $\{(s, x, \text{read}), (s, y, \text{write})\} \subseteq b$ , то  $f_o^*(y) \geq f_o(x)$ , или если  $\{(s, y, \text{read}), (s, x, \text{write})\} \subseteq b$ , то  $f_o(x) \geq f_o^*(y)$ .

**Определение 3.26.** Функция переходов  $T()$  безопасна, если она обладает  $ss$ -свойством и  $*$ -свойством.

**Теорема 3.10 (БТБ-T).** Система  $S(V, T, z_0)$  безопасна для безопасного начального состояния  $z_0$ , если ее функция переходов безопасна.

*Доказательство.* По аналогии с доказательством теоремы 3.1 необходимо показать, что если функция переходов  $T(q, (b, f)) = (b^*, f^*)$  безопасна, то состояние  $(b^*, f^*)$  безопасно по условиям определений 3.22 и 3.23. Этот факт следует из условий определений 3.24 и 3.25.

Таким образом, при безопасном начальном состоянии любая реализация системы, а следовательно, и система в целом будут безопасными. Теорема доказана.

В рамках изучения определений безопасности функции переходов можно рассмотреть вопрос о возможностях смены уровня конфиденциальности субъектов и объектов. Примем следующие обозначения:

- для  $x \in S$ ,  $c_s(x) \subseteq S$  — множество субъектов, имеющих право менять уровень доступа субъекта  $x$ ;
- для  $y \in O$ ,  $c_o(y) \subseteq S$  — множество субъектов, имеющих право менять уровень конфиденциальности объекта  $y$ .

В этом случае в параметры функции переходов необходимо внести еще один параметр, определяющий субъекта, дающего запрос системы.

**Определение 3.27.** Функция переходов  $T(s, q, (b, f)) = (b^* f^*)$  безопасна в смысле администрирования, если выполнены следующие условия:

- если  $f_s^*(x) \neq f_s(x)$ , то  $s \in c_s(x)$ ;
- если  $f_o^*(y) \neq f_o(y)$ , то  $s \in c_o(y)$ .

Таким образом, задавая множества  $c_s(x)$  и  $c_o(y)$ , можно рассматривать случаи, когда все субъекты могут менять уровни конфиденциальности объектов и уровни доступа субъектов; никто не может или только системный администратор может менять уровни конфиденциальности объектов и уровни доступа субъектов.

### 3.1.5. Модель мандатной политики целостности информации Биба

Классическая модель Белла—Лападула, в первую очередь, ориентирована на обеспечение защиты от угрозы конфиденциальности информации. В то же время ее математическая основа используется в модели Биба [11], реализующей мандатную политику целостности.

В модели Биба рассматриваются следующие доступы субъектов на объекты и субъекты:

- *modify* — доступ субъекта на модификацию объекта (аналог доступа *write* в модели Белла—Лападула);
- *invoke* — доступ на обращение субъекта к субъекту;
- *observe* — доступ на чтение субъекта к объекту (аналог доступа *read* в модели Белла—Лападула);
- *execute* — доступ на выполнение.

Основными элементами модели Биба являются:

$S$  — множество субъектов;

$O$  — множество объектов;

$(LI, \leq)$  — решетка уровней целостности, например:  $LI = \{I (important), VI (very important), C (crucial)\}$ , где  $I < VI < C$ ;

$RI = \{\text{modify}, \text{invoke}, \text{observe}, \text{execute}\}$  — множество видов доступа и видов прав доступа;

$B = \{b \subseteq S \times O \times RI\}$  — множество возможных множеств текущих доступов в системе;

$(i_s, i_o, i_c) \in I = LI^s \times LI^o \times LI^c$  — тройка функций  $(i_s, i_o, i_c)$ , определяющих:  $i_s: S \rightarrow LI$  — уровень целостности субъекта;  $i_o: O \rightarrow LI$  — уровень целостности объекта;  $i_c: S \rightarrow LI$  — текущий уровень целостности субъекта; при этом для каждого  $s \in S$  выполняется условие  $i_c(s) \leq i_s(s)$ ;

$V = B \times I$  — множество состояний системы.

Элементы модели, необходимые для реализации дискреционной политики безопасности в модели Биба, не отличаются от аналогичных элементов в модели Белла—Лападула и рассматриваться не будут. Так же как в классической модели Белла—Лападула, в модели Биба не рассматриваются вопросы администрирования уровней целостности субъектов и объектов.

В отличие от требований безопасности классической модели Белла—Лападула требования безопасности в модели Биба являются динамическими, для их описания используются элементы текущего и последующего состояний системы.

**Определение 3.28.** Доступ  $(s, o, r) \in S \times O \times RI$  соответствует требованиям политики *low-watermark* для субъектов относительно  $i = (i_s, i_o, i_c) \in I$ , если выполняется одно из условий:

- $r = \text{execute}$ ;
- $r = \text{modify}$  и  $i_c(s) \geq i_o(o)$ ;
- $r = \text{invoke}$ ,  $o \in S$  и  $i_c(s) \geq i_c(o)$ ;
- $r = \text{observe}$  и после получения доступа в последующем состоянии системы  $i_c(s)' = i_c(s) \otimes i_o(o)$ , где  $\otimes$  — наибольшая нижняя граница элементов решетки  $(LI, \leq)$ ;  $i_c(s)'$  — значение функции текущего уровня целостности субъекта в последующем состоянии системы.

**Определение 3.29.** Доступ  $(s, o, r) \in S \times O \times RI$  соответствует требованиям политики *low-watermark* для объектов относительно  $i = (i_s, i_o, i_c) \in I$ , если выполняется одно из условий:

- $r \in \{\text{execute}, \text{invoke}, \text{observe}\}$ ;
- $r = \text{modify}$  и после получения доступа в последующем состоянии системы  $i_o(o)' = i_c(s) \otimes i_o(o)$ , где  $i_o(o)'$  — значение функции уровня целостности объекта в последующем состоянии системы.

Динамическое изменение функций уровня целостности субъекта или объекта может потребовать уточнения требований политики *low-watermark*. Например, если в системе допускается подача субъектом одновременно нескольких запросов на доступ, то результат их выполнения может зависеть от последовательности выполнения запросов. Если субъект запросил одновременно доступы *observe* и *modify*, то предоставление в первую очередь доступа *observe* может привести к понижению уровня целостности субъекта, а следовательно, может сделать невозможным получение им доступа *modify*. Кроме того, понижение уровня целостности объекта при доступе *modify* к нему может привести к модификации информации с высоким уровнем целостности субъектом с низким уровнем целостности. Аналогичный пример для случая понижения уровня конфиденциальности объекта анализируется при описании политики *low-watermark* в модели Белла—Лападула.

Возможным путем уточнения политики *low-watermark* является выполнение требования неизменности значений функций  $(i_s, i_o,$

$i_c$ ) для всех субъектов и объектов во всех состояниях системы. В этом случае всегда справедливо равенство  $i_s = i_o$ .

**Определение 3.30.** Доступ  $(s, o, r) \in S \times O \times RI$  соответствует требованиям политики *low-watermark* при неизменных значениях функций  $(i_s, i_o)$ , если выполняется одно из условий:

- $r \in \{execute, observe\}$ ;
- $r = modify$  и  $i_s(s) \geq i_o(o)$ ;
- $r = invoke, o \in S$  и  $i_s(s) \geq i_s(o)$ .

Выполнение требований политики *low-watermark* при неизменных значениях функций  $(i_s, i_o)$  позволяет предотвратить модификацию субъектом информации в объекте с более высоким или не сравнимым с субъектом уровнем целостности. В то же время отсутствие ограничений на доступ *observe* может позволить субъекту реализовать информационный поток от объекта с меньшим уровнем целостности в объект с большим уровнем целостности. Для предотвращения угрозы возникновения информационных потоков данного вида возможно использование строгой политики *low-watermark* при неизменных значениях функций  $(i_s, i_o)$ .

**Определение 3.31.** Доступ  $(s, o, r) \in S \times O \times RI$  соответствует требованиям строгой политики *low-watermark* при неизменных значениях функций  $(i_s, i_o)$ , если выполняется одно из условий:

- $r = execute$ ;
- $r = modify$  и  $i_s(s) \geq i_o(o)$ ;
- $r = invoke, o \in S$  и  $i_s(s) \geq i_s(o)$ ;
- $r = observe$  и  $i_s(s) \leq i_o(o)$ .

По аналогии с классической моделью Белла—Лападула в модели Биба можно определить требования политики *low-watermark* для состояния и системы в целом. После чего можно сформулировать и доказать для требований политики *low-watermark* теоремы, аналогичные теоремам 3.1—3.4 классической модели Белла—Лападула.

## 3.2. МОДЕЛЬ СИСТЕМ ВОЕННЫХ СООБЩЕНИЙ

### 3.2.1. Общие положения и основные понятия

Построенная на основе модели Белла—Лападула модель систем военных сообщений (СВС) была предложена в 1984 г. [16]. Она ориентирована, в первую очередь, на системы приема, передачи и обработки почтовых сообщений, реализующих мандатную политику безопасности. В то же время предложенные в модели СВС подходы к определению основных свойств безопасности могут быть использованы для построения и анализа произвольных систем мандатного разграничения доступа.

Определения основных понятий, используемых при описании свойств модели СВС, либо давались в модели Белла—Лападула, либо их смысл интуитивно ясен, например: пользователь, идентификатор пользователя, уровни конфиденциальности, доступ, множество доступов. Дополнительно даются следующие определения.

**Определение 3.32.** Роль пользователя — совокупность прав пользователя, определяемая характером выполняемых им действий в системе. Пользователь может менять свои роли во время работы в системе.

**Определение 3.33.** Объект — это одноуровневый блок информации. Объект — минимальный классифицируемый блок информации, он не содержит других объектов, т. е. не является многоуровневым.

**Определение 3.34.** Контейнер — это многоуровневая информационная структура. Контейнер может содержать объекты и другие контейнеры.

**Определение 3.35.** Сущность — объект или контейнер.

**Определение 3.36.** Способ доступа к содержимому контейнера (*CCR*) — атрибут контейнеров, определяющий порядок обращения к его содержимому (с учетом уровня конфиденциальности контейнера или с учетом только уровня конфиденциальности сущности контейнера, к которой осуществляется обращение).

**Определение 3.37.** Идентификатор сущности — уникальный номер или имя сущности.

**Определение 3.38.** Непосредственная ссылка — ссылка на сущность, совпадающая с идентификатором сущности.

**Определение 3.39.** Косвенная ссылка — ссылка на сущность, являющуюся частью контейнера, через последовательность двух и более ссылок на сущности, в которой только первая ссылка есть идентификатор (непосредственная ссылка).

**Определение 3.40.** Операция — функция, которая может быть выполнена над сущностями. В СВС основными операциями над сообщениями являются:

- операции над входящими сообщениями;
- операции над исходящими сообщениями;
- операции хранения и получения сообщений.

**Определение 3.41.** Сообщение — особый тип сущностей, имеющихся в СВС. В большинстве случаев сообщение есть контейнер, хотя в некоторых системах, только получающих сообщения, оно может быть и объектом. Каждое сообщение как контейнер содержит несколько сущностей, описывающих его параметры, например: кому (*to*), от кого (*from*), информация (*info*), дата-время-группа (*date-time-group*), текст (*text*), безопасность (*security*).

В модели СВС описываются четыре постулата безопасности, выполнение которых необходимо для ее корректной работы.

1. Системный офицер безопасности корректно разрешает доступ пользователей к сущностям и назначает уровни конфиденциальности устройств и множества ролей.

2. Пользователь назначает или переназначает корректные уровни конфиденциальности сущностей, когда создает или редактирует в них информацию.

3. Пользователь корректно направляет сообщения по адресатам и определяет множества доступа к созданным им самим сущностям.

4. Пользователь правильно определяет атрибут *CCR* контейнеров.

### 3.2.2. Неформальное описание модели СВС

Для лучшего пояснения свойств модели СВС ее описание составлено из двух частей. В первой части представлены основные свойства модели, описанные неформально. Во второй части свойства модели СВС выражены с использованием математической модели.

Всего в модели СВС описываются десять неформальных свойств.

1. *Авторизация*. Пользователь может осуществлять операцию над сущностями только в том случае, если идентификатор пользователя или его роль присутствуют во множестве доступов сущности вместе с данной операцией и правильными индексами операндов сущностей.

2. *Иерархия уровней конфиденциальности*. Уровень конфиденциальности любого контейнера, по крайней мере, не ниже максимума уровней конфиденциальности сущностей, в нем содержащихся.

3. *Безопасный перенос информации*. Информация, извлекаемая из объекта, наследует уровень конфиденциальности объекта. Информация, внедряемая в объект, не должна иметь уровень конфиденциальности выше, чем сам объект.

4. *Безопасный просмотр*. Пользователь может просматривать (на некотором устройстве вывода) сущность с уровнем конфиденциальности не выше уровня доступа пользователя и уровня конфиденциальности устройства вывода.

5. *Доступ к сущностям с атрибутом CCR*. Пользователь может иметь доступ косвенно к сущностям контейнера с атрибутом *CCR* только в том случае, если пользователь имеет уровень доступа не ниже, чем уровень конфиденциальности контейнера.

6. *Доступ по косвенной ссылке*. Пользователь может использовать идентификатор контейнера, чтобы получить через него косвенную ссылку на сущность только в том случае, если он авторизован для просмотра этой сущности с использованием этой ссылки.

7. *Требования пометки вывода*. Любая сущность, просматриваемая пользователем, должна быть помечена ее уровнем конфиденциальности.

8. *Определение доступов, множества ролей, уровней устройств*. Только пользователь с ролью «системный офицер безопасности» может определять права доступа и множество ролей пользователя, а также уровень конфиденциальности устройств вывода иноля, а также уровень конфиденциальности устройств вывода информации. Выбор роли из множества авторизованных ролей пользователя может быть осуществлен только самим пользователем или пользователем с ролью «системный офицер безопасности».

9. *Безопасное понижение уровня конфиденциальности*. Никакой уровень конфиденциальности не может быть понижен, за исключением тех случаев, когда понижение выполняет пользователь с соответствующей ролью.

10. *Безопасное отправление сообщений*. Никакое черновое сообщение не может быть отправлено, за исключением случая, когда это делает пользователь с ролью «отправитель».

### 3.2.3. Формальное описание модели СВС

#### Безопасное состояние

Система модели СВС представляется конечным автоматом. Основными элементами модели являются:

*OP* — множество операций;

$(L, \leq)$  — решетка уровней конфиденциальности;

*UI* — множество идентификаторов пользователей;

*RL* — множество ролей пользователей;

*US* — множество пользователей. Для любого пользователя  $u \in US$  — множество пользователей. Для любого пользователя,  $R(u) \subseteq RL \in US$ ,  $CU(u) \in L$  — уровень доступа пользователя,  $RO(u) \subseteq RL$  — текущее множество его авторизованных ролей,  $RO(u) \subseteq RL$  — текущее множество ролей пользователя;

*RF* — множество ссылок, состоящее из двух подмножеств ( $DR$  — множество непосредственных ссылок; *IR* — множество косвенных ссылок). Хотя точная природа этих ссылок не важна, предположим, что непосредственные ссылки есть целые числа. Каждая косвенная ссылка есть конечная последовательность целых чисел  $\langle n_1, \dots, n_m \rangle$ , где  $\langle n_i \rangle$  — непосредственная ссылка;

*VS* — множество значений сущностей (например, множество битовых или символьных строк);

*TY* — множество типов сущностей, включающее в себя, например, *DM* — черновое сообщение — и *RM* — отправленное сообщение;

*ES* — множество сущностей. При этом для  $e \in ES$  определяются:

- $CE(e) \in L$  — уровень конфиденциальности этой сущности;
- $AS(e) \subseteq (UI \cup RL) \times OP \times N$  — множество троек, определяющих множество разрешенных доступов к сущности, где  $(u, op, k) \in AS(e)$  тогда и только тогда, когда пользователь  $u$  с заданными идентификатором или ролью авторизован для выполнения операции  $op$  над сущностью  $e$ , ссылка на которую является  $k$ -м параметром операции  $op$ ;
- $T(e) \in TY$  — тип сущности; если  $T(e_1) = T(e_2)$ , тогда обе сущности  $e_1$  и  $e_2$  — или контейнеры, или объекты;
- $V(e) \in VS$  — значение сущности;
- $H(e) = \langle e_1, \dots, e_n \rangle$  — структура сущности, где  $e_i$  —  $i$ -я сущность, непосредственно содержащаяся в контейнере  $e$ ;
- $CCR(e)$  — атрибут сущностей-контейнеров;  $CCR(e)$  истинно тогда и только тогда, когда контейнер  $e$  помечен  $CCR$ , иначе  $CCR(e)$  ложно;
- $RE(e) \in UI$  — идентификатор отправителя для сущности сообщения;

$O \subseteq ES$  — множество сущностей-контейнеров, описывающих устройства вывода информации. Для каждого  $o \in O$  определены значения двух функций:

- $D(o)$  — множество упорядоченных пар  $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ , где каждый  $x_i$  есть некоторый пользователь или сущность и соответствующее  $y_i$  есть «явление»  $x_i$ ; или некоторая ссылка, или идентификатор, или результат применения одной из описанных функций к  $x_i$ . Таким образом, каждое  $D(o)$  описывает, что могут видеть пользователи на устройстве вывода информации. При этом  $((x, V(x)) \in D(o)) \Rightarrow (x \in H(o))$ , т.е. если значение сущности можно получить на устройстве вывода, то эта сущность входит в состав контейнера устройства вывода;
- $CD(o)$  — максимальный уровень конфиденциальности информации, разрешенный для вывода на  $o$ ;

$U: UI \rightarrow US$  — функция идентификаторов пользователей;

$E: RF \rightarrow ES$  — функция ссылок на сущности. При этом для любого  $n > 1$   $E(\langle i_1, \dots, i_n \rangle) = e$  тогда и только тогда, когда  $E(\langle i_1, \dots, i_{n-1} \rangle) = e^*$ , где  $e^*$  — контейнер такой, что  $e$  — это  $i_n$ -й элемент  $H(e^*)$ . Для любой ссылки  $r \in RF$ , если  $E(r) = e$ , то говорим, что  $r$  есть ссылка на  $e$ . Кроме того, косвенная ссылка  $\langle i_1, \dots, i_n \rangle$  на сущность  $e$  определяет последовательность сущностей  $e_1, \dots, e_{n-1}$  таких, что каждая сущность  $e_j$  для  $1 < j < n$  определяется через непосредственную ссылку  $\langle i_j \rangle$ , и  $e_j$  есть  $j$ -я сущность контейнера  $e_{j-1}$ . Будем говорить, что такая косвенная ссылка  $\langle i_1, \dots, i_n \rangle$  основывается на каждой сущности  $e_j$ , где  $0 < j < n$ ;

$LO: UI \rightarrow RF$  — функция входов пользователей (*logon*) в систему.

Для произвольной функции  $F: X \rightarrow Y$  обозначим:

- $dom(F)$  — область определения функции  $F()$ ;
- $rng(F)$  — область значений функции  $F()$ ;
- $F^{-1}(Z) = \{x \in X: F(x) \in Z\}$ , где  $Z \subseteq Y$ .

**Определение 3.42.** Состояние системы есть упорядоченная тройка  $s = (U, E, LO)$ , где  $U$  — функция идентификаторов;  $E$  — функция ссылок;  $LO$  — функция входов. При этом:

- $dom(LO) \subseteq dom(U)$ ;
- $rng(LO) \subseteq E^{-1}(O)$ ;
- для  $o \in rng(E) \cap O$  и  $(x, y) \in D(o)$  выполняется условие  $x \in rng(E) \cup rng(U)$  — только информация о пользователях или сущностях, существующих в данном состоянии может быть выведена на устройствах вывода;
- для ссылки  $r \in dom(E)$  выполняется условие  $((x, r) \in D(o)) \Rightarrow (E(r) = x)$  — если выводится ссылка, то определена сущность, на которую она указывает;
- для  $u_1, u_2 \in dom(LO)$  выполняется условие  $(E(LO(u_1)) = E(LO(u_2))) \Rightarrow (u_1 = u_2)$  — в данном состоянии с данного устройства вывода информации может осуществлять вход в систему только один пользователь.

Если дано состояние системы  $s = (U, E, LO)$ , то используем следующие обозначения:

- $u_s = U(u)$  — пользователь с идентификатором  $u$  в состоянии  $s$ ;
- $r_s = E(r)$  — сущность по ссылке  $r$  в состоянии  $s$ ;
- $u'_s = E(LO(u))$  — сущность-устройство вывода информации, с которого осуществил вход в систему пользователь с идентификатором  $u$  в состоянии  $s$ .

**Определение 3.43.** Состояние  $s$  безопасно, если для любых двух сущностей  $x, y \in rng(E)$ , устройства вывода информации  $o \in O \cap \cap rng(E)$ , идентификатора пользователя  $w \in dom(LO)$  и пользователя  $u \in rng(U)$  выполняются условия:

- $(x \in H(y)) \Rightarrow (CE(x) \leq CE(y))$  — уровень конфиденциальности сущности в составе контейнера не выше уровня конфиденциальности контейнера;
- $(x \in H(w'_s)) \Rightarrow (CU(w_s) \geq CE(x))$  — уровень доступа пользователя с данным идентификатором не ниже уровня конфиденциальности сущностей, выводимых на устройстве вывода информации, с которого он осуществил вход в систему;
- $((x, V(x)) \in D(o)) \Rightarrow ((x, CE(x)) \in D(o))$  — если выводится значение сущности, то должен быть выведен и ее уровень конфиденциальности;
- $RO(u) \subseteq R(u)$  — текущее множество ролей пользователя должно быть частью множества ролей, на которые он авторизован;
- $CD(o) \geq CE(o)$  — максимальный уровень конфиденциальности сущностей, разрешенный для вывода на устройстве вывода информации, должен быть не ниже текущего уровня конфиденциальности устройства вывода.

## Безопасность переходов

**Определение 3.44.** Система  $S(T, s_0)$  — пара элементов:

- $s_0$  — начальное состояние системы;
- $T: UI \times I \times S \rightarrow S$  — функция переходов системы, где  $S$  — множество возможных состояний системы;  $I$  — множество запросов системы. При этом каждый запрос  $i \in I$  имеет вид  $\langle op, x_1, \dots, x_n \rangle$ , где  $op \in OP$  и  $x_j \in RF \cup UI \cup VS$  для  $1 \leq j \leq n$ , т.е. параметром запроса может быть ссылка на сущность, идентификатор пользователя или значение сущности.

**Определение 3.45.**  $\pi: N_0 \rightarrow UI \times I \times S$  — история системы; при этом для  $n \in N_0$  выполняется условие: если  $\pi(n) = (u, i, s)$  и  $\pi(n+1) = (u^*, i^*, s^*)$ , то  $T(u, i, s) = s^*$ .

**Определение 3.46.** Состояния  $s = (U, E, LO)$  и  $s^* = (U^*, E^*, LO^*)$  эквивалентны, за исключением некоторого множества ссылок  $r \subset dom(E)$  (обозначим через  $s \sim^r s^*$ ) тогда и только тогда, когда выполняются условия:

- $U = U^*$ ;
- $LO = LO^*$ ;
- $dom(E) = dom(E^*)$ ;
- для любой функции  $F$ , за исключением  $V$  (функции значений сущностей), справедливо равенство  $F = F^*$ ;
- для любой ссылки  $r \in dom(E) \setminus \rho$  справедливо равенство  $V(r_s) = V(r_{s^*})$ .

**Определение 3.47.** Запрос  $(u, i, s) \in UI \times I \times S$  потенциально модифицирует сущность по ссылке  $r \in dom(E)$  тогда и только тогда, когда существуют состояния  $s_1, s_1^*$  и  $\rho \subset dom(E)$  такие, что  $s_1 \sim^{\rho} s$ ,  $T(u, i, s_1) = s_1^*$ , и для некоторой функции  $F()$  справедливо равенство  $F(r_{s_1}) \neq F(r_{s_1^*})$ . При этом сущность по ссылке  $u \in dom(E)$  называется источником потенциальной модификации тогда и только тогда, когда выполняется одно из условий:

- $y = r$ ;
- существуют состояния  $s_2, s_2^*$  такие, что  $s_1 \sim^{\{y\}} s_2$ ,  $T(u, i, s_2) = s_2^*$  и  $F(r_{s_1}) \neq F(r_{s_2^*})$ .

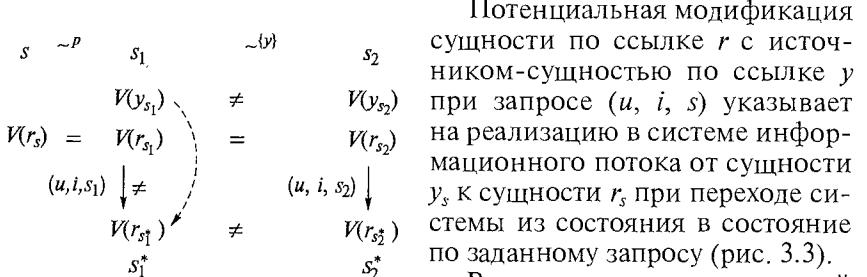


Рис. 3.3. Потенциальная модификация с источником

ный поток, возникающий в системе, может не приводить к изменению модифицируемой сущности, например в случае, если она по значению совпадает с сущностью-источником.

Дадим восемь определений смыслов безопасности функции переходов.

**Определение 3.48.** Функция переходов  $T()$  безопасна в смысле доступов к сущностям тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняется одно из условий: или  $s = s^*$  (запрос отвергается), или  $(op \in i \cap OP \text{ и } r_k \in i \cap RF) \Rightarrow (\text{или } (u, op, k) \in AS(r_k)_s)$ , или существует роль  $l \in RO(u_s)$ :  $(l, op, k) \in AS((r_k)_s)$ .

**Определение 3.49.** Функция переходов  $T()$  безопасна в смысле модификации сущностей тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняется условие (сущность по ссылке  $x$  потенциально модифицируется с источником-сущностью по ссылке  $y$ )  $\Rightarrow (CE(x_s) \geq CE(y_s))$ .

**Определение 3.50.** Функция переходов  $T()$  безопасна в смысле доступов к контейнерам с атрибутом  $CCR$  тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняется условие (сущность по ссылке  $z$  потенциально модифицируется с источником-сущностью по косвенной ссылке  $r \in i \cap IR$ , основанной на сущности по ссылке  $u$  с истинным атрибутом  $CCR(y_s)$ )  $\Rightarrow (CU(u_s) \geq CE(y_s))$ .

**Определение 3.51.** Функция переходов  $T()$  безопасна в смысле доступов по косвенной ссылке тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняется (для непосредственной ссылки  $x \in DR$  выполняется условие  $(x_s, x) \in D(u_s)$ ) и существует косвенная ссылка  $r \in i \cap RF$ :  $r_s = x_s$  и сущность по ссылке  $r$  основана на контейнере по ссылке  $z$  с истинным атрибутом  $CCR(z_s)$   $\Rightarrow (CU(u_s) \geq CE(z_s))$  — значение непосредственной ссылки на сущность может быть получено по косвенной ссылке с учетом правил доступа к контейнерам с атрибутом  $CCR$ .

**Определение 3.52.** Функция переходов  $T()$  безопасна в смысле администрирования тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняются условия:

- (для ссылки на устройство вывода  $o \in E^{-1}(O)$  истинно  $CD(o_s) \neq CD(o_{s^*})$  или для идентификатора пользователя  $x \in dom(U)$  выполняется одно из условий: или  $CU(x_s) \neq CU(x_{s^*})$ , или  $R(x_s) \neq R(x_{s^*})$ )  $\Rightarrow (security\_officer \in RO(u_s))$  — максимальный уровень конфиденциальности информации, выводимой на устройстве вывода, уровень доступа пользователя или множество его авторизованных ролей может изменить только пользователь с ролью «системный офицер безопасности»;
- (для идентификатора пользователя  $x \in dom(U)$  истинно  $RO(x_s) \neq RO(x_{s^*})$ )  $\Rightarrow (\text{или } x_s = u_s, \text{ или } security\_officer \in RO(u_s))$  — изменить множество текущих ролей пользователя можно только

самому пользователю или пользователю с ролью «системный офицер безопасности».

**Определение 3.53.** Функция переходов  $T()$  безопасна в смысле понижения уровня конфиденциальности сущностей тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняется условие (для  $x \in \text{dom}(E) \setminus E^{-1}(\{u_s\})$ ) справедливо неравенство  $CE(x_s) > CE(x_{s^*}) \Rightarrow (\text{downgrader} \in RO(u_s))$  — за исключением сущности устройства вывода, на которой вошел в систему пользователь, уровень конфиденциальности сущности может понижать только пользователь со специальной ролью.

**Определение 3.54.** Функция переходов  $T()$  безопасна в смысле отправки сообщений тогда и только тогда, когда для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , выполняются условия:

- $(T(x_s) = RM) \Rightarrow (T(x_{s^*}) = RM \text{ и } RE(x_s) = RE(x_{s^*}))$  — у сообщения тип и идентификатор отправителя не могут быть изменены после отправки ( $RM$  — тип «отправленное сообщение»);

- $(T(x_s) \neq RM \text{ и } T(x_{s^*}) = RM) \Rightarrow (T(x_s) = DM, RE(x_{s^*}) = u, \text{ существует ссылка } r_s = x_s \text{ и } i \text{ есть операция вида } \langle \text{release}, r \rangle, \text{ где } \text{releaser} \in RO(u_s))$  — может быть с использованием некоторой ссылки отправлено только черновое сообщение ( $DM$  — тип «черновое сообщение») и только пользователем с ролью «отправитель»; при этом идентификатор этого пользователя устанавливается в соответствующее поле сообщения.

**Теорема 3.11 (БТБ-СВС).** Каждое состояние системы  $\Sigma(T, s_0)$  безопасно, если  $s_0$  безопасно и функция переходов  $T()$  удовлетворяет следующим требованиям для  $u, i, s, s^*$  таких, что  $T(u, i, s) = s^*$ , и для  $x, y \in RF, w \in UI$  выполняются условия:

- $(x_s \notin H(y_s) \text{ и } x_{s^*} \in H(y_{s^*})) \Rightarrow (CE(x_{s^*}) \leq CE(y_{s^*}))$ ;
- $(x_s \in H(y_s) \text{ и } CE(x_{s^*}) > CE(y_{s^*})) \Rightarrow (x_{s^*} \notin H(y_{s^*}))$ ;
- $(xs \notin H(w_s) \text{ и } x_{s^*} \in H(w_{s^*})) \Rightarrow (CE(x_{s^*}) \leq CU(w_{s^*}))$ ;
- $(x_s \in H(w_s) \text{ и } CE(x_{s^*}) > CU(w_{s^*})) \Rightarrow (x_{s^*} \notin H(w_{s^*}))$ ;
- $((x_s, V(x_s)) \notin D(w_s) \text{ и } (x_{s^*}, V(x_{s^*})) \in D(w_{s^*})) \Rightarrow ((x_{s^*}, CE(x_{s^*})) \in D(w_{s^*}))$ ;
- $((x_s, V(x_s)) \in D(w_s) \text{ и } (x_{s^*}, CE(x_{s^*})) \notin D(w_{s^*})) \Rightarrow ((x_{s^*}, V(x_{s^*})) \notin D(w_{s^*}))$ ;
- $(R(w_s) \neq R(w_{s^*}) \text{ или } RO(w_s) \neq RO(w_{s^*})) \Rightarrow (RO(w_{s^*}) \subseteq R(w_{s^*}))$ ;
- $(CE(w_s) \neq CE(w_{s^*}) \text{ или } CD(w_s) \neq CD(w_{s^*})) \Rightarrow (CD(w_{s^*}) \geq CE(w_{s^*}))$ .

**Доказательство.** Доказательство осуществляется аналогично доказательству теоремы 3.1 (A1) модели Белла—Лападула.

**Определение 3.55.** Функция переходов  $T()$  безопасна в смысле БТБ, если выполнены условия теоремы 3.11 (БТБ-СВС).

**Определение 3.56.** Функция переходов  $T()$  безопасна тогда и только тогда, когда она безопасна в смыслах определений 3.48—3.55.

**Определение 3.57.** История  $\pi()$  системы  $\Sigma(T, s_0)$  безопасна, если все ее состояния и переходы безопасны.

**Определение 3.58.** Система  $\Sigma(T, s_0)$  безопасна, если все ее истории безопасны.

**Теорема 3.12.** Система  $\Sigma(T, s_0)$  безопасна, если начальное состояние  $s_0$  безопасно и функция переходов  $T()$  безопасна.

**Доказательство.** Доказательство следует из определений.

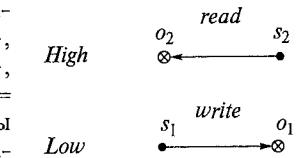
В заключение рассмотрения модели СВС следует отметить, что в ней не содержится описания механизмов администрирования. В частности, при описании функции переходов пропущены такие возможные действия в системе, как создание новых сущностей, присваивание им уровня конфиденциальности  $CE(e)$  и множества доступов  $AS(e)$ . Исходя из постулатов безопасности предполагается, что данные действия осуществляются корректно. Изменения значения множества доступов  $AS(e)$  для существующей сущности могут быть осуществлены с использованием доступов, определенных в самом значении  $AS(e)$  для данной сущности. Поэтому для задания правил изменения значений функции  $AS()$  не требуется изменения определений безопасности функции переходов  $T()$ .

### Контрольные вопросы

1. Каковы основные недостатки классической модели Белла—Лападула?

2. Как сформулировать теорему 3.4 (БТБ) для модели мандатной политики целостности информации Биба?

3. Постройте пример системы Белла—Лападула со следующими параметрами:  $S = \{s_1, s_2\}$ ,  $O = \{o_1, o_2\}$ ,  $R = \{\text{read}, \text{write}\}$ ,  $(L, \leq) = \{\text{Low}, \text{High}\}$ ,  $M$  — не используется,  $f_s(s_1) = f_o(o_1) = \text{Low}$ ,  $f_s(s_2) = f_o(o_2) = \text{High}$ . Рассмотрите возможные варианты действия системы в состоянии, описываемом графиком текущих доступов, по запросу субъекта  $s_2$  на доступ на запись в объект  $o_1$ .



4. Переформулируйте определения  $ss$ -свойства и  $*$ -свойства функции переходов  $T(s, q, (b, f)) = (b^*, f^*)$ , включив в них определение безопасности функции переходов в смысле администрирования.

5. Докажите теорему 3.11 (БТБ-СВС).

6. Каким образом десять неформальных свойств модели СВС реализуются в ее формальном описании?

7. Где в определениях безопасности модели СВС реализованы свойства безопасности ( $ss$ -свойство,  $*$ -свойство и  $ds$ -свойство) классической модели Белла—Лападула?

8. Рассмотрите пример использования определения потенциальной модификации булевой сущности по ссылке  $r$  с источником булевой сущностью по ссылке  $y$  при запросе  $(u, i, s)$ , реализующем функцию  $V(r_s) = V(r_s) \text{ and } V(y_s)$ .

# МОДЕЛИ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ

## 4.1. АВТОМАТНАЯ МОДЕЛЬ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ

В автоматной модели безопасности информационных потоков [2] система защиты представляется детерминированным автоматом, на вход которого поступает последовательность команд пользователей. Основными элементами автоматной модели безопасности информационных потоков являются:

$V$  — множество состояний системы;

$U$  — множество пользователей;

$M$  — множество матриц доступов;

$CC$  — множество команд пользователей, изменяющих матрицу доступов;

$VC$  — множество команд пользователей, изменяющих состояние;

$C = CC \cup VC$  — множество всех команд пользователей;

$Out$  — множество выходных значений;

$cndo: U \times C \times V \times M \rightarrow V \times M$  — функция переходов системы.

В зависимости от вида команды при переходе системы в последующее состояние или изменяется состояние системы, или вносятся изменения в матрицу доступов. При этом для каждого пользователя в матрице доступов определяются команды, которые ему разрешено выполнять. Таким образом, автоматная модель безопасности информационных потоков реализует дискреционное разграничение доступа.

Для каждого пользователя системы задается функция выходов, определяющая, что каждый из них «видит» на устройстве вывода в соответствующем состоянии системы:

$out: U \times V \rightarrow Out$  — функция выходов системы.

Таким образом, определен автомат, в котором:

$(v_0, m_0)$  — начальное состояние;

$V \times M$  — множество внутренних состояний автомата;

$U \times C$  — входной алфавит;

$Out$  — выходной алфавит;

$cndo()$  — функция переходов автомата;

$out()$  — функция выходов автомата.

Функция  $cndo()$  может быть по индукции определена для конечных последовательностей команд пользователей:

$cndo: (U \times C)^* \times V \times M \rightarrow V \times M$ .

Пусть  $w \in (U \times C)^*$  — последовательность команд пользователей;  $s \in U$  — пользователь;  $G \subset U$  — подмножество множества пользователей;  $A \subset C$  — подмножество множества команд. Используем следующие обозначения:

$[[w]]$  — последовательность состояний, полученная из исходного состояния в результате выполнения последовательности команд пользователей  $w$ ;

$[[w]]_s$  — последовательность выходов, наблюдаемых пользователем  $s$  при реализации последовательности состояний  $[[w]]$ ;

$P_G(w)$  — подпоследовательность последовательности  $w$ , полученная из нее удалением всех пар  $(s, c)$ , где  $s \in G$ ;

$P_A(w)$  — подпоследовательность последовательности  $w$ , полученная из нее удалением всех пар  $(s, c)$ , где  $c \in A$ ;

$P_{GA}(w)$  — подпоследовательность последовательности  $w$ , полученная из нее удалением всех пар  $(s, c)$ , где  $s \in G$  и  $c \in A$ .

**Определение 4.1.** Пусть  $G, G' \subset U$ .  $G$  информационно не влияет на  $G'$  (обозначим через  $G :| G'$ ), если для каждой  $w \in (U \times C)^*$  и для  $s \in G'$  справедливо равенство  $[[w]]_s = [[P_G(w)]]_s$ .

**Определение 4.2.** Пусть  $A \subset C, G' \subset U$ .  $A$  информационно не влияет на  $G'$  (обозначим через  $A :| G'$ ), если для каждой  $w \in (U \times C)^*$  и для  $s \in G'$  справедливо равенство  $[[w]]_s = [[P_A(w)]]_s$ .

**Определение 4.3.** Пусть  $A \subset C, G, G' \subset U$ .  $G$  и  $A$  информационно не влияют на  $G'$  (обозначим через  $G, A :| G'$ ), если для каждой  $w \in (U \times C)^*$  и для  $s \in G'$  справедливо равенство  $[[w]]_s = [[P_{GA}(w)]]_s$ .

**Пример 4.1.** Пусть  $f$  — файл,  $s \in U$  — пользователь,  $A = \{create(f), write(f), modify(f), delete(f)\}$  — набор команд. Тогда  $\{s\}, A :| \{s\}$  означает, что пользователь  $s$  может осуществлять только чтение из файла  $f$ .

**Определение 4.4.** Политика безопасности в автоматной модели безопасности информационных потоков — это набор требований информационного невлияния.

**Пример 4.2.** Мандатную политику безопасности в автоматной модели безопасности информационных потоков, которая состоит в том, что пользователи с большим уровнем доступа не должны информационно влиять на пользователей с меньшим уровнем доступа, можно выразить следующим образом.

Пусть

$(L, \leq)$  — шкала уровней доступа к информации;

$level: U \rightarrow L$  — функция уровней доступа пользователей;

$U[-\infty, x] = \{s \in U: level(s) \leq x\}$  — множество пользователей с уровнем доступа, не большим  $x$ ;

$U[x, +\infty] = \{s \in U : \text{level}(s) \geq x\}$  — множество пользователей с уровнем доступа, не меньшим  $x$ .

Тогда мандатная политика безопасности есть совокупность требований информационного невлияния:

для  $x, x' \in L$ ,  $x < x'$  истинно  $U[x', +\infty] \vdash U[-\infty, x]$ .

**Определение 4.5.** Пусть  $G \subset U$ . Пользователи из  $G$  невидимы для остальных пользователей, если  $G \vdash (U \setminus G)$ . Пользователи из  $G$  изолированы от остальных пользователей, если  $G \vdash (U \setminus G)$  и  $(U \setminus G) \vdash G$ .

С помощью требований информационного невлияния могут быть определены различные виды политик безопасности, определяющие не только правила разграничения доступа пользователей системы, но и конфигурацию, порядок взаимодействия между собой распределенных компонент компьютерной системы.

## 4.2. ПРОГРАММНАЯ МОДЕЛЬ КОНТРОЛЯ ИНФОРМАЦИОННЫХ ПОТОКОВ

### 4.2.1. Основные элементы модели

В программной модели контроля информационных потоков [11] предлагается механизм построения системы защиты, реализующей дискреционную политику безопасности для командных интерпретаторов. Основными элементами модели являются:

$Q: D_1 \times \dots \times D_n \rightarrow E$  — программа, отображающая множество значений входных параметров  $D_1 \times \dots \times D_n$  в множество выходных значений  $E$ ;

$M: D_1 \times \dots \times D_n \rightarrow E \cup \{\text{access\_denied}\}$  — механизм защиты, где  $\text{access\_denied}$  — сообщение о нарушении защиты. При этом для  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  выполняется одно из условий:

- $M(d_1, \dots, d_n) = Q(d_1, \dots, d_n)$ ;
- $M(d_1, \dots, d_n) = \text{access\_denied}$ ;

$I: D_1 \times \dots \times D_n \rightarrow \Omega$  — политика безопасности, где  $\Omega$  — множество разрешений. Политика безопасности в программной модели контроля информационных потоков определяется через множество значений входных параметров, от которых разрешено вычисление выходного значения программы. Для остальных (запрещенных) значений входных параметров должно выдаваться сообщение о нарушении защиты.

**Определение 4.6.** Пусть  $Q: D_1 \times \dots \times D_n \rightarrow E$  — программа. Политика безопасности «допустить»  $(i_1, \dots, i_k)$ , где  $1 \leq i_1 < \dots < i_k \leq n$ , для входных параметров  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  разрешает выдать результат работы программы  $Q(d_1, \dots, d_n)$ , если он зависит только от параметров  $d_{i_1}, \dots, d_{i_k}$ .

**Определение 4.7.** Пусть  $Q: D_1 \times \dots \times D_n \rightarrow E$  — программа;  $I: D_1 \times \dots \times D_n \rightarrow \Omega$  — политика безопасности. Механизм защиты  $M: D_1 \times \dots \times D_n \rightarrow E \cup \{\text{access\_denied}\}$  эффективен по отношению к политике безопасности  $I$ , если существует функция  $\mu: \Omega \rightarrow E \cup \{\text{access\_denied}\}$  такая, что для каждого  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  справедливо равенство:

$$M(d_1, \dots, d_n) = \mu(I(d_1, \dots, d_n)).$$

**Замечание 4.1.** Для любой программы  $Q: D_1 \times \dots \times D_n \rightarrow E$  и политики безопасности  $I: D_1 \times \dots \times D_n \rightarrow \Omega$  существует эффективный механизм защиты, например:  $M: D_1 \times \dots \times D_n \rightarrow \{\text{access\_denied}\}$ .

**Определение 4.8.** Пусть  $M_1$  и  $M_2$  — эффективные механизмы защиты для программы  $Q$  по отношению к политике безопасности  $I$ .

$M_1 \leq M_2$ , если для  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  выполняется условие

$$(M_1(d_1, \dots, d_n) = Q(d_1, \dots, d_n)) \Rightarrow (M_2(d_1, \dots, d_n) = Q(d_1, \dots, d_n)).$$

$M_1 < M_2$ , если  $M_1 \leq M_2$  и найдется набор параметров  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$ , для которого выполняется условие

$$M_2(d_1, \dots, d_n) = Q(d_1, \dots, d_n) \text{ и } M_1(d_1, \dots, d_n) = \text{access\_denied}.$$

**Замечание 4.2.** На множестве эффективных механизмов защиты для программы  $Q$  по отношению к политике безопасности  $I$  отношения «≤» и «<» являются соответственно отношениями частичного и строгого порядка.

**Определение 4.9.** Пусть  $M_1$  и  $M_2$  — эффективные механизмы защиты для программы  $Q$  по отношению к политике безопасности  $I$ . Определим  $M = M_1 \cup M_2$  — механизмы защиты для программы  $Q$ , где для  $(d_1, \dots, d_n) \in D_1 \times \dots \times D_n$  выполняется условие  $M(d_1, \dots, d_n) = Q(d_1, \dots, d_n)$ , когда или  $M_1(d_1, \dots, d_n) = Q(d_1, \dots, d_n)$ , или  $M_2(d_1, \dots, d_n) = Q(d_1, \dots, d_n)$ . В противном случае  $M(d_1, \dots, d_n) = \text{access\_denied}$ .

**Утверждение 4.1.** Пусть  $M_1$  и  $M_2$  — эффективные механизмы защиты для программы  $Q$  по отношению к политике безопасности  $I$ , тогда  $M = M_1 \cup M_2$  — эффективный механизм защиты для программы  $Q$  по отношению к политике безопасности  $I$ . При этом выполняются соотношения:

$$M_1 \cup M_2 = M_2 \cup M_1;$$

$$M_1 \leq M_1 \cup M_2;$$

$$M_2 \leq M_1 \cup M_2.$$

**Доказательство.** Доказательство следует из определений 4.7—4.9.

**Замечание 4.3.** Утверждение 4.1 верно для  $M = M_1 \cup \dots \cup M_k$ , где  $M_1, \dots, M_k$  — эффективные механизмы защиты для программы  $Q$  по отношению к политике безопасности  $I$ .

**Замечание 4.4.** Так как  $Q$  — программа, выполняющаяся в некоторой компьютерной системе, то без ограничения общности можно считать, что  $D_1, \dots, D_n$  — конечные множества. Тогда если существует  $M$  — эффективный механизм защиты для программы  $Q$  по отношению к безопасности политике  $I$ , то существует  $M^*$  — максимальный эффективный механизм защиты для программы  $Q$  по отношению к политике безопасности  $I$ . При этом  $M^*$  равен объединению всех эффективных механизмов защиты для программы  $Q$  по отношению к политике безопасности  $I$ , которых конечное число.

## 4.2.2. Контролирующий механизм защиты

**Определение 4.10.** Пусть каждая программа  $Q: D_1 \times \dots \times D_n \rightarrow E$  может быть представлена схемой из четырех видов элементов.

1.  $[start]$  — начальный элемент.
2.  $[r \leftarrow f(w_1, \dots, w_a)]$  — элемент присваивания.
3.  $[if(g(w_1, \dots, w_b) = true) \text{ then } [...] \text{ else } [...]]$  — элемент условия.
4.  $[stop]$  — конечный элемент.

Пусть

$x_1, \dots, x_n$  — входные параметры программы;  
 $y_1, \dots, y_m$  — внутренние переменные программы;  
 $z$  — выходное значение программы.

Определим контролирующий механизм защиты для программы  $Q$  по отношению к политике «допустить»( $i_1, \dots, i_k$ ). Пусть  $v_1^*, \dots, v_{n+m}^*, z^*, c^*$  — множества символьных переменных. Заменим в зависимости от вида каждый элемент программы  $Q$  на следующие последовательности элементов.

1. Начальный элемент.

$[start];$   
 $[v_1^* \leftarrow \{x_1\}];$

$\dots$   
 $[v_n^* \leftarrow \{x_n\}];$   
 $[v_{n+1}^* \leftarrow \emptyset];$

$\dots$   
 $[v_{n+m}^* \leftarrow \emptyset];$   
 $[z^* \leftarrow \emptyset];$   
 $[c^* \leftarrow \emptyset].$

2. Элемент присваивания.

$[v_{j_1}^* \leftarrow v_{j_1}^* \cup \dots \cup v_{j_a}^*];$   
 $[v_{j_1}^* \leftarrow f(v_{j_1}, \dots, v_{j_a})].$

3. Элемент условия.

$[c^* \leftarrow c^* \cup v_{j_1}^* \cup \dots \cup v_{j_b}^*];$   
 $[if(g(v_{j_1}, \dots, v_{j_b}) = true) \text{ then } [...] \text{ else } [...]].$

4. Конечный элемент.

$[c^* \leftarrow c^* \cup z^*];$   
 $[if(c^* \subset \{i_1, \dots, i_k\}) \text{ then } [stop] \text{ else } [access\_denied]].$

**Теорема 4.1.** Контролирующий механизм защиты для программы  $Q: D_1 \times \dots \times D_n \rightarrow E$  по отношению к политике «допустить»( $i_1, \dots, i_k$ ) является эффективным.

**Доказательство.** Контролирующий механизм защиты выполняет все вычисления программы  $Q$ . При этом если выходное значение  $z$  работы программы  $Q$  зависело от входного параметра  $x_i$ , то, очевидно,  $\{x_i\} \subset c^*$ . Следовательно, если политика безопасности требует запрета выдачи выходного значения программы, то контролирующий механизм защиты выдает  $access\_denied$ . Если контролирующий механизм защиты выдает выходное значение, отличное от  $access\_denied$ , то оно совпадает с выходным значением программы  $Q$ . Теорема доказана.

**Замечание 4.5.** В общем случае контролирующий механизм защиты для программы  $Q: D_1 \times \dots \times D_n \rightarrow E$  по отношению к политике «допустить»( $i_1, \dots, i_k$ ) не является максимальным.

**Пример 4.3.** Для следующей программы  $Q: D_1 \times D_2 \rightarrow E$  и политики безопасности «допустить»( $2$ ) контролирующий механизм защиты не будет являться максимальным.

```
[start];
[if ((x_1 = 1) = true) then [v ← 1] else [v ← 2]];
[z ← 1];
[stop].
```

По окончании работы контролирующего механизма защиты справедливо равенство  $c^* = \{x_1\}$ , и, следовательно, его результатом будет  $access\_denied$ . В то же время очевидно, что результат работы программы от переменной  $x_1$  не зависит.

## 4.3. ВЕРОЯТНОСТНАЯ МОДЕЛЬ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ

### 4.3.1. Схема компьютерной системы

В рассматриваемой вероятностной модели безопасности информационных потоков [18] анализируются компьютерные системы, в которых реализована мандатная политика безопасности. В модели предполагается, что все объекты (в том числе и субъекты) компьютерной системы объединены по трем группам: объекты  $\Sigma$ , реализующие систему защиты; объекты  $H$ , обрабатывающие информацию высокого уровня конфиденциальности; объекты  $L$ , обрабатывающие информацию низкого уровня конфиденциальности. Все информационные потоки между  $H$  и  $L$  проходят через систему защиты  $\Sigma$  (рис. 4.1).

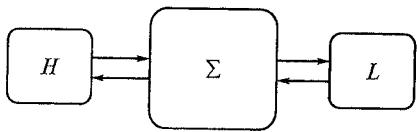


Рис. 4.1. Схема компьютерной системы

ки от  $H$  к  $L$  запрещены. Однако в большинстве реальных систем реализовать данное жесткое требование не представляется возможным. В модели рассматриваются ряд подходов к определению возможных информационных потоков между  $H$  и  $L$ , основанных на понятиях информационной невыводимости и информационного невлияния.

Пусть на множестве значений объектов системы задано вероятностное распределение, т. е.  $H$  и  $L$  являются случайными величинами.

Согласно требованиям мандатной политики безопасности, любые информационные потоки от  $H$  к  $L$  запрещены. Однако в большинстве реальных систем реализовать данное жесткое требование не представляется возможным. В модели рассматриваются ряд подходов к определению возможных информационных потоков между  $H$  и  $L$ , основанных на понятиях информационной невыводимости и информационного невлияния.

### 4.3.2. Информационная невыводимость

**Определение 4.11.** Компьютерная система соответствует требованиям информационной невыводимости, если для  $p(H) > 0$ ,  $p(L) > 0$  справедливо неравенство  $p(H|L) > 0$ .

Так как при  $p(H) > 0$ ,  $p(L) > 0$  справедливо равенство

$$p(L|H) = p(H, L)/p(H) = p(H|L)p(L)/p(H),$$

то в условиях определения 4.1 из истинности неравенства  $p(H|L) > 0$  следует истинность неравенства  $p(L|H) > 0$ , что предполагает отсутствие информационных потоков от низкоуровневых объектов к высокоуровневым. Таким образом, требования информационной невыводимости являются более строгими, чем требования безопасности классической модели Белла—ЛаПадула, и фактически предполагают изоляцию друг от друга высокоуровневых и низкоуровневых объектов системы, что является чрезмерно жестким требованием.

### 4.3.3. Информационное невлияние

В традиционной модели информационного невлияния требуется, чтобы низкоуровневая информация была независима от высокоуровневой, т. е. чтобы выполнялось условие определения 4.12.

**Определение 4.12.** Компьютерная система соответствует требованиям информационного невлияния (без учета времени), если для  $p(H) > 0$ ,  $p(L) > 0$  справедливо равенство

$$p(L|H) = p(L).$$

**Замечание 4.6.** При  $p(H) > 0$ ,  $p(L) > 0$  условие определения 4.12 равносильно условию

$$p(H|L) = p(H).$$

Условие определения 4.12 также является жестким. Однако если ввести параметр времени, то возможно данное условие сделать в большей степени применимым для использования в реальных системах. Конечный вид условия информационного невлияния получим в результате последовательности рассуждений.

Если  $L_t$  описывает состояния всех низкоуровневых объектов, а  $H_t$  — всех высокоуровневых объектов в момент времени  $t = 0, 1, 2, \dots$ , то нет необходимости требовать выполнения условия

$$p(H_t|L_{t-1}) = p(H_t),$$

т. е. текущее значение низкоуровневых объектов может содержать информацию о последующих значениях высокоуровневых объектов.

Например, если низкоуровневым является некий неконфиденциальный файл, обрабатываемый пользователем с низким уровнем доступа, а высокоуровневым объектом является журнал аудита, то значение файла и операции, совершаемые над ним пользователем в момент времени  $t$ , могут отображаться в журнале аудита в момент времени  $t+1$ , о чём «знает» низкоуровневый пользователь.

Учитывая тот факт, что состояние системы влияет на последующие состояния только через информацию, хранимую в объектах системы, казалось бы, необходимо потребовать, чтобы текущее значение низкоуровневых объектов не зависело от значения высокоуровневых объектов в предыдущие моменты работы системы, т. е. чтобы выполнялось условие

$$p(L_t|H_{t-1}) = p(L_t), \text{ или } p(H_{t-1}|L_t) = p(H_{t-1}), \\ \text{для } t = 1, 2, \dots$$

Однако следует отметить, что данный вариант определения соотношения  $L_t$  и  $H_{t-1}$  является слишком строгим, так как предполагает независимость  $L_t$  и  $H_{t-1}$ . В то же время значение высокоуровневых объектов на текущем шаге часто оказывает влияние на значение низкоуровневых объектов на последующих шагах работы системы. Например, рассмотрим работу монитора ссылок компьютерной системы.

Монитор ссылок в реальных системах защиты является высокоуровневым субъектом, принимающим решения по запросам на доступ к объектам, полученным от других субъектов системы. Очевидно, что такое решение, полученное низкоуровневым субъектом в момент времени  $t$  работы системы, содержит информацию о значении высокоуровневого монитора ссылок в предыдущий момент времени  $t-1$ .

Более целесообразным представляется подход, обеспечивающий невозможность накопления низкоуровневыми объектами новой информации о значениях высокоуровневых объектов. Необходимо потребовать, чтобы знание значений  $L_{t-1}$  и  $L_t$ , не давало новой информации о  $H_{t-1}$ , т.е. должно выполняться условие

$$p(L_t | H_{t-1}, L_{t-1}) = p(L_t | L_{t-1}) \text{ для } t = 1, 2, \dots,$$

или

$$p(H_{t-1} | L_t, L_{t-1}) = p(H_{t-1} | L_{t-1}) \text{ для } t = 1, 2, \dots$$

Таким образом, запрещается обратный информационный поток из  $L_t$  в  $H_{t-1}$ , но не запрещается поток из  $L_t$  в  $H_{t+1}$ . Кроме того, следует отметить, что, решая проблемы, обозначенные в рассмотренных выше примерах, последнее правило блокирует возникновение неблагоприятных информационных потоков по времени.

Дадим определение информационного невлияния с учетом времени.

**Определение 4.13.** Компьютерная система соответствует требованиям информационного невлияния (с учетом времени), если для  $p(L_s) > 0$ ,  $p(L_t) > 0$ ,  $p(H_s) > 0$  выполняется условие

$$p(L_t | H_s, L_s) = p(L_t | L_s),$$

где  $s, t = 0, 1, 2, \dots$  и  $s < t$ .

**Пример 4.4.** Согласно описанию автоматной модели безопасности информационных потоков, система защиты представляется детерминированным автоматом, на вход которого поступает последовательность команд пользователей. Для каждого пользователя системы задана функция выходов, определяющая, что каждый из них «видит» на устройстве вывода в соответствующем состоянии системы. Если всех пользователей системы разделить на две группы (низкоуровневые и высокоуровневые), то требование информационного невлияния автоматной модели является требованием независимости низкоуровневого вывода системы от ее высокоуровневого ввода.

Рассмотрим систему автоматной модели как совокупность четырех объектов: высокоуровневых и низкоуровневых портов ввода/вывода  $high-in$ ,  $high-out$ ,  $low-in$ ,  $low-out$ . При этом системный вывод полностью определяется системным вводом. Вероятностные отношения исключаются.

Однако если интерпретировать детерминированность автомата, используя события с вероятностью  $\{0, 1\}$ , то информационное невлияние можно определить следующим образом.

**Определение 4.14.** Пусть система автоматной модели безопасности информационных потоков, представленная последовательностью событий с вероятностью  $\{0, 1\}$ :  $high-in$ ,  $high-out$ ,  $low-in$ ,

$low-out$ , для  $t = 0, 1, 2, \dots$  Тогда система автоматной модели соответствует требованиям информационного невлияния, если выполняется условие

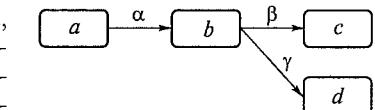
$$p(low-out_t | high-in_s, low-in_s) = p(low-out_t | low-in_s),$$

где  $s, t = 0, 1, 2, \dots$  и  $s < t$ .

### Контрольные вопросы

1. Какой вид политики разграничения доступа используется в качестве основы автоматной модели безопасности информационных потоков?

2. Пусть  $a, b, c, d$  — компьютеры;  $\alpha, \beta, \gamma$  — сетевые информационные каналы, определяемые как совокупность команд сетевых интерфейсов. Опишите компьютерную систему, представленную на схеме, с использованием семи требований информационного невлияния автоматной модели безопасности информационных потоков.



3. Почему программная модель контроля информационных потоков может быть наиболее эффективно использована для систем защиты, реализованных в командных интерпретаторах?

4. Постройте контролирующие механизмы защиты для следующих программ.

а)

$Program1(x_1, x_2, x_3)$

{

$a = x_1 - x_2;$

$if (x_3 > 0) \quad y = a + x_1;$

$else \quad y = a + x_2;$

$y = y \cdot x_1;$

}

Являются ли они максимальными?

б)

$Program2(x_1, x_2, x_3)$

{

$a = x_2 + x_3;$

$if (x_1 = 0) \quad a = a + x_3;$

$else \quad a = a - x_3;$

$y = a - x_2;$

}

5. Почему использование определения требований информационного невлияния (с учетом времени) позволяет обеспечить возможность функционирования в компьютерной системе монитора ссылок и системы аудита?

# МОДЕЛИ РОЛЕВОГО РАЗГРАНИЧЕНИЯ ДОСТУПА

## 5.1. ПОНЯТИЕ РОЛЕВОГО РАЗГРАНИЧЕНИЯ ДОСТУПА

Ролевое разграничение доступа (РРД) представляет собой развитие политики дискреционного разграничения доступа; при этом права доступа субъектов системы на объекты группируются с учетом специфики их применения, образуя роли.

РРД является составляющей многих современных компьютерных систем. Как правило, РРД применяется в системах защиты СУБД, отдельные элементы РРД реализуются в сетевых ОС.

Задание ролей позволяет определить более четкие и понятные для пользователей компьютерной системы правила разграничения доступа. При этом РРД наиболее эффективно используется в компьютерных системах, для пользователей которых четко определен круг их должностных полномочий и обязанностей.

Роль является совокупностью прав доступа на объекты компьютерной системы. Однако РРД не является частным случаем дискреционного разграничения доступа, так как правила РРД определяют порядок предоставления прав доступа субъектам (пользователям) компьютерной системы в зависимости от сессии его работы и от имеющихся или отсутствующих у него ролей в каждый момент времени, что является характерным для систем мандатного разграничения доступа. В то же время правила РРД являются более гибкими, чем правила мандатного разграничения доступа, построенные на основе жестко определенной решетки (шкалы) ценности информации.

Для анализа и изучения свойств систем РРД используются математические модели РРД. В основе всех математических моделей РРД лежит базовая модель РРД.

## 5.2. БАЗОВАЯ МОДЕЛЬ РРД

Базовая модель РРД [3, 20] определяет самые общие принципы построения моделей РРД.

Основными элементами базовой модели РРД являются:

$U$  — множество пользователей;

$R$  — множество ролей;

$P$  — множество прав доступа на объекты компьютерной системы;

$S$  — множество сессий пользователей;

$PA: R \rightarrow 2^P$  — функция, определяющая для каждой роли множество прав доступа; при этом для каждого  $p \in P$  существует  $r \in R$  такая, что  $p \in PA(r)$ ;

$UA: U \rightarrow 2^R$  — функция, определяющая для каждого пользователя множество ролей, на которые он может быть авторизован;

$user: S \rightarrow U$  — функция, определяющая для каждой сессии пользователя, от имени которого она активизирована;

$roles: S \rightarrow 2^R$  — функция, определяющая для пользователя множество ролей, на которые он авторизован в данной сессии; при этом в каждый момент времени для каждого  $s \in S$  выполняется условие  $roles(s) \subseteq UA(user(s))$ .

Отметим, что могут существовать роли, на которые не авторизован ни один пользователь.

В базовой модели РРД предполагается, что множества  $U, R, P$  и функции  $PA, UA$  не изменяются с течением времени. Множество ролей, на которые авторизуется пользователь в течение одной сессии, модифицируется самим пользователем. В базовой модели РРД отсутствуют механизмы, позволяющие одной сессии активизировать другую сессию. Все сессии активизируются пользователем.

Для обеспечения возможности большего соответствия реальным компьютерным системам, каждый пользователь которых занимает определенное положение в служебной иерархии, на множестве ролей реализуется иерархическая структура.

**Определение 5.1.** Иерархией ролей в базовой модели РРД называется заданное на множестве ролей  $R$  отношение частичного порядка «≤» (отношение «≤» обладает свойствами рефлексивности, антисимметричности и транзитивности). При этом выполняется условие

для  $u \in U$ , если  $r, r' \in R, r \in UA(u)$  и  $r' \leq r$ , то  $r' \in UA(u)$ .

Таким образом, наряду с данной ролью пользователь должен быть авторизован на все роли, в иерархии ее меньшие.

Отношение частичного порядка на множестве ролей не обязательно задает на нем решетку.

Другим важным механизмом базовой модели РРД являются ограничения, накладываемые на множества ролей, на которые может быть авторизован пользователь или на которые он авторизуется в течение одной сессии. Данный механизм также необходим для широкого использования РРД, так как обеспечивает большее соответствие используемым в компьютерных системах технологиям обработки информации.

Дадим определения ряда распространенных видов ограничений.

**Определение 5.2.** В базовой модели РРД заданы ограничения статического взаимного исключения ролей или прав доступа, если выполняются условия:

$$\begin{aligned} R &= R_1 \cup \dots \cup R_n, \text{ где } R_i \cap R_j = \emptyset, \text{ для } 1 \leq i < j \leq n; \\ |UA(u) \cap R_i| &\leq 1 \text{ для } u \in U, i \in 1, 2, \dots, n; \\ P &= P_1 \cup \dots \cup P_m, \text{ где } P_i \cap P_j = \emptyset, \text{ для } 1 \leq i < j \leq m; \\ |PA(r) \cap P_i| &\leq 1 \text{ для } p \in U, i \in 1, 2, \dots, m. \end{aligned}$$

Множество ролей и множество прав доступа разделяются на непересекающиеся подмножества. При этом каждый пользователь может обладать не более чем одной ролью из каждого подмножества ролей, а каждая роль — не более чем одним правом доступа из каждого подмножества прав доступа.

**Определение 5.3.** В базовой модели РРД задано ограничение динамического взаимного исключения ролей, если выполняются условия:

$$\begin{aligned} R &= R_1 \cup \dots \cup R_n, \text{ где } R_i \cap R_j = \emptyset, \text{ для } 1 \leq i < j \leq n; \\ |\text{roles}(s) \cap R_i| &\leq 1 \text{ для } s \in S, i \in 1, 2, \dots, n. \end{aligned}$$

Множество ролей разделяется на непересекающиеся подмножества. При этом в каждой сессии пользователь может обладать не более чем одной ролью из каждого подмножества ролей.

**Определение 5.4.** В базовой модели РРД заданы статические количественные ограничения на обладание ролью или правом доступа, если определены две функции:

$$\begin{aligned} \alpha: R &\rightarrow N_0; \\ \beta: P &\rightarrow N_0, \end{aligned}$$

где  $N_0$  — множество натуральных чисел с нулем, и выполняются условия:

$$\begin{aligned} |UA^{-1}(r)| &\leq \alpha(r) \text{ для } r \in R; \\ |PA^{-1}(p)| &\leq \beta(p) \text{ для } p \in P. \end{aligned}$$

Для каждой роли устанавливается максимальное число пользователей, которые могут быть на нее авторизованы, а для каждого права доступа устанавливается максимальное число ролей, которые могут им обладать.

**Определение 5.5.** В базовой модели РРД задано динамическое количественное ограничение на обладание ролью, если определена функция

$$\gamma: R \rightarrow N_0$$

и выполняется условие

$$|\text{roles}^{-1}(r)| \leq \gamma(r) \text{ для } r \in R.$$

Для роли устанавливается максимальное число сессий пользователей, которые могут одновременно быть на нее авторизованы.

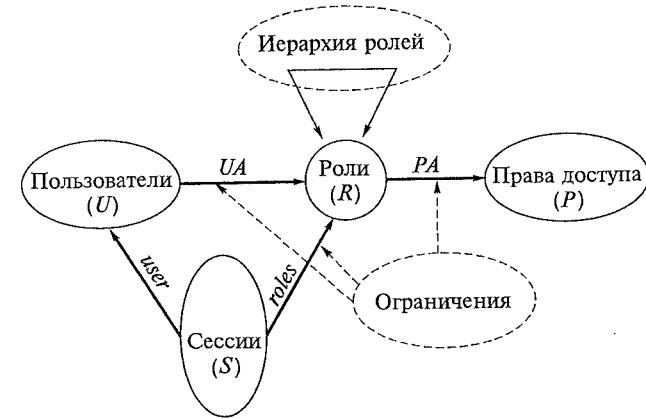


Рис. 5.1. Структура элементов базовой модели РРД

**Определение 5.6.** В базовой модели РРД заданы статические ограничения необходимого обладания ролью или правом доступа, если определены две функции:

$$\begin{aligned} \alpha: R &\rightarrow 2^R, \\ \beta: P &\rightarrow 2^P, \end{aligned}$$

и выполняются условия:

$$\begin{aligned} \text{для } u \in U, \text{ если } r, r' \in R, r \in UA(u) \text{ и } r' \in \alpha(r), \text{ то } r' \in UA(u); \\ \text{для } r \in R, \text{ если } p, p' \in P, p \in PA(r) \text{ и } p' \in \beta(p), \text{ то } p' \in PA(r). \end{aligned}$$

Для каждой роли для того чтобы на нее мог быть авторизован пользователь, могут быть определены роли, на которые пользователь также должен быть авторизован. Для каждого права доступа для того чтобы им обладала роль, могут быть определены права доступа, которыми данная роль также должна обладать.

**Определение 5.7.** В базовой модели РРД задано динамическое ограничение необходимого обладания ролью, если определена функция

$$\gamma: R \rightarrow 2^R$$

и выполняется условие

$$\text{для } s \in S, \text{ если } r, r' \in R, r \in \text{roles}(s) \text{ и } r' \in \gamma(r), \text{ то } r' \in \text{roles}(s).$$

Для каждой роли для того чтобы на нее мог быть авторизован пользователь в некоторой сессии, могут быть определены роли, на которые пользователь также должен быть авторизован в данной сессии.

Общая структура элементов базовой модели РРД имеет вид, представленный на рис. 5.1.

### 5.3.1. Основные положения

Как уже отмечалось, в базовой модели РРД предполагается, что множества  $U$ ,  $R$ ,  $P$  и функции  $PA$ ,  $UA$  не изменяются с течением времени или существует единственная роль — «офицер безопасности», которая предоставляет возможность изменять эти множества и функции. В реальных компьютерных системах, в которых одновременно могут работать сотни и тысячи пользователей, а структура ролей и прав доступа может быть очень сложной, проблема администрирования является чрезвычайно важной задачей. Для решения этой задачи рассматривается построенная на основе базовой модели РРД модель администрирования РРД.

В дополнение к используемым элементам базовой модели РРД в модели администрирования РРД рассматриваются следующие элементы:

$AR$  — множество административных ролей ( $AR \cap R = \emptyset$ );

$AP$  — множество административных прав доступа ( $AP \cap P = \emptyset$ );

$APA$ :  $AR \rightarrow 2^P$  — функция, определяющая для каждой административной роли множество административных прав доступа; при этом для каждого  $p \in AP$  существует  $r \in AR$  такая, что  $p \in APA(r)$ ;

$AUA$ :  $U \rightarrow 2^{AR}$  — функция, определяющая для каждого пользователя множество административных ролей, на которые он может быть авторизован.

Кроме того, переопределяется следующая функция:

$roles$ :  $S \rightarrow 2^R \cup 2^{AR}$  — функция, определяющая для пользователя множество ролей, на которые он авторизован в данной сессии; при этом в каждый момент времени для каждого  $s \in S$  выполняется условие  $roles(s) \subseteq UA(user(s)) \cup AUA(user(s))$ .

Как и в базовой модели РРД, в модели администрирования РРД реализуются иерархия административных ролей и механизмы ограничений.

**Определение 5.8.** Иерархией ролей в модели администрирования РРД называется заданное на множестве ролей  $AR$  отношение частичного порядка  $\leq$ . При этом выполняется условие

для  $u \in U$ , если  $r, r' \in AR$ ,  $r \in AUA(u)$  и  $r' \leq r$ , то  $r' \in AUA(u)$ .

Общая структура элементов модели администрирования РРД имеет вид, представленный на рис. 5.2.

Административные роли могут быть разделены на три группы по своему назначению:

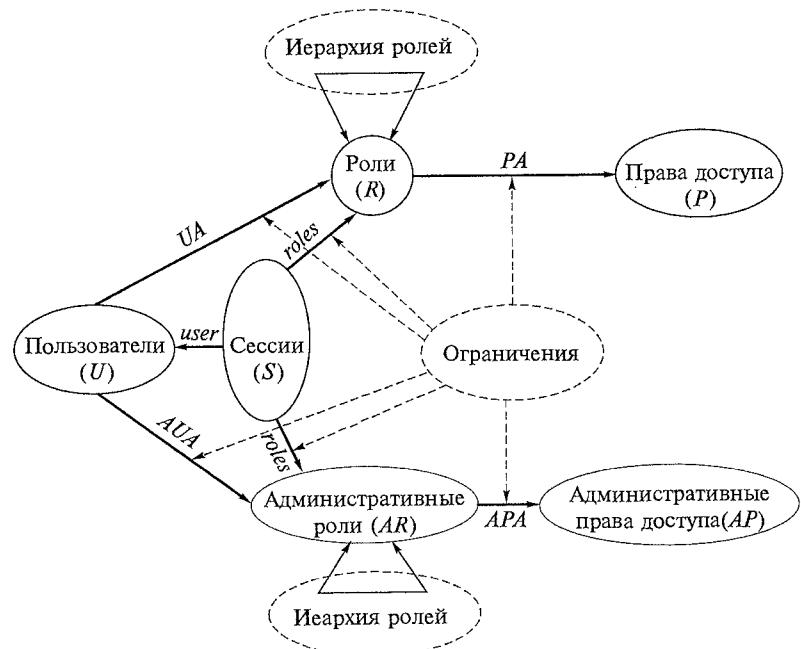


Рис. 5.2. Структура элементов модели администрирования РРД

- администрирование множеств авторизованных ролей пользователей;
  - администрирование множеств прав доступа, которыми обладают роли;
  - администрирование иерархии ролей.

Как правило, каждой административной роли назначают подмножество иерархии ролей, параметры которых данная административная роль позволяет изменять. Рассмотрим каждую из групп административных ролей подробнее.

### 5.3.2. Администрирование множеств авторизованных ролей пользователей

При администрировании множеств авторизованных ролей пользователей изменяются значения функции  $UA()$ . Для изменения значений функции  $UA()$  определяются специальные административные роли из множества  $AR$ .

Для администрирования множеств авторизованных ролей пользователей необходимо определять:

- для каждой административной роли множество ролей, множества авторизованных пользователей которых она позволяет изменять;

- для каждой роли предварительное условие, которому должны соответствовать пользователи, прежде чем они будут включены в множество ее авторизованных пользователей.

**Пример 5.1.** Пусть задана иерархия ролей (рис. 5.3, а) и иерархия административных ролей (рис. 5.3, б).

Минимальная роль в иерархии — служащий (*E*). Иерархия ролей разработчиков проектов имеет максимальную роль — директор (*DIR*), минимальную роль — инженер (*ED*). В управлении выполняются работы по двум проектам. В каждом проекте опреде-

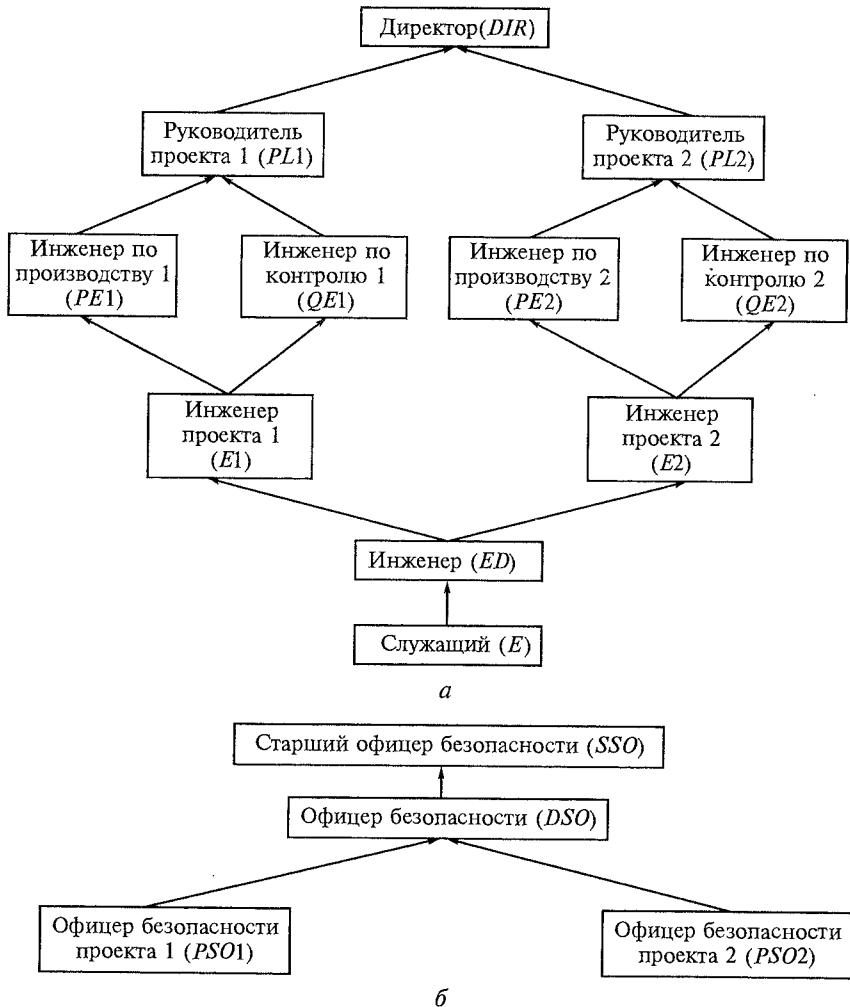


Рис. 5.3. Пример иерархии ролей (а) и иерархии административных ролей (б)

лены максимальная роль — руководитель проекта (*PL1*, *PL2* соответственно), минимальная роль — инженер проекта (*E1*, *E2* соответственно) и не сравнимые между собой роли — инженер по производству (*PE1*, *PE2* соответственно) и инженер по контролю (*QE1*, *QE2* соответственно). Иерархия административных ролей состоит из четырех ролей с максимальной ролью — старший офицер безопасности (*SSO*).

В рассматриваемом примере административная роль *PSO1* позволяет включать в множества авторизованных ролей пользователя роли *PE1*, *QE1*, *E1*. При этом для того чтобы любая из перечисленных ролей могла быть включена в множество авторизованных ролей пользователя, он уже должен обладать ролью *ED*.

Для роли  $x \in R$  и пользователя  $u \in U$  обозначим:

$x(u)$  — булева функция такая, что  $x(u) = \text{true}$  тогда и только тогда, когда  $\{x' \in R, \text{ где } x \leq x'\} \cap UA(u) \neq \emptyset$ .

**Определение 5.9.** Предварительным условием для роли  $r \in R$  называется булева функция  $c_r(u) = c_r(x_1(u), \dots, x_k(u))$ , где  $u \in U$ ,  $x_1, \dots, x_k \in R$  и  $c_r(y_1, \dots, y_k)$  — булева функция от  $k$  булевых переменных. При этом роль  $r \in R$  может быть включена в множество авторизованных ролей пользователя  $u \in U$ , если  $c_r(u) = \text{true}$ .

Обозначим через *CR* все предварительные условия для ролей из *R*.

**Определение 5.10.** Для администрирования множеств авторизованных ролей пользователей на множестве административных ролей задаются следующие функции:

- *can-assign*:  $AR \rightarrow CR \times 2^R$  — функция, определяющая для каждой административной роли множество ролей, которые могут быть включены в множество авторизованных ролей пользователя с использованием данной административной роли при выполнении заданных предварительных условий;

- *can-revoke*:  $AR \rightarrow 2^R$  — функция, определяющая для каждой административной роли множество ролей, которые могут быть исключены из множества авторизованных ролей пользователя с использованием данной административной роли.

Как правило, множество ролей, являющееся значением функции *can-assign()* или *can-revoke()*, задается интервалом ролей одного из четырех видов:

$$\begin{aligned}[x, y] &= \{x' \in R, \text{ где } x \leq x' \text{ и } x' \leq y\}; \\ (x, y] &= \{x' \in R, \text{ где } x < x' \text{ и } x' \leq y\}; \\ [x, y) &= \{x' \in R, \text{ где } x \leq x' \text{ и } x' < y\}; \\ (x, y) &= \{x' \in R, \text{ где } x < x' \text{ и } x' < y\}, \end{aligned}$$

где  $x, y \in R$ .

**Пример 5.2.** Рассмотрим иерархию ролей и иерархию административных ролей из примера 5.1. Определим значения функций *can-*

Таблица 5.1. Значения функции  
*can-assign()*

Административная роль	Предварительное условие	Множество ролей
<i>PSO1</i>	<i>ED</i>	[ <i>E1</i> , <i>PL1</i> ]
<i>PSO2</i>	<i>ED</i>	[ <i>E2</i> , <i>PL2</i> ]
<i>DSO</i>	<i>ED and (not PL1)</i>	[ <i>PL2</i> , <i>PL2</i> ]
<i>DSO</i>	<i>ED and (not PL2)</i>	[ <i>PL1</i> , <i>PL1</i> ]

*assign()* и *can-revoke()* соответственно для административных ролей *PSO1*, *PSO2* и *DSO* (табл. 5.1, 5.2).

Таким образом, административная роль *PSO1* позволяет включить для пользователя, уже обладающего ролью *ED*, в множество его авторизованных ролей роли *E1*, *PE1* и *QE1*. Административная роль *DSO* позволяет включить для пользователя, уже обладающего ролью *ED*, в множество его авторизованных ролей роль *PL1*; при этом данный пользователь не должен обладать ролью *PL2*.

Следует отметить, что в общем случае значения функций *can-assign()* и *can-revoke()* определяются независимо друг от друга, т. е. удаление роли из множества авторизованных ролей пользователя может быть выполнено независимо от того, каким образом эта роль была включена в это множество, и наоборот.

Кроме того, значения функции *can-revoke()* определяются в общем случае без учета иерархии ролей. Следовательно, может оказаться, что для некоторой административной роли разрешено удаление роли  $r \in R$  из множества авторизованных ролей  $UA(u)$  пользователя  $u \in U$ . Если при этом найдется роль  $r' \in R$  такая, что  $r \leq r'$  и  $r' \in UA(u)$ , то удаление роли  $r$  из множества авторизованных ролей пользователя  $u$  не будет иметь эффекта, так как права доступа роли  $r$  являются подмножеством прав доступа роли  $r'$ , авторизованным пользователем которой останется пользователь  $u$ .

Одним из путей решения данной проблемы может являться каскадное удаление. При каскадном удалении, если роль  $r$  удаляется из множества авторизованных ролей пользователя, то из этого множества должны быть удалены все роли  $r'$  такие, что  $r \leq r'$ . Однако если для некоторой административной роли  $ar \in AR$  выполняются условия  $r \in can-revoke(ar)$ ,  $r' \notin can-revoke(ar)$ , то каскадное удаление будет невозможно. Таким образом, порядок определения значений функции *can-revoke()* и процедура удаления роли из множества авторизованных ролей пользователя требуют дополнительного уточнения.

Таблица 5.2. Значения функции  
*can-revoke()*

Административная роль	Множество ролей
<i>PSO1</i>	[ <i>E1</i> , <i>PL1</i> ]
<i>PSO2</i>	[ <i>E2</i> , <i>PL2</i> ]
<i>DSO</i>	( <i>ED</i> , <i>DIR</i> )

### 5.3.3. Администрирование множеств прав доступа, которыми обладают роли

При администрировании множеств прав доступа ролей изменяются значения функции *PA()*, для чего определяются специальные административные роли из множества *AR*.

Подход к определению порядка администрирования множеств прав доступа ролей аналогичен подходу, использованному для администрирования множеств авторизованных ролей пользователя.

При этом предварительные условия определяются для прав доступа.

**Определение 5.11.** Для администрирования множеств прав доступа, которыми обладают роли, на множестве административных ролей задаются:

- $can-assign_p: AR \rightarrow CR \times 2^R$  — функция, определяющая для каждой административной роли множество ролей, для которых разрешено включать права доступа в множества прав доступа с использованием данной административной роли при выполнении заданных предварительных условий;

- $can-revoke_p: AR \rightarrow 2^R$  — функция, определяющая для каждой административной роли множество ролей, для которых разрешено удаление прав доступа из множеств прав доступа с использованием данной административной роли.

**Пример 5.3.** Рассмотрим иерархию ролей и иерархию административных ролей из примера 5.1. Определим значения функций *can-assign<sub>p</sub>()* и *can-revoke<sub>p</sub>()* соответственно для административных ролей *PSO1*, *PSO2* и *DSO* (табл. 5.3, 5.4).

Таким образом, административная роль *DSO* позволяет включить права доступа, входящие в множество прав доступа роли *DIR*, в множества прав доступа ролей *PL1*, *PL2*. Причем данные права

Таблица 5.3. Значения функции  
*can-assign<sub>p</sub>()*

Административная роль	Предварительное условие	Множество ролей
<i>DSO</i>	<i>DIR</i>	[ <i>PL1</i> , <i>PL1</i> ]
<i>DSO</i>	<i>DIR</i>	[ <i>PL2</i> , <i>PL2</i> ]
<i>PSO1</i>	<i>PL1 and (not QE1)</i>	[ <i>PE1</i> , <i>PE1</i> ]
<i>PSO1</i>	<i>PL1 and (not PE1)</i>	[ <i>QE1</i> , <i>QE1</i> ]
<i>PSO2</i>	<i>PL2 and (not QE2)</i>	[ <i>PE2</i> , <i>PE2</i> ]
<i>PSO2</i>	<i>PL2 and (not PE2)</i>	[ <i>QE2</i> , <i>QE2</i> ]

Таблица 5.4. Значения функции *can-revoke<sub>p</sub>()*

Административная роль	Множество ролей
<i>DSO</i>	( <i>ED</i> , <i>DIR</i> )
<i>PSO1</i>	[ <i>QE1</i> , <i>QE1</i> ]
<i>PSO1</i>	[ <i>PE1</i> , <i>PE1</i> ]
<i>PSO2</i>	[ <i>QE2</i> , <i>QE2</i> ]
<i>PSO2</i>	[ <i>PE2</i> , <i>PE2</i> ]

доступа могут быть включены в множества прав доступа ролей, находящихся ниже  $PL_1$ ,  $PL_2$  по иерархии. Административная роль  $PSO_1$  позволяет включить права доступа, входящие в множество прав доступа роли  $PL_1$ , в множества прав доступа ролей  $PE_1$  и  $QE_1$ , но только в каждую по отдельности. Административная роль  $DSO$  позволяет удалять права доступа из множеств прав доступа ролей, находящихся в иерархии между ролями  $ED$  и  $DIR$ , а административная роль  $PSO_1$  позволяет удалять права доступа из множеств прав доступа ролей  $PE_1$  и  $QE_1$ .

В общем случае значения функции  $can-revoke_p()$  определяются без учета иерархии ролей. В связи с этим необходимо решать проблему удаления прав доступа ролей, аналогичную рассмотренной ранее проблеме удаления ролей из множеств авторизованных ролей пользователей.

### 5.3.4. Администрирование иерархии ролей

Определение правил администрирования, позволяющих изменять иерархию ролей, является самой сложной задачей, рассматриваемой в модели администрирования РРД. Для решения данной задачи используются подходы, реализованные при определении правил администрирования множеств авторизованных ролей пользователей и прав доступа ролей. Задаются три иерархии, элементами которых соответственно являются:

- возможности — множества прав доступа и других возможностей;
- группы — множества пользователей и других групп;
- объединения — множества пользователей, прав доступа, групп, возможностей и других объединений.

Иерархия объединений является наиболее общей и может включать в себя иерархии возможностей и групп. Определение возможностей и групп требуется для обеспечения соответствия правил администрирования ролей в модели и используемых на практике технологий обработки информации и создания административных структур организаций. Например, для выполнения своих функций пользователю может быть необходим некоторый набор прав доступа, причем отсутствие в этом наборе некоторого права доступа может сделать бессмысленным обладание имеющимися правами.

На основе иерархий возможностей, групп и объединений задается иерархия ролей, элементами которой являются роли-возможности, роли-группы, роли-объединения ( $UP$ -роли).

**Определение 5.12.** Роли-возможности — роли, которые обладают только определенными в соответствующей возможности правами доступа.

**Определение 5.13.** Роли-группы — роли, на которые могут быть авторизованы одновременно только все пользователи соответствующей группы.

**Определение 5.14.** Роли-объединения — роли, которые обладают возможностями, правами доступа и на которые могут быть авторизованы группы пользователей и отдельные пользователи.

Роли-объединения являются общим случаем ролей, рассматриваемых в модели администрирования РРД.

**Определение 5.15.** Роль-объединение  $r_1$  в иерархии ролей пре-восходит роль-объединение  $r_2$ , если выполняются условия:

- возможности и права доступа роли  $r_2$  являются подмножеством возможностей и прав доступа роли  $r_1$ ;
- пользователи и группы пользователей, которые могут быть авторизованы на роль  $r_1$ , являются подмножеством множества пользователей и групп пользователей, которые могут быть авторизованы на роль  $r_2$ .

**Определение 5.16.** Для администрирования возможностей и групп пользователей на множестве административных ролей задаются:

•  $can-assign_a: AR \rightarrow CR \times 2^{UPR}$  — функция, определяющая для каждой административной роли множество ролей-объединений, в множество прав доступа которых разрешено включать возможности с использованием данной административной роли при выполнении заданных предварительных условий;

•  $can-revoke_a: AR \rightarrow 2^{UPR}$  — функция, определяющая для каждой административной роли множество ролей-объединений, из множества прав доступа которых разрешено удаление возможностей с использованием данной административной роли;

•  $can-assign_g: AR \rightarrow CR \times 2^{UPR}$  — функция, определяющая для каждой административной роли множество ролей-объединений, которые разрешено включать в множество авторизованных ролей групп пользователей с использованием данной административной роли при выполнении заданных предварительных условий;

•  $can-revoke_g: AR \rightarrow 2^{UPR}$  — функция, определяющая для каждой административной роли множество ролей-объединений, которые разрешено удалять из множества авторизованных ролей групп пользователей с использованием данной административной роли.

**Замечание.** В множество значений заданных функций входит множество  $2^{UPR}$ , где  $UPR$  — множество ролей объединений, что обеспечивает общность способов определения правил администрирования для всех используемых иерархий.

Необходимость определения иерархий возможностей и групп обусловлена также тем, что построенные на их основе правила администрирования отличны друг от друга. Предварительные условия, определенные в функции  $can-assign_a()$ , используются аналогично предварительным условиям, определенным в функции

*can-assign<sub>p</sub>*( ), а предварительные условия, определенные в функции *can-assign<sub>g</sub>*( ), используются аналогично предварительным условиям, определенным в функции *can-assign*( ). Например, для того чтобы в множество прав доступа роли-объединения была включена возможность, данная возможность должна входить в множества прав доступа ролей, указанных в предварительном условии функции *can-assign<sub>a</sub>*( ), а для того чтобы роль-объединение была включена в множество авторизованных ролей группы пользователей, пользователи данной группы должны уже обладать ролями в соответствии с предварительным условием функции *can-assign<sub>g</sub>*( ).

Включение возможности в множество прав доступа роли-объединения означает, что соответствующая роль-возможность в иерархии ролей станет непосредственно «ниже» роли-объединения. Наоборот, включение роли-объединения в множество авторизованных ролей группы пользователей означает, что соответствующая роль-группа в иерархии ролей станет непосредственно «выше» роли-объединения.

**Определение 5.17.** Для администрирования иерархии ролей (добавления и удаления ролей, включения или удаления отношений иерархии) на множестве административных ролей задается *can-modify*:  $AR \rightarrow 2^{UPR}$  — функция, определяющая для каждой административной роли интервал ролей (исключая границы интервала), на котором возможно изменение иерархии ролей с использованием данной административной роли.

**Пример 5.4.** Рассмотрим иерархию ролей и иерархию административных ролей из примера 5.1. Определим значения функции *can-modify*( ) для административных ролей *PSO1* и *DSO* (табл. 5.5).

Таким образом, административные роли *PSO1* и *DSO* позволяют изменять иерархию ролей соответственно на интервалах (*E1*, *PL1*) и (*ED*, *DIR*).

Описанные способы задания правил администрирования иерархии ролей в общем случае требуют уточнения. Например, возможно возникновение следующей ситуации. Пользователь с административной ролью *DSO* удаляет из иерархии роль *PL1*. В то же время роль *PL1* входит в предварительные условия и участвует в определении интервалов значений функций *can-assign*( ) и *can-revoke*( ).

Таким образом, для административной роли *DSO* должно быть запрещено удаление ролей, участвующих в задании значений и предварительных условий функций администрирования.

Кроме того, возможна следующая ситуация.

**Пример 5.5.** Рассмотрим иерархию ролей и иерархию административных

ролей из примера 5.1 и функцию *can-modify*( ), значения которой заданы в соответствии с табл. 5.5 (рис. 5.4).

Предположим, что пользователь с административной ролью *DSO* добавил в иерархию роли *X* и *Y*. Так как пользователь с административной ролью *PSO1* может изменять иерархию ролей на интервале ролей (*E1*, *PL1*), пусть он определит роль *QE1* как старшую над ролью *PE1*. Таким образом, административная роль *PSO1* позволяет определить роль *X* старшей над ролью *Y*, несмотря на то, что эти роли не входят в определенный для *PSO1* интервал (*E1*, *PL1*).

Возможна реализация нескольких подходов по определению порядка администрирования иерархии ролей в данной ситуации. Во-первых, можно запретить пользователю с административной ролью *DSO* добавлять роли *X* и *Y*, так как это нарушает целостность интервала (*E1*, *PL1*), на котором разрешено администрирование для роли *PSO1*. Во-вторых, можно запретить пользователю с административной ролью *PSO1* определять роль *QE1* старшей над ролью *PE1*. В-третьих, можно разрешить выполнять все запрашиваемые действия с использованием административных ролей *DSO* и *PSO1*.

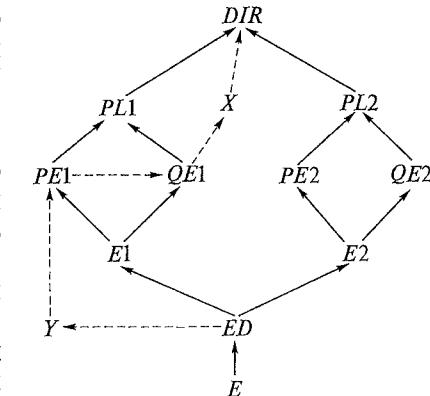


Рис. 5.4. Пример модификации иерархии ролей

## 5.4. МОДЕЛЬ МАНДАТНОГО РРД

### 5.4.1. Защита от угрозы конфиденциальности информации

Ролевое разграничение доступа является развитием дискреционного разграничения доступа. В то же время оно является достаточно гибким, и, используя механизм ролей, возможна реализация мандатной политики безопасности, ориентированной на защиту от угрозы конфиденциальности информации.

Рассмотрим подход [21], реализующий мандатное разграничение доступа на основе базовой модели РРД.

Используем следующие обозначения:

*U* — множество пользователей (субъектов);  
*O* — множество объектов;

Таблица 5.5. Значения функции *can-modify*( )

Административная роль	Множество ролей
<i>PSO1</i>	( <i>E1</i> , <i>PL1</i> )
<i>DSO</i>	( <i>ED</i> , <i>DIR</i> )

- $(L, \leq)$  — решетка уровней конфиденциальности;
- $c: U \rightarrow L$  — функция уровня доступа пользователя;
- $c: O \rightarrow L$  — функция уровня конфиденциальности объекта;
- $A = \{read, write\}$  — виды доступа.

Будем различать два вида мандатного разграничения доступа: либеральный и строгий (Белла—Лападула).

**Определение 5.18.** Доступ  $(u, o, r)$  является безопасным для либерального мандатного разграничения доступа, если выполняется одно из условий:

- $r = read$  и  $c(u) \geq c(o)$  (*ss-свойство*);
- $r = write$  и, если существует доступ  $(u, o', read)$ , то  $c(o) \geq c(o')$  (либеральное *\*-свойство*).

**Определение 5.19.** Доступ  $(u, o, r)$  является безопасным для строгого мандатного разграничения доступа, если выполняется одно из условий:

- $r = read$  и  $c(u) \geq c(o)$  (*ss-свойство*);
- $r = write$  и, если существует доступ  $(u, o', read)$ , то  $c(o) = c(o')$  (строгое *\*-свойство*).

Построим систему РРД. Пусть

$$\begin{aligned} R &= \{x\_read \mid x \in L\} \cup \{x\_write \mid x \in L\} — \text{множество ролей;} \\ P &= \{(o, read) \mid o \in O\} \cup \{(o, write) \mid o \in O\} — \text{множество прав} \end{aligned}$$

доступа.

Зададим на множестве ролей  $R$  иерархию; при этом иерархии ролей на множествах  $\{x\_read \mid x \in L\}$  и  $\{x\_write \mid x \in L\}$  будут независимы.

**Определение 5.20.** Иерархией на множестве ролей  $R$  в соответствии с требованиями либерального мандатного разграничения доступа называется отношение частичного порядка  $\llleq$ , где для  $r, r' \in R$  справедливо неравенство  $r \leq r'$ , если выполняется одно из условий:

- $r = x\_read, r' = x'\_read$  и  $x \leq x'$ ;
- $r = x\_write, r' = x'\_write$  и  $x' \leq x$ .

**Определение 5.21.** Иерархией на множестве ролей  $R$  в соответствии с требованиями строгого мандатного разграничения доступа называется отношение частичного порядка  $\llleq$ , где для  $r, r' \in R$  справедливо неравенство  $r \leq r'$ , если выполняется одно из условий:

- $r = x\_read, r' = x'\_read$  и  $x \leq x'$ ;
- $r = x\_write, r' = x'\_write$  и  $x = x'$  (каждая роль вида  $x\_write$  сравнивается только сама с собой).

**Определение 5.22.** Модель РРД соответствует требованиям либерального мандатного разграничения доступа, если иерархия ролей соответствует требованиям определения 5.20 и выполняются ограничения:

- ограничение функции  $UA()$  — для каждого пользователя  $u \in U$  роль  $x\_read = \bigoplus (UA(u) \cap \{y\_read \mid y \in L\}) \in UA(u)$  (здесь  $x = c(u)$ ) и  $x\_write = \bigoplus \{y\_write \mid y \in L\} \in UA(u)$  (здесь  $x = \otimes L$ );

- ограничение функции  $roles()$  — для каждой сессии  $s \in S$  множество  $roles(s) = \{x\_read, x\_write\}$ ;

- ограничение функции  $PA()$  — должно выполняться:  
для каждого  $x \in L$  доступ  $(o, read) \in PA(x\_read)$  тогда и только тогда, когда доступ  $(o, write) \in PA(x\_write)$ ;

- для каждого доступа  $(o, read)$  существует единственная роль  $x\_read: (o, read) \in PA(x\_read)$  (здесь  $x = c(o)$ ).

**Определение 5.23.** Модель РРД соответствует требованиям строгого мандатного разграничения доступа, если иерархия ролей соответствует требованиям определения 5.21 и выполняются ограничения:

- ограничение функции  $UA()$  — для каждого пользователя  $u \in U$  роль  $x\_read = \bigoplus (UA(u) \cap \{y\_read \mid y \in L\}) \in UA(u)$  (здесь  $x = c(u)$ ) и  $UA(u) = \{y\_write \mid y \in L\}$ ;

- ограничение функции  $roles()$  — для каждой сессии  $s \in S$  множество  $roles(s) = \{x\_read, x\_write\}$ ;

- ограничение функции  $PA()$  — должно выполняться:  
для каждого  $x \in L$  доступ  $(o, read) \in PA(x\_read)$  тогда и только тогда, когда доступ  $(o, write) \in PA(x\_write)$ ;

- для каждого доступа  $(o, read)$  существует единственная роль  $x\_read: (o, read) \in PA(x\_read)$  (здесь  $x = c(o)$ ).

Таким образом, требования соответствия либеральному и строгому мандатному разграничению доступа для моделей РРД совпадают во всем, кроме требований к соответствующей иерархии ролей и ограничениям на функцию  $UA$ .

**Теорема 5.1.** Если модель РРД соответствует требованиям либерального или строгого мандатного разграничения доступа, то в ней для  $o, o' \in O$  таких, что  $c(o) > c(o')$ , невозможно возникновение информационных потоков от  $o$  к  $o'$ .

**Доказательство.** Доказательство выполним для либерального мандатного разграничения доступа. Докажем от противного. Пусть  $o, o' \in O, c(o) > c(o')$  и возможно возникновение информационного потока от  $o$  к  $o'$ , т.е. существуют роли  $r, r' \in R$  и сессия  $s \in S$  такие, что  $(o, read) \in PA(r), (o', write) \in PA(r')$  и  $r, r' \in roles(s)$ . По определению 5.22 должно выполняться условие  $r = c(o)\_read, r' = c(o')\_write$  и, следовательно,  $c(o) = c(o')$  — противоречие.

Доказательство для строгого мандатного разграничения доступа выполняется аналогично.

#### 5.4.2. Защита от угрозы конфиденциальности и целостности информации

Модель мандатного РРД, в первую очередь, ориентирована на обеспечение защиты от угрозы конфиденциальности информации. В то же время возможно доопределение свойств модели с

целью анализа систем защиты от двух угроз: целостности информации и конфиденциальности информации.

Используем обозначения модели РРД защиты от угрозы конфиденциальности информации. Также используем следующие обозначения:

- $(LI, \leq)$  — решетка уровней целостности информации;
- $ci: U \rightarrow LI$  — функция уровня целостности пользователя;
- $ci: O \rightarrow LI$  — функция уровня целостности объекта.

Также как в модели РРД защиты от угрозы конфиденциальности информации, будем различать два вида мандатного контроля целостности информации: либеральный и строгий.

**Определение 5.24.** Доступ  $(u, o, r)$  является безопасным для либерального мандатного контроля целостности, если выполняется одно из условий:

- $r = read$  и  $ci(u) \leq ci(o)$ ;
- $r = write$  и, если существует доступ  $(u, o', read)$ , то  $ci(o) \leq ci(o')$ .

**Определение 5.25.** Доступ  $(u, o, r)$  является безопасным для строгого мандатного контроля целостности, если выполняется одно из условий:

- $r = read$  и  $ci(u) \leq ci(o)$ ;
- $r = write$  и, если существует доступ  $(u, o', read)$ , то  $ci(o) = ci(o')$ .

Рассмотрим иерархию ролей на множестве  $RI = \{xi\_read \mid xi \in LI\} \cup \{xi\_write \mid xi \in LI\}$ . Иерархии ролей на множествах  $\{xi\_read \mid xi \in LI\}$  и  $\{xi\_write \mid xi \in LI\}$  независимы.

**Определение 5.26.** Иерархией на множестве ролей  $RI$  в соответствии с требованиями либерального мандатного контроля целостности называется отношение частичного порядка  $\leq$ , где для  $r, r' \in RI$  справедливо неравенство  $r \leq r'$ , если выполняется одно из условий:

- $r = xi\_read, r' = xi'\_read$  и  $xi' \leq xi$ ;
- $r = xi\_write, r' = xi'\_write$  и  $xi \leq xi'$ .

**Определение 5.27.** Иерархией на множестве ролей  $RI$  в соответствии с требованиями строгого мандатного контроля целостности называется отношение частичного порядка  $\leq$ , где для  $r, r' \in RI$  справедливо неравенство  $r \leq r'$ , если выполняется одно из условий:

- $r = xi\_read, r' = xi'\_read$  и  $xi' \leq xi$ ;
- $r = xi\_write, r' = xi'\_write$  и  $xi \leq xi'$  (каждая роль вида  $xi\_write$  сравнима только сама с собой).

**Определение 5.28.** Модель РРД соответствует требованиям либерального мандатного контроля целостности, если иерархия ролей  $RI$  соответствует требованиям определения 5.26 и выполняются ограничения:

- ограничение функции  $UA()$  — для каждого пользователя  $u \in U$  роль  $xi\_read = \bigoplus (UA(u) \cap \{yi\_read \mid yi \in LI\}) \in UA(u)$  (здесь  $xi = ci(u)$ ) и  $xi\_write = \bigoplus \{yi\_write \mid yi \in LI\} \in UA(u)$  (здесь  $xi = \bigoplus LI$ );

• ограничение функции  $roles()$  — для каждой сессии  $s \in S$  множество  $roles(s) = \{xi\_read, xi\_write\}$ ;

• ограничение функции  $PA()$  — должно выполняться:

для каждого  $xi \in LI$  доступ  $(o, read) \in PA(xi\_read)$  тогда и только тогда, когда доступ  $(o, write) \in PA(xi\_write)$ ;

для каждого доступа  $(o, read)$  существует единственная роль  $xi\_read: (o, read) \in PA(xi\_read)$  (здесь  $xi = ci(o)$ ).

**Определение 5.29.** Модель РРД соответствует требованиям строгого мандатного контроля целостности, если иерархия ролей  $RI$  соответствует требованиям определения 5.27 и выполняются ограничения:

- ограничение функции  $UA()$  — для каждого пользователя  $u \in U$  роль  $xi\_read = \bigoplus (UA(u) \cap \{yi\_read \mid yi \in LI\}) \in UA(u)$  (здесь  $xi = ci(u)$ ) и  $UA(u) = \{yi\_write \mid yi \in LI\}$ ;

• ограничение функции  $roles()$  — для каждой сессии  $s \in S$  множество  $roles(s) = \{xi\_read, xi\_write\}$ ;

• ограничение функции  $PA()$  — должно выполняться:

для каждого  $xi \in LI$  доступ  $(o, read) \in PA(xi\_read)$  тогда и только тогда, когда доступ  $(o, write) \in PA(xi\_write)$ ;

для каждого доступа  $(o, read)$  существует единственная роль  $xi\_read: (o, read) \in PA(xi\_read)$  (здесь  $xi = ci(o)$ ).

Построим систему РРД мандатного разграничения доступа и контроля целостности. Пусть

$$R = (\{x\_read \mid x \in L\} \times \{xi\_read \mid xi \in LI\}) \cup (\{x\_write \mid x \in L\} \times \{xi\_write \mid xi \in LI\}) — множество ролей.$$

Зададим на множестве ролей  $R$  иерархию; при этом возможно произвольное сочетание иерархий ролей либерального или строгого мандатного разграничения доступа с либеральным или строгим контролем целостности.

Иерархии ролей на множествах  $\{x\_read \mid x \in L\} \times \{xi\_read \mid xi \in LI\}$  и  $\{x\_write \mid x \in L\} \times \{xi\_write \mid xi \in LI\}$  независимы.

Рассмотрим реализацию либерального мандатного разграничения доступа и контроля целостности.

**Определение 5.30.** Иерархией на множестве ролей  $R$  в соответствии с требованиями либерального мандатного разграничения доступа и контроля целостности называется отношение частичного порядка  $\leq$ , где для  $r = (rc, ri), r' = (rc', ri') \in R$  (при этом  $rc, rc' \in \{x\_read \mid x \in L\} \cup \{x\_write \mid x \in L\}, ri, ri' \in \{xi\_read \mid xi \in LI\} \cup \{xi\_write \mid xi \in LI\}$ ) справедливо неравенство  $r \leq r'$ , если  $rc \leq rc'$  в иерархии ролей либерального мандатного разграничения доступа (определение 5.20) и  $ri \leq ri'$  в иерархии ролей либерального контроля целостности (определение 5.26).

**Определение 5.31.** Модель РРД соответствует требованиям либерального мандатного разграничения доступа и контроля целостности, если иерархия ролей  $R$  соответствует требованиям определения 5.30 и выполняются ограничения:

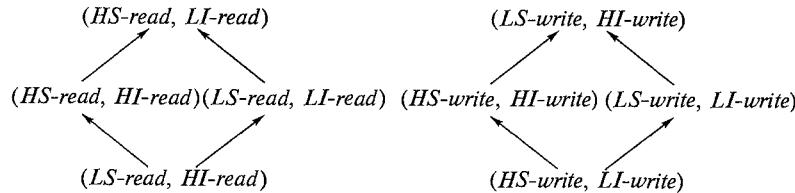


Рис. 5.5. Либеральные мандатное разграничение доступа и мандатный контроль целостности

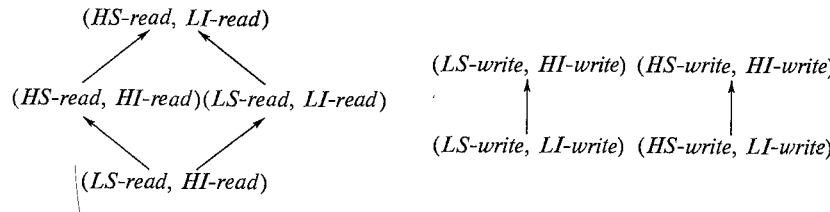


Рис. 5.6. Строгое мандатное разграничение доступа и либеральный мандатный контроль целостности

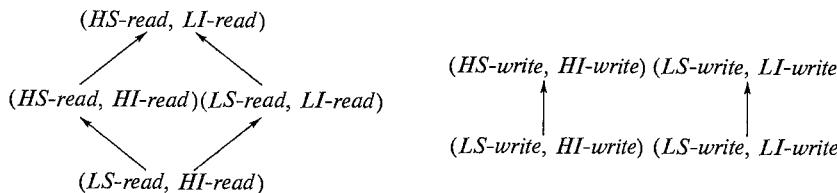


Рис. 5.7. Либеральное мандатное разграничение доступа и строгий мандатный контроль целостности

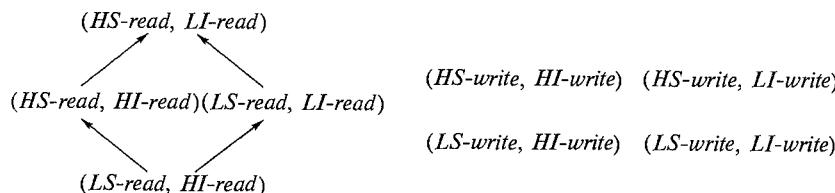


Рис. 5.8. Строгие мандатное разграничение доступа и мандатный контроль целостности

- ограничение функции  $UA()$  — для каждого пользователя  $u \in U$  роль  $(x\_read, xi\_read) = \oplus (UA(u) \cap (\{y\_read | y \in L\} \times \{yi\_read | yi \in LI\})) \in UA(u)$  (здесь  $x = c(u)$ ,  $xi = ci(u)$ ) и  $(x\_write, xi\_write) = \oplus (\{y\_write | y \in L\} \times \{yi\_write | yi \in LI\}) \in UA(u)$  (здесь  $x = \otimes L$ ,  $xi = \oplus LI$ );

- ограничение функции  $roles()$  — для каждой сессии  $s \in S$  множество  $roles(s) = \{(x\_read, xi\_read), (x\_write, xi\_write)\}$ ;

- ограничение функции  $PA()$  — должно выполняться:  
для каждого  $x \in L$ ,  $xi \in LI$  доступ  $(o, read) \in PA((x\_read, xi\_read))$  тогда и только тогда, когда доступ  $(o, write) \in PA((x\_write, xi\_write))$ ;  
для каждого доступа  $(o, read)$  существует единственная роль  $(x\_read, xi\_read)$ :  $(o, read) \in PA((x\_read, xi\_read))$  (здесь  $x = c(o)$ ,  $xi = ci(o)$ ).

Для решеток уровней конфиденциальности  $(L, \leq) = \{LS, HS\}$  и уровней целостности  $(LI, \leq) = \{LI, HI\}$  на рис.5.5—5.8 представлены иерархии ролей для четырех возможных сочетаний иерархий ролей либерального или строгого мандатного разграничения доступа с либеральным или строгим мандатным контролем целостности.

### Контрольные вопросы

- Какие основные проблемы определения правил изменения иерархии ролей рассматриваются в модели администрирования РРД?
- Какого вида ограничения, описанные в базовой модели РРД, могут быть использованы при определении требований либерального и строгого мандатного разграничения доступа?
- Докажите, что при соответствии модели РРД требованиям либерального или строгого мандатного разграничения доступа для каждого доступа  $(o, write)$  существует единственная роль  $x\_write$  такая, что  $(o, write) \in PA(x\_write)$  (здесь  $x = c(o)$ ).
- Каким образом в определениях либерального или строгого мандатного разграничения доступа модели РРД реализованы  $ss$ -свойство и  $'$ -свойство, определенные в классической модели Белла—Лападула?
- Постройте графы иерархии ролей для модели мандатного РРД для решеток уровней конфиденциальности  $(L, \leq) = \{LS, MS, HS\}$  и уровней целостности  $(LI, \leq) = \{LI, MI, HI\}$ .

# СУБЪЕКТНО-ОРИЕНТИРОВАННАЯ МОДЕЛЬ ИЗОЛИРОВАННОЙ ПРОГРАММНОЙ СРЕДЫ

## 6.1. ОСНОВНЫЕ ПОНЯТИЯ. МОНИТОР БЕЗОПАСНОСТИ ОБЪЕКТОВ

Как правило, модели, связанные с реализацией политики безопасности, не учитывают возможности субъектов по изменению конфигурации или параметров функционирования компьютерной системы, которые могут привести к изменению ее свойств и как предельный случай к полной неприменимости той или иной модели к описанию отношений субъект — объект в измененной системе.

В данной главе на основе [6, 7] рассматривается субъектно-ориентированная модель изолированной программной среды, в которой основное внимание уделяется определению порядка безопасного взаимодействия субъектов системы, описанию и обоснованию необходимых условий реализации в системе изолированной программной среды.

Будем полагать, что в любой дискретный момент времени множество субъектов компьютерной системы не пусто (в противном случае соответствующие моменты времени исключаются из рассмотрения и рассматриваются отрезки с ненулевой мощностью множества субъектов).

**Аксиома 6.1.** Субъекты в компьютерной системе могут быть порождены только активной компонентой (субъектами) из объектов.

Определим механизм порождения новых субъектов.

**Определение 6.1.** Объект  $o$  называется источником для субъекта  $s'$ , если существует субъект  $s$ , в результате воздействия которого на объект  $o$  в системе возникает субъект  $s'$ . Субъект  $s$ , порождающий новый субъект из объекта  $o$ , в свою очередь, называется активизирующим субъектом для субъекта  $s'$ , которого назовем порожденным субъектом.

Введем следующее обозначение:

$Create(s, o) \rightarrow s'$  — операция порождения субъектов (из объекта  $o$  порожден субъект  $s'$  при активизирующем воздействии субъекта  $s$ ).

Операция  $Create()$  задает отображение декартова произведения множеств субъектов и объектов на объединение множества субъектов с пустым множеством. Заметим также, что в компью-

терной системе действует дискретное время и фактически новый субъект  $s'$  порождается в момент времени  $t+1$  относительно момента времени  $t$ , в который произошло воздействие порождающего субъекта на объект-источник.

Очевидно, что операция порождения субъектов зависит как от свойств активизирующего субъекта, так и от содержания объекта-источника.

Считаем, что если  $Create(s, o) \rightarrow \emptyset$ , то порождение нового субъекта из объекта  $o$  при активизирующем воздействии  $s$  невозможно. Так, практически во всех операционных средах существует понятие исполняемого файла — объекта, способного быть источником для порождения субъекта.

**Определение 6.2.** Объект  $o$  в момент времени  $t$  ассоциирован с субъектом  $s$ , если состояние объекта  $o$  повлияло на состояние субъекта  $s$  в следующий момент времени (т. е. субъект  $s$  использует информацию, содержащуюся в объекте  $o$ ).

Введем обозначение «множество объектов  $\{o\}$ , ассоциировано с субъектом  $s$  в момент времени  $t$ »:

$$s(\{o\}_t)$$

Субъект в общем случае реализует некоторое отображение множества ассоциированных объектов в момент времени  $t$  на множество ассоциированных объектов в момент времени  $t+1$ . В связи с этим можно выделить ассоциированные объекты, изменение которых изменяет вид отображения ассоциированных объектов (объекты, содержащие, как правило, код программы): функционально ассоциированные и ассоциированные объекты-данные (являющиеся аргументом операции, но не изменяющие вида отображения). Далее под ассоциированными объектами понимаются функционально ассоциированные объекты, в иных случаях делаются уточнения.

**Замечание 6.1.** В момент порождения субъекта  $s'$  из объекта  $o$  он является ассоциированным объектом для субъекта  $s'$ .

**Определение 6.3.** Потоком информации от объекта  $o$  (источника) к объекту  $o'$  (приемнику) называется преобразование информации в объекте  $o'$ , реализуемое в субъекте  $s$  и зависящее от  $o$ .

Заметим, что как  $o$ , так и  $o'$  могут быть ассоциированными или неассоциированными объектами для субъекта  $s$ , а также «пустыми» объектами.

Используем следующее обозначение:

$Stream(s, o) \rightarrow o'$  — поток информации от объекта  $o$  к объекту  $o'$ .

В определении подчеркнуто, что поток информации рассматривается не между субъектом и объектом, а между объектами, например между объектом и ассоциированными объектами субъекта (либо между двумя объектами). Активная роль субъекта выражается в реализации данного потока (это означает, что операция

порождения потока локализована в субъекте и отображается состоянием его функционально ассоциированных объектов). Отметим, что операция *Stream()* может создавать новый объект или уничтожать его.

**Замечание 6.2.** Отношение «между объектами существует информационный поток» является транзитивным (относительно пары субъектов), т. е. если существует  $\text{Stream}(s, o) \rightarrow o'$  и существует  $\text{Stream}(s', o') \rightarrow o''$ , то существует и  $\text{Stream}((s, s'), o) \rightarrow o''$ .

**Определение 6.4.** Доступом субъекта  $s$  к объекту  $o$  будем называть порождение потока информации между некоторым объектом (например, между ассоциированными с субъектом объектами  $s(\{o'\})$ ) и объектом  $o$ .

Выделим все множество потоков  $P$  для фиксированной декомпозиции компьютерной системы на субъекты и объекты во все моменты времени (все множество потоков является объединением потоков по всем моментам дискретного времени) и произвольным образом разобьем его на два непересекающихся подмножества  $N$  и  $L$ , где

$$P = N \cup L, N \cap L = \emptyset.$$

Пусть

$N$  — подмножество потоков, характеризующее несанкционированный доступ;

$L$  — подмножество потоков, характеризующих легальный доступ.

**Определение 6.5.** Правила разграничения доступа субъектов к объектам есть формально описанные потоки, принадлежащие подмножеству  $L$ .

В предлагаемой субъектно-ориентированной модели не производится уточнений известных моделей политик безопасности (политика безопасности описывает только критерии разбиения на множества  $L$  и  $N$ ), но формулируются условия корректного существования элементов компьютерной системы, обеспечивающих реализацию той или иной политики безопасности. Поскольку критерий разбиения на множества  $L$  и  $N$  не связан со следующими далее утверждениями (постулируется лишь наличие субъекта, реализующего фильтрацию потоков), то можно говорить об инвариантности субъектно-ориентированной модели относительно любой принятой в компьютерной системе политики безопасности (не противоречащей условиям утверждений).

**Определение 6.6.** Объекты  $o$  и  $o'$  тождественны в момент времени  $t$ , если они совпадают, как слова, записанные в одном языке.

Для введения понятия тождественности субъектов условимся о наличии процедуры сортировки ассоциированных объектов, которая позволяет говорить о возможности попарного сравнения. На практике всегда существует алгоритм, обеспечивающий возмож-

ность попарного сравнения и зависящий от конкретной архитектуры компьютерной системы. Например, достаточно легко выделить и попарно сравнивать участки оперативной памяти, отвечающие коду программ (отличающиеся абсолютным адресом загрузки в оперативную память) или содержанию переменных и массивов.

**Определение 6.7.** Субъекты  $s$  и  $s'$  тождественны в момент времени  $t$ , если попарно тождественны все ассоциированные с ними объекты.

**Замечание 6.3.** Порожденные субъекты тождественны, если тождественны порождающие субъекты и объекты-источники.

Верность данного замечания следует из тождества функционально ассоциированных объектов в порождающих субъектах, которые отвечают за порождение нового субъекта, а также из тождества аргументов (ассоциированных объектов-данных), которые отвечают объектам-источникам.

Для разделения всего множества потоков в компьютерной системе на подмножества  $L$  и  $N$  необходимо существование субъекта, который:

- активизировался бы при возникновении любого потока;
- производил бы фильтрацию потоков в соответствии с принадлежностью множествам  $L$  или  $N$ .

**Определение 6.8.** Монитор обращений (МО) — субъект, активизирующийся при возникновении потока от любого субъекта к любому объекту.

Можно выделить два вида МО:

- индикаторный МО — устанавливающий только факт обращения субъекта к объекту;
- содержательный МО — субъект, функционирующий таким образом, что при возникновении потока от ассоциированного объекта  $o$  любого субъекта  $s$  к объекту  $o'$  или обратно существует ассоциированный с МО объект  $o_0$  (в данном случае речь идет об ассоциированных объектах-данных), тождественный объекту  $o$  или одному из  $s(\{o\})$ ; при этом содержательный МО полностью участвует в потоке от субъекта к объекту (в том смысле, что информация проходит через его ассоциированные объекты-данные и существует тождественное отображение объекта на какой-либо ассоциированный объект МО).

Теперь сформулируем понятие монитора безопасности (в литературе также применяется понятие монитора ссылок). Это понятие связано с упоминаемой выше задачей фильтрации потоков. Поскольку целью является обеспечение безопасности компьютерной системы, то и целевая функция монитора — фильтрация с целью обеспечения безопасности (разделение на  $N$  и  $L$  задано априорно).

**Определение 6.9.** Монитор безопасности объектов (МБО) — монитор обращений, который разрешает поток, принадлежащий

только множеству легального доступа  $L$ . Разрешение потока в данном случае понимается как выполнение операции над объектом — получателем потока, а запрещение — как невыполнение (т. е. неизменность объекта — получателя потока).

Монитор безопасности объектов фактически является механизмом реализации политики безопасности в компьютерной системе.

## 6.2. МОНИТОР БЕЗОПАСНОСТИ СУБЪЕКТОВ

Представляется очевидным, что при изменении функционально ассоциированных с МБО объектов могут измениться и свойства самого МБО, заключающиеся в фильтрации потоков и, как следствие, могут возникнуть потоки, принадлежащие множеству  $N$  (рис. 6.1).

Введем в связи с этим понятие корректности субъектов.

**Определение 6.10.** Субъекты  $s$  и  $s'$  называются невлияющими друг на друга (или корректными относительно друг друга), если в любой момент времени отсутствует поток (изменяющий состояние объекта) между любыми объектами  $o$  и  $o'$ , ассоциированными соответственно с субъектами  $s$  и  $s'$ . Таким образом, выполняется условие

для  $t \geq 0$ ,  $o \in s(\{o\}_t)$ ,  $o' \in s'(\{o\}_t)$  не существует субъекта  $s''$  такого, что или  $\text{Stream}(s'', o) \rightarrow o'$ , или  $\text{Stream}(s'', o') \rightarrow o$ .

Смысл понятия корректности можно пояснить на примере: существующие в едином пространстве ОС программы не должны иметь функциональных возможностей изменения «чужого» вектора кода и состояния переменных.

**Определение 6.11.** Субъекты  $s$  и  $s'$  называются абсолютно невлияющими друг на друга (или абсолютно корректными относи-

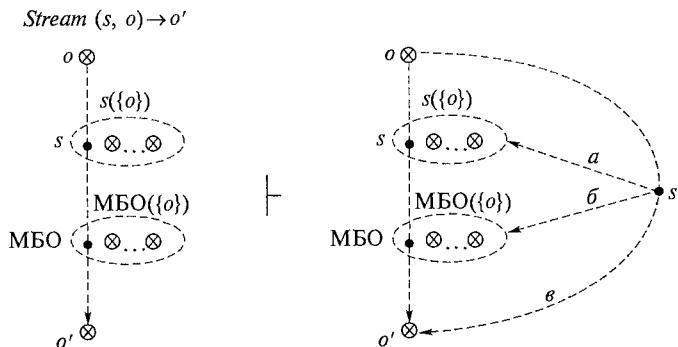


Рис. 6.1. Возможные пути обхода политики безопасности нарушителем  $s'$ :  
а — изменение функциональности субъекта; б — изменение функциональности МБО; в — реализация информационного потока в обход МБО

тельно друг друга), если множества ассоциированных объектов указанных субъектов не имеют пересечения. Таким образом, выполняется условие

для  $t \geq 0$   $s(\{o\}_t) \cap s'(\{o\}_t) = \emptyset$ .

Абсолютная корректность легко достижима в случае виртуального адресного пространства.

Определение абсолютной корректности позволяет сформулировать достаточные условия гарантированного осуществления только легального доступа.

**Теорема 6.1 (достаточное условие гарантированного выполнения политики безопасности в компьютерной системе 1).** МБО разрешает порождение потоков только из множества  $L$ , если все существующие в системе субъекты абсолютно корректны относительно него и друг друга.

**Доказательство.** Условие абсолютной корректности (по определению 6.11) предполагает неизменность функционально ассоциированных объектов МБО (поскольку потоков, изменяющих ассоциированные объекты МБО, не существует). С другой стороны, такие потоки могут появиться при изменении ассоциированных объектов, принадлежащих другим субъектам компьютерной системы, так как при этом изменятся свойства субъектов, что приводит по существу к появлению новых субъектов, для которых правила разграничения доступа могут быть не установлены. Условие корректности субъектов относительно друг друга делает это невозможным (по определению абсолютной корректности). Это, в свою очередь, означает, что МБО реализует только потоки из подмножества  $L$ . Теорема доказана.

Однако сформулированная теорема накладывает весьма жесткие и трудноисполнимые условия на свойства субъектов в компьютерной системе. Кроме того, невозможно гарантировать корректность любого субъекта, активизируемого в компьютерной системе, относительно МБО. В связи с этим логично ограничить множество порождаемых субъектов, которые априорно корректны относительно МБО. Введем определение монитора порождения субъектов (по аналогии с монитором обращений) и МБО.

**Определение 6.12.** Монитор порождения субъектов (МПС) — субъект, активизирующийся при любом порождении субъектов.

По аналогии с переходом от МО к МБО введем понятие монитора безопасности субъектов.

**Определение 6.13.** Монитор безопасности субъектов (МБС) — субъект, который разрешает порождение субъектов только для фиксированного подмножества пар активизирующих субъектов и объектов-источников.

Воздействие МБС выделяет во всем множестве субъектов  $S$  подмножество разрешенных субъектов  $E$ .

Сформулируем ряд базовых определений, которые будут использоваться в дальнейшем.

**Определение 6.14.** Компьютерная система называется замкнутой по порождению субъектов (обладает замкнутой программной средой), если в ней действует МБС, разрешающий порождение только фиксированного конечного подмножества субъектов для любых объектов-источников, рассматриваемых для фиксированной декомпозиции компьютерной системы на субъекты и объекты.

Однако замкнутости компьютерной системы по порождению субъектов недостаточно для описания свойств системы в части защищенности, поскольку необходимо обеспечить корректность порождаемых МБС субъектов относительно него самого и МБО. Механизм замкнутой программной среды сокращает множество возможных субъектов до некоторого множества фиксированной мощности, но при этом допускает существование некорректных субъектов, включенных в замкнутую среду.

Сформулируем определение изолированности программной среды.

**Определение 6.15.** Множество субъектов программной среды называется изолированным (абсолютно изолированным), если в ней действует МБС и субъекты из порожденного множества корректны (абсолютно корректны) относительно друг друга и МБС.

**Замечание 6.4.** Любое подмножество субъектов изолированной (абсолютно изолированной) программной среды, включающее МБС, также составляет изолированную (абсолютно изолированную) среду.

**Замечание 6.5.** Дополнение изолированной (абсолютно изолированной) программной среды субъектом, корректным (абсолютно корректным) относительно любого из числа входящих в изолированную (абсолютно изолированную) среду, оставляет ее изолированной (абсолютно изолированной).

Теперь возможно переформулировать достаточное условие гарантированного выполнения политики безопасности следующим образом.

**Теорема 6.2 (достаточное условие гарантированного выполнения политики безопасности в компьютерной системе 2).** Если в абсолютно изолированной программной среде существует МБО и порождаемые субъекты абсолютно корректны с МБО, МБС и другими субъектами, а также МБС абсолютно корректен с МБО, то в такой программной среде МБО разрешает порождение потоков только из множества  $L$ .

**Доказательство.** Из определения абсолютной изолированности следует возможность существования в программной среде только конечного множества субъектов, которые, в свою очередь, корректны относительно МБС (по определению 6.15 и замечаниям 6.4 и 6.5). По условию теоремы (корректность МБО с любым из

порождаемых субъектов и МБС) ассоциированные объекты МБО могут изменяться только самим МБО, следовательно, в компьютерной системе реализуются только потоки, принадлежащие множеству  $L$ . Теорема доказана.

Легко видеть, что данное утверждение является более конструктивным относительно предыдущего достаточного условия гарантированной защищенности, поскольку ранее требовалась корректность МБО относительно произвольного субъекта, что практически невозможно. В данном же случае множество субъектов ограничено за счет применения механизма МБС, а следовательно, можно убедиться в попарной корректности порождаемых субъектов.

### 6.3. ИЗОЛИРОВАННАЯ ПРОГРАММНАЯ СРЕДА

При рассмотрении операции порождения субъекта возникает проблема, связанная с тем, что в реальных компьютерных системах одинаково поименованные объекты в различные моменты времени могут иметь различное размещение (например, размещение в разных каталогах).

Предположим, что зафиксировано состояние объекта  $o$  в некоторый момент времени  $t_0$ . Будем обозначать состояние объекта  $o$  в момент времени  $t$  как  $o[t]$ .

**Определение 6.16.** Операция порождения субъекта  $Create(s, o) \rightarrow s'$  называется порождением с контролем неизменности объекта-источника, если для любого момента времени  $t > t_0$ , в который активизирована операция порождения объекта  $Create()$ , порождение субъекта  $s'$  возможно только при тождественности объектов  $o[t_0]$  и  $o[t]$ .

**Замечание 6.6.** В условиях определения 6.16 порожденные субъекты  $s'[t_1]$  и  $s'[t_2]$  тождественны, если  $t_1 > t_0$  и  $t_2 > t_0$ . При  $t_1 = t_2$  порождается один и тот же субъект.

При порождении субъектов с контролем неизменности объекта в компьютерной системе допустимы потоки от субъектов к объектам-источникам, участвующим в порождении субъектов, с изменением их состояния.

**Теорема 6.3 (базовая теорема изолированной программной среды (ИПС)).** Если с момента времени  $t_0$  в изолированной (абсолютно изолированной) программной среде действует только порождение субъектов с контролем неизменности объекта и все субъекты корректны (абсолютно корректны) относительно друг друга, то в любой момент времени  $t > t_0$  программная среда также остается изолированной (абсолютно изолированной).

**Доказательство.** По условию теоремы в программной среде возможно существование потоков, изменяющих состояние объектов,

не ассоциированных в этот момент времени с каким-либо субъектом. Если объект с измененным состоянием не является источником для порождения субъекта, то множество субъектов изолированной среды нерасширяемо, в противном случае (измененный объект является источником для порождения субъекта) по условиям теоремы (порождение субъекта с контролем) порождение субъекта невозможно. Следовательно, мощность множества субъектов не может превышать той, которая была зафиксирована до изменения состояния любого объекта. По замечанию 6.5 (о замкнутости множества субъектов в изолированной программной среде с невозрастием мощности множества субъектов) получим, что множество субъектов программной среды изолировано. Теорема доказана.

Можно сформулировать методологию проектирования (разработки) гарантированно защищенных компьютерных систем. Сущность данной методологии состоит в том, что при проектировании защитных механизмов компьютерной системы необходимо опираться на совокупность приведенных выше (теоремы 6.1 — 6.3) достаточных условий, которые должны быть реализованы для субъектов, что гарантирует защитные свойства, определенные при реализации МБО в компьютерной системе (т. е. гарантированное выполнение заданной МБО политики безопасности).

Рассмотренная концепция изолированной программной среды является расширением классического подхода к реализации ядра безопасности. Обычно модель функционирования ядра безопасности изображается в виде схемы, представленной на рис. 6.2.

Объект управления (ОУ) содержит в себе информацию о потоках множества  $L$  (защита по «белому списку» — разрешенные потоки) или  $N$  (защита по «черному списку» — запрещенные потоки).

Для учета влияния субъектов в компьютерной системе на систему защиты необходимо рассматривать расширенную схему взаимодействия элементов системы и гарантирования политики безопасности.

При этом должна быть особо подчеркнута роль МБС при порождении субъектов из объектов (рис. 6.3).

Взаимодействие субъектов и объектов при порождении потоков уточнено введением ассоциированных с субъектом объектов. ОУ содержит информацию о разрешенных значениях отображения  $Stream()$  (об элементах множества  $L$  или  $N$ ) и  $Create()$  (элементы множества  $E$ ). ОУ может быть ассоциирован как с МБО, так и с МБС.

Опираясь на теорему 6.3 (базовую теорему ИПС), опишем метод субъектно-объектного взаимодействия в рамках ИПС для наиболее распространенной архитектуры компьютерной системы.



Рис. 6.2. Классическая схема ядра безопасности

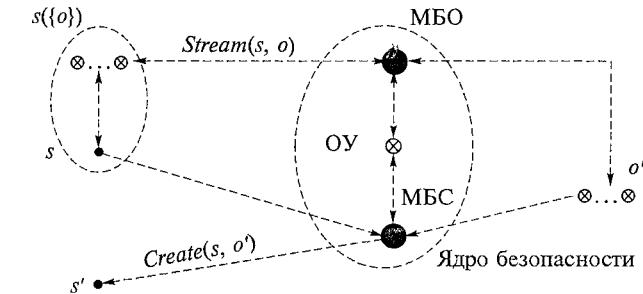


Рис. 6.3. Ядро безопасности с учетом контроля порождения субъектов

Из теоремы 6.3 следует, что для создания гарантированно защищенной компьютерной системы (в смысле выполнения заданной политики безопасности) необходимо выполнить следующие действия.

1. Убедиться в попарной корректности субъектов, замыкаемых в ИПС (либо убедиться в корректности любого субъекта относительно МБО и МБС).

2. Спроектировать и реализовать программно (или програмно-аппаратно) МБС так, чтобы:

- для любого субъекта и любого объекта производился контроль порождения субъектов, т. е. чтобы реализация МБС соответствовала его определению;
- порождение любого субъекта происходило с контролем неизменности объекта-источника.

3. Реализовать МБО в рамках априорно сформулированной политики безопасности.

Приведенные выше утверждения верны только тогда, когда описанная и реализованная политика безопасности не нарушает их условий (проверка данного факта зависит от модели политики безопасности и является отдельной весьма важной задачей).

Кроме того, необходимо обратить внимание на следующее. ОУ, который является ассоциированным объектом МБС (обычно ассоциированный объект-данные), играет решающую роль в проектировании ИПС. При возможности изменения состояния ОУ потенциально возможно «размыкание» программной среды, т. е. добавление к множеству разрешенных субъектов дополнительных, реализующих функции нарушителя. С другой стороны, процесс управления безопасностью подразумевает возможность изменения объекта управления. Возможность изменения объекта управления (реализация потока  $Stream(\text{субъект управления}, \text{ассоциированный объект субъекта управления}) \rightarrow \text{объект управления}$ ) должна присутствовать для выделенных субъектов (возможно с дополнительным условием активизации этого субъекта выделенным пользователем или пользователями).

Важную роль при проектировании ИПС играет свойство большинства компьютерных систем, заключающееся в поэтапной активизации субъектов из объектов различного уровня представления информации.

В общем случае можно говорить о рекурсивной структуре объектов некоторого уровня, вмещающей в себя объекты предыдущего уровня. С учетом иерархической структуры представления объектов можно говорить о том, что в начальные этапы активизации компьютерной системы декомпозиция на субъекты и объекты динамически изменяется. Следовательно, основная теорема ИПС может быть применима только на отдельных интервалах времени, когда уровень представления объектов постоянен и декомпозиция фиксирована. Можно утверждать, что ИПС, действующую от момента активизации до момента окончания работы компьютерной системы, невозможно сформировать в начальный момент активизации.

Пусть в компьютерной системе выделяется конечное число уровней представления объектов  $U = \{0, \dots, R\}$ , где  $R$  — максимальный уровень представления объекта.

С точки зрения выполнения условий теоремы 6.3 имеет смысл говорить о некотором «стационарном» состоянии компьютерной системы, когда в отображениях *Stream()* и *Create()* участвуют только объекты уровня  $R$ . Тогда реализация МБС может быть значительно упрощена (в том смысле, что все аргументы-объекты операции *Create()* имеют уровень  $R$ ).

Практическая реализация всех компьютерных систем позволяет выделить две фазы их работы: активизация субъектов с ростом уровня представления объектов (фаза загрузки или начальная фаза) и фаза стационарного состояния (когда уровень представления объектов не увеличивается).

Тогда практическая реализация ИПС может состоять из двух этапов.

1. Предопределенное выполнение начальной фазы, включающее в себя момент активизации МБС (и МБО).

2. Работа в стационарной фазе в режиме ИПС (возможно, с контролем неизменности объектов-источников).

Введем понятие последовательности активизации компонент компьютерной системы. Смысл вводимых понятий и формулируемых ниже утверждений состоит в необходимости приведения субъектов в одно и то же состояние после активизации первичного субъекта аппаратно-программного уровня или в задании предопределенной последовательности активизации субъектов компьютерной системы.

Используем следующие обозначения:

$Z_L$  — последовательность пар  $(i, j)$ , длины  $L$  таких, что  $Create(s_i, o_j)[t] \rightarrow s_m[t+1]$ , где  $t = 0, 1, 2, \dots, L-1$  — моменты времени;

$S_Z$  — множество всех субъектов, включенных в последовательность  $Z_L$ ;

$O_Z$  — множество всех объектов, включенных в последовательность  $Z_L$ .

Для многопоточных компьютерных систем можно рассматривать несколько (возможно, зависимых друг от друга) последовательностей  $Z_L$  и соответственно множеств  $S_Z$  и  $O_Z$ .

**Определение 6.17.** Состоянием компьютерной системы в момент времени  $t$  называется упорядоченная совокупность состояний субъектов.

**Теорема 6.4 (условие одинакового состояния).** Состояние компьютерной системы в моменты времени  $t_x^1$  и  $t_x^2$  ( $t_x^1$  и  $t_x^2$  исчисляются для двух отрезков активности компьютерной системы от нулевого момента активизации  $t_0^1$  и  $t_0^2$ , например включения питания аппаратной части) одинаково, если:

- $t_x^1 = t_x^2$ ;
- тождественны все субъекты  $s[t_0^1]$  и  $s[t_0^2]$ ;
- неизменны все объекты из множества  $O_Z$ ;
- неизменна последовательность  $Z_L$ .

*Доказательство.* Используем метод математической индукции.

Верность утверждения при  $t = 0$  следует из определения тождественности субъектов.

Пусть утверждение верно для  $t < k$ , где  $k > 0$ .

Тогда в момент времени  $k$  могут быть порождены только тождественные субъекты, поскольку тождественны активизирующие субъекты (по предположению индукции) и по условию утверждения неизменны элементы множества  $O_Z$ . Теорема доказана.

Длина  $L$  последовательности  $Z_L$  определяется по признакам:

- невозможности управления субъектами, принадлежащими множеству  $S_Z$  со стороны пользователя (в противном случае последовательность активизации субъектов может быть изменена);
- доступности для контроля неизменности всех объектов из множества  $O_Z$ ;
- невозрастания уровня представления информации (в данном случае имеется в виду, что существует такой момент времени  $t_x$ , что для любого  $t > t_x$  объект-аргумент  $o$  операции  $Stream(s, o)[t]$  принадлежит одному уровню представления).

Пусть в последовательности  $Z_L$  можно выделить  $z_i$  такое, что для любого  $z_n$ ,  $t > i$ , отображений *Create()* и *Stream()* используют только объекты уровня  $R$ . Другими словами, с момента времени  $i$  наступает стационарная фаза функционирования компьютерной системы.

В этих условиях, а также при попарной корректности субъектов и действии МБС с контролем неизменности объектов-источников на уровне  $R$  с момента времени  $L > i$  верна следующая теорема.

**Теорема 6.5 (достаточное условие ИПС при ступенчатой загрузке).** При условии неизменности  $Z_L$  и неизменности объектов из  $O_Z$  в компьютерной системе с момента времени установления неизменности  $Z_L$  и  $O_Z$  действует ИПС.

**Доказательство.** Необходимо заметить, что все условия теоремы 6.5 соответствуют условиям теоремы 6.4. Уточнения касаются структуры последовательности  $Z_L$ .

Согласно теореме 6.4 с момента времени  $t = 0$  до момента времени  $t = i$  действует изолированная (в рамках)  $S_Z$  программная среда.

Для доказательства утверждения необходимо убедиться в том, что:

- 1) МБС в момент времени  $t = i$  гарантированно активизируется;
- 2) в любой момент  $t > i$  программная среда изолирована.

Первое следует из теоремы 6.4 (при  $t = i$  состояние программной среды всегда будет одинаково, следовательно, всегда будет активирован субъект МБС). Второе следует из определения МБС и условия теоремы.

С момента времени  $t = 0$  до момента времени  $i$  программная среда изолирована, с момента времени  $t > i$  программная среда также изолирована, следовательно, компьютерная система изолирована при любом  $t \geq 0$ . Теорема доказана.

### Контрольные вопросы

1. Каковы основные функции МБО и МБС в ИПС, в чем их отличие друг от друга?

2. Почему для реализации ИПС необходимо требовать наличия в компьютерной системе контроля порождения субъектов?

3. В чем отличие структуры ядра безопасности в классических моделях безопасности компьютерных систем от структуры ядра безопасности в субъектно-ориентированной модели ИПС?

4. Рассмотрите на примере реальных компьютерных систем пути реализации в них требований субъектно-ориентированной модели ИПС при ступенчатой загрузке.

## ГЛАВА 7

# ПРОБЛЕМЫ ПРИМЕНЕНИЯ МОДЕЛЕЙ БЕЗОПАСНОСТИ ПРИ ПОСТРОЕНИИ ЗАЩИЩЕННЫХ КОМПЬЮТЕРНЫХ СИСТЕМ

## 7.1. ПРОБЛЕМА АДЕКВАТНОСТИ РЕАЛИЗАЦИИ МОДЕЛИ БЕЗОПАСНОСТИ В РЕАЛЬНОЙ КОМПЬЮТЕРНОЙ СИСТЕМЕ

### 7.1.1. Общая постановка задачи построения системы защиты

Рассмотрим в общем случае проблему построения системы защиты.

Используем следующие обозначения:

$t = 0, 1, 2, \dots$  — время;

$R$  — множество видов доступа;

$O_t$  — множество объектов в момент времени  $t$ ;

$S_t \subset O_t$  — множество субъектов в момент времени  $t$ ;

$G_t = (O_t, E_t)$  — граф текущих доступов в системе в момент времени  $t$ , где  $O_t$  — вершины графа,  $E_t \subset O_t \times O_t \times R$  — множество ребер (каждое ребро соответствует текущему доступу в системе в момент времени  $t$ );

$G_0$  — граф текущих доступов в начальном состоянии системы;

$G^*$  — множество всех последовательностей графов текущих доступов (каждая последовательность соответствует заданной траектории функционирования системы).

Согласно основной аксиоме 1.2 (см. гл. 1) все вопросы безопасности информации описываются доступами субъектов к объектам, следовательно, в общем случае свойства системы защиты информации могут быть определены на основе описания свойств графов последовательностей из  $G^*$ . Среди всех последовательностей из  $G^*$  согласно требованиям политики безопасности априорно выделяются два подмножества последовательностей:  $L^*$  — подмножество разрешенных траекторий;  $N^*$  — подмножество запрещенных траекторий ( $G^* = L^* \cup N^*$ ,  $L^* \cap N^* = \emptyset$ ).

Задача системы защиты состоит в том, чтобы любая реальная траектория функционирования системы не попала в множество запрещенных траекторий  $N^*$ . В то же время сама система защиты является частью компьютерной системы. Следовательно, в общем случае для успешного решения данной задачи система защиты, обладающая конечными ресурсами, должна в каждый момент времени хранить и анализировать информацию обо всей предыстории функционирования компьютерной системы, а также предсказывать будущее, что является алгоритмически неразрешимой задачей.

Таким образом, необходимо определять требования политики безопасности компьютерной системы так, чтобы стала возможной реализация системы защиты, соответствующей данным требованиям.

### 7.1.2. Гомоморфизм компьютерной системы и ее математической модели безопасности

Требования политики безопасности в большинстве случаев определяются с использованием математических моделей безопасности компьютерных систем. Как уже отмечалось, большинство таких моделей построены с использованием понятия детерминированного конечного автомата. При этом реальная компьютерная система строится таким образом, чтобы математическая модель, соответствующая заданной политике безопасности, являлась гомоморфным отображением реальной системы.

Рассмотрим этот вопрос подробнее. Используем следующие обозначения:

$V$  — множество состояний компьютерной системы;

$G$  — множество состояний автомата, построенного в соответствии с требованиями математической модели;

$\alpha_1, \dots, \alpha_n$  — множество алгоритмов, реализуемых в компьютерной системе; при этом каждый алгоритм переводит компьютерную систему из текущего состояния в последующее:  $v \xrightarrow{\alpha_i} v^*$ , где  $v, v^* \in V, i \in 1, \dots, n$ . Например, могут рассматриваться алгоритмы чтения файла, изменения уровня доступа пользователя, установления защищенного соединения;

$\beta_1, \dots, \beta_n$  — множество операций математической модели; при этом каждая операция переводит автомат из текущего состояния в последующее:  $g \xrightarrow{\beta_i} g^*$ , где  $g, g^* \in G, i \in 1, \dots, n$ .

Для того чтобы компьютерная система соответствовала математической модели, должен существовать гомоморфизм  $\sigma: V \rightarrow G$  такой, что для любых  $v, v^* \in V$  и  $i \in 1, \dots, n$  выполняется условие

$(v \xrightarrow{\alpha_i} v^*) \Rightarrow (\sigma(v) \xrightarrow{\beta_i} \sigma(v^*))$ ,  
т.е. для каждого  $i \in 1, \dots, n$  коммутативна диаграмма, представленная на рис. 7.1.

Построение гомоморфизма реальной компьютерной системы в математическую модель, соответствующую требованиям политики безопасности, на практике нередко является труднореализуемой задачей. Кроме недостатков самих математи-

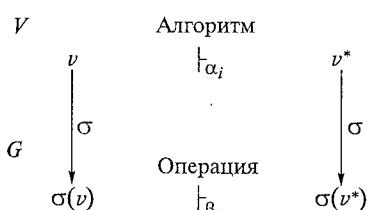


Рис. 7.1. Гомоморфизм компьютерной системы и ее математической модели безопасности

ческих моделей безопасности это связано с тем, что модели в большинстве случаев содержат жесткие требования к определению порядка функционирования систем защиты, что не позволяет непосредственно реализовать их в реальной системе. Поэтому отдельные элементы компьютерной системы могут не удовлетворять требованиям математической модели, например доверенные субъекты в модели Белла — ЛАПадула. Кроме реализации основной математической модели должны разрабатываться модели для определения порядка функционирования элементов компьютерной системы, не соответствующих требованиям основной модели. Должны быть согласованы между собой требования всех использованных математических моделей, что также может являться сложной задачей.

Таким образом, при реализации реальной компьютерной системы актуальна проблема ее адекватности математической модели безопасности. Суть этой проблемы состоит в том, что необходимо дать ответ на вопрос: в какой степени ряд допущений, сделанных при построении на основе математической модели реальной компьютерной системы, скажется на соответствии компьютерной системы требованиям политики безопасности? Если оставить без ответа данный вопрос, то может оказаться, что компьютерная система, формально признанная безопасной, не будет реально таковой являться.

С другой стороны, не менее сложным является вопрос об адекватности математической модели безопасности условиям функционирования и требованиям безопасности компьютерной системы. В какой степени математическая модель может обеспечить выполнение требований политики безопасности? От ответа на данный вопрос также зависит безопасность компьютерной системы и эффективность использования ее ресурсов, выделяемых на реализацию системы защиты.

## 7.2. ПРОБЛЕМЫ РЕАЛИЗАЦИИ ПОЛИТИКИ БЕЗОПАСНОСТИ

### 7.2.1. Реализация дискреционной политики безопасности

При реализации в реальных системах дискреционной политики безопасности наиболее существенной является проблема отсутствия в большинстве математических моделей четких правил разграничения доступа, которые должны быть реализованы в реальной системе. По этой причине на объекты, имеющие важное значение в обеспечении безопасности системы, могут быть некорректно назначены права доступа субъектов системы.

Например, ОС семейства *Windows NT* сертифицирована в соответствии с требованиями класса защиты *F-C2* и уровня адек-

	$o_1$	$o_2$
$u_1$	$own, r, w$	$w, exe$
$u_2$		$own, r, w$

Рис. 7.2. Заполнение матрицы доступов в примере 7.1

безопасности *E3* по *ITSEC* [15], а ОС семейства *Windows 2000* сертифицирована в соответствии с требованиями уровня адекватности *EAL4* и профиля защиты *Controlled Access Protection Profile Version 1.d «Общих критериев»* [10]. В то же время настройки, правила конфигурирования и администрирования ОС в соответствии с требованиями указанных критериев не устанавливаются по умолчанию при инсталляции. Таким образом, пользователи ОС должны самостоятельно выполнить дополнительную настройку ОС, что представляет угрозу ошибки или неправильной интерпретации требований критериев. При этом, как показано в подразд. 7.3, непосредственной реализации требований критериев недостаточно для обеспечения соответствующего уровня гарантий безопасности.

Кроме указанного недостатка компьютерные системы, реализующие дискреционную политику безопасности, в общем случае не обеспечивают защиты от атак с использованием программных закладок вида «тロянский конь».

**Пример 7.1.** Реализация атаки с использованием программных закладок вида «тロянский конь».

Используем следующие обозначения:

$u_1$  — пользователь;  $u_2$  — нарушитель;

$o_1$  — ценный объект;  $o_2$  — объект, содержащий закладку вида «тロянский конь».

Пусть матрица доступов имеет заполнение, представленное на рис. 7.2.

Атака реализуется по сценарию, который представлен на рис. 7.3.

Нарушитель подбрасывает закладку пользователю. Пользователь, запуская закладку, дает ей возможность от своего имени переписывать информацию из ценного объекта в себя, после чего нарушитель читает ценную информацию в доступном ему объекте-закладке.

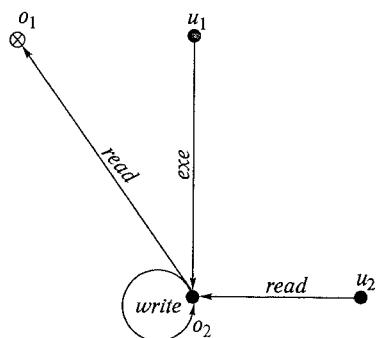


Рис. 7.3. Реализация атаки с использованием программной закладки вида «тロянский конь»

## 7.2.2. Реализация мандатной политики безопасности

Задачи, решаемые при создании компьютерных систем, реализующих мандатную политику безопасности, можно классифицировать по трем основным группам.

1. Обеспечение выполнения требований пометки субъектов и объектов (каждому субъекту должен быть присвоен уровень доступа, каждому объекту должен быть присвоен уровень конфиденциальности) и правил мандатного разграничения доступа.

2. Определение порядка функционирования доверенных субъектов компьютерной системы.

3. Обеспечение безопасности информационных потоков.

Решение первой задачи состоит, в первую очередь, в корректной реализации математической модели системы мандатного разграничения доступа и надежного механизма присваивания меток субъектам и объектам доступа.

При решении второй задачи, как правило, используют известные математические модели (например, дискреционного или ролевого разграничения доступа), причем только для описания порядка действий доверенных субъектов, которые выходят за рамки правил мандатного разграничения доступа.

Основной проблемой при реализации мандатной политики безопасности является решение третьей задачи — обеспечение безопасности информационных потоков. Это связано с тем, что крайне сложно в современных компьютерных системах выявить возможные неблагоприятные информационные потоки по памяти и, особенно, по времени. Кроме того, разработчиками систем мандатного разграничения доступа нередко упускается из виду, что система защиты должна не только препятствовать доступу к объектам с высоким уровнем конфиденциальности субъектов, не имеющих на это прав. Она должна также обеспечивать защиту от возникновения информационных потоков от объектов с большим уровнем конфиденциальности к объектам с меньшим уровнем конфиденциальности, реализуемых с использованием кооперации субъектов, имеющих доступ к конфиденциальной информации.

Рассмотрим ряд примеров программ [19], представленных на некотором абстрактном языке программирования высокого уровня, иллюстрирующих трудности практической реализации механизма контроля информационных потоков в системах мандатного разграничения доступа.

**Пример 7.2.** Неблагоприятные информационные потоки по памяти с использованием локальных и логических переменных.

Используем следующие обозначения:

$f_1$  — конфиденциальный файл (уровень конфиденциальности *High*), который может содержать запись «*A*» или запись «*B*»;

$f_2$  — неконфиденциальный файл (уровень конфиденциальности  $Low < High$ ).

Приведенные ниже процедуры реализуют неблагоприятные информационные потоки по памяти через локальную переменную (процедура  $p_1$ ) и логическую переменную (процедура  $p_2$ ).

*Procedure  $p_1(f_1: file, f_2: file)$*

```
Open  $f_1$  for read;  
Read A from  $f_1$ ;  
Close  $f_1$ ;  
Open  $f_2$  for write;  
Write A to  $f_2$ ;  
Close  $f_2$ ;
```

*End.*

*Procedure  $p_2(f_1: file, f_2: file)$*

```
Open  $f_1$  for read  
If ( $f_1 = \langle\langle A\rangle\rangle$ ) Then  
    Close  $f_1$ ;  
    Open  $f_2$  for write;  
    Write  $\langle\langle A\rangle\rangle$  to  $f_2$ ;  
Else  
    Close  $f_1$ ;  
    Open  $f_2$  for write;  
    Write  $\langle\langle B\rangle\rangle$  to  $f_2$ ;  
End If,  
Close  $f_2$ ;
```

*End.*

Анализируя процедуры  $p_1$  и  $p_2$ , можно предложить простой способ предотвращения реализуемых ими неблагоприятных информационных потоков. Процесс, единожды получивший доступ на чтение к файлу с некоторым уровнем конфиденциальности, не должен получать доступ на запись к файлам с более низким уровнем конфиденциальности. Такое решение может быть применено для пользовательских процессов в ОС. Однако оно не может быть признано универсальным, так как системные процессы, например монитор ссылок, при мандатном разграничении доступа должны одновременно оперировать с данными различных уровней конфиденциальности.

Другой путь — инициализация каждой переменной процесса как объекта доступа — также не может являться оптимальным.

Для предотвращения неблагоприятных информационных потоков через локальные или логические переменные при написании программ в тех случаях, когда это возможно, необходимо:

- открывать все файлы, необходимые для работы программы в начале ее выполнения;
- закрывать все файлы в конце выполнения программы;

• непосредственную обработку информации из открытых файлов осуществлять во внутренних процедурах, использующих только локальные переменные.

**Пример 7.3.** Реализация неблагоприятного информационного потока по времени.

Используем следующие обозначения:

$f_1$  — конфиденциальный файл (уровень конфиденциальности  $High$ ), который может содержать запись « $A$ » или запись « $B$ »;

$f_2$  — неконфиденциальный файл (уровень конфиденциальности  $Low$ );

$s_1$  — субъект, работающий по программе:

*Process  $s_1(f_1: file)$*

```
Open  $f_1$  for read;  
While  $f_1 = \langle\langle A\rangle\rangle$  Do End;  
Close  $f_1$ ;
```

*End.*

$s_2$  — субъект-нарушитель, работающий по программе:

*Process  $s_2(f_1: file, f_2: file)$*

```
Start  $s_1(f_1)$ ;  
Wait 10 seconds;  
Open  $f_2$  for write;  
If (stop  $s_1$ ) Then  
    Write  $\langle\langle B\rangle\rangle$  to  $f_2$ ;  
Else
```

```
    Write  $\langle\langle A\rangle\rangle$  to  $f_2$ ;
```

*End If;*

*Close  $f_2$ ;*

*End.*

Предполагается, что выполняются условия:

$f_o(f_1) = High$ ;  
 $f_o(f_2) = Low$ ;  
 $f_s(s_1) = f_s(s_2) = High$ ;  
 $f_c(s_1) = High$ ;  
 $f_c(s_2) = Low$ .

Субъект-нарушитель  $s_2$ , не открывая на чтение конфиденциальных файлов, активизирует от своего имени субъект  $s_1$  с высоким текущим уровнем доступа и в зависимости от результатов его работы (субъект  $s_1$  либо сразу завершит работу, либо «зависнет») записывает информацию в файл с низким уровнем конфиденциальности (рис. 7.4).

Для предотвращения неблагоприятных информационных потоков по времени данного вида можно потребовать запрета получения процессами-субъектами низкого уровня доступа к ин-

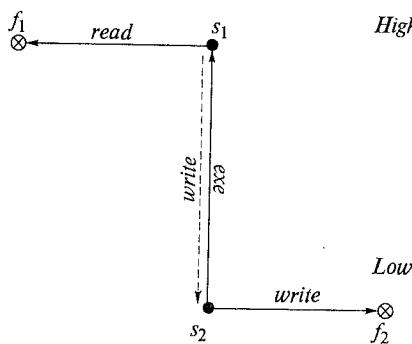


Рис. 7.4. Пример реализации информационного потока по времени

ми его использования. Например, если процесс открывает файл на чтение, то он может разрешить или запретить другим процессам одновременное чтение этого файла. Системная переменная ОС, используемая для определения условий совместного использования файла, может быть использована для реализации неблагоприятных информационных потоков по времени.

Анализируя рассмотренные примеры, целесообразно сделать следующие выводы. Без дополнительных уточнений свойств политики безопасности, порядка написания кода программ, определения порядка конфигурирования и администрирования невозможна реализация компьютерных систем с мандатным разграничением доступа. В то же время, слишком строгая политика безопасности может привести к трудностям или даже невозможности практического использования системы защиты. Кроме того, как уже отмечалось, доопределение и усложнение свойств безопасности также несет в себе угрозу ошибки и, как следствие, угрозу неадекватности реализации математической модели и политики безопасности в компьютерной системе.

### 7.3. ОБОСНОВАНИЕ ПОЛИТИКИ БЕЗОПАСНОГО АДМИНИСТРИРОВАНИЯ ОС СЕМЕЙСТВ *WINDOWS NT/2000/XP*

#### 7.3.1. Общие положения

Рассмотрим пример решения проблемы применения математической модели при построении и обосновании свойств реальной системы защиты. В примере мандатная политика целостности Биба [11] использована для обоснования политики безопасного администрирования компьютерных систем, построенных на основе ОС семейств *Windows NT/2000/XP*.

формации о результатах работы процессов-субъектов высокого уровня доступа. В то же время такое решение не всегда возможно на практике. Кроме того, возможны иные способы реализации неблагоприятных информационных потоков по времени, для которых это решение неэффективно. Так, большинство современных ОС позволяют при открытии процессом файла определять параметры совместного с другими процессами его использования. Например, если процесс открывает файл на чтение, то он может разрешить или запретить другим процессам одновременное чтение этого файла. Системная переменная ОС, используемая для определения условий совместного использования файла, может быть использована для реализации неблагоприятных информационных потоков по времени.

Анализируя рассмотренные примеры, целесообразно сделать следующие выводы. Без дополнительных уточнений свойств политики безопасности, порядка написания кода программ, определения порядка конфигурирования и администрирования невозможна реализация компьютерных систем с мандатным разграничением доступа. В то же время, слишком строгая политика безопасности может привести к трудностям или даже невозможности практического использования системы защиты. Кроме того, как уже отмечалось, доопределение и усложнение свойств безопасности также несет в себе угрозу ошибки и, как следствие, угрозу неадекватности реализации математической модели и политики безопасности в компьютерной системе.

В политике безопасного администрирования рассматриваются две угрозы безопасности компьютерной системы.

*Угроза 1.* Наличие у пользователя возможности разместить ресурсы на компьютере (запустить процесс или разместить файловые данные) несет угрозу получения им всех привилегий на данном компьютере.

*Угроза 2.* Обращение пользователя к компьютеру (с целью получения данных локально или по сетевым коммуникационным каналам) несет угрозу захвата его привилегий пользователем, разместившим свои ресурсы на этом компьютере.

Отметим, что в политике не рассматриваются процессы, создаваемые ОС для выполнения команд пользователя по некоторым системным предопределенным интерфейсам от имени данного пользователя. Например, при обращении пользователя к созданному ОС именованному коммуникационному каналу (*pipe*) создается процесс от имени этого пользователя. Данный процесс представляет предопределенный интерфейс взаимодействия с пользователем.

Будем считать, что нарушитель имеет два уровня возможностей в системе:

- если нарушитель может разместить на компьютере свой ресурс, то он имеет возможность управления функциями автоматизированной системы (нарушитель 3-го уровня по Руководящим документам Гостехкомиссии России [1]);
- если нарушитель может только удаленно обращаться к компьютеру, то он имеет возможность ведения диалога с автоматизированной системой (нарушитель 1-го уровня по Руководящим документам Гостехкомиссии России).

Компьютерная система удовлетворяет требованиям безопасного администрирования тогда и только тогда, когда выполняется следующий основной постулат: получение нарушителем полного контроля над компьютером системы не должно нести угрозы безопасности для других компьютеров системы посредством реализации нарушителем угроз 1 и 2.

Цель политики — представить эффективно проверяемые и практически реализуемые условия выполнения в компьютерной системе требований безопасного администрирования. Таким образом, реализация политики должна обеспечить защиту или сокращение ущерба от реализации нарушителем угроз 1 и 2.

#### 7.3.2. Математическая модель политики безопасного администрирования

Положения политики безопасного администрирования обосновываются с использованием математической модели.

В рассматриваемой математической модели существенную роль при обосновании положений политики играет определенное в модели отношение подчиненности компьютеров системы, которое аналогично понятию отношения доверия между доменами в ОС семейства *Windows NT/2000/XP*.

Введем следующие обозначения:

$t = 0, 1, 2, \dots$  — время;

$2^A$  — множество всех подмножеств некоторого множества  $A$ ;

$A'_t$  — множество  $A$  в момент времени  $t$ ;

$F()'$  — значение некоторой функции  $F()$  в момент времени  $t$ ;

$A'_p$  — множество  $A$  в момент времени  $t$  на траектории функционирования компьютерной системы  $p$ ;

$F()_p'$  — значение некоторой функции  $F()$  в момент времени  $t$  на траектории функционирования компьютерной системы  $p$ ;

$P$  — множество траекторий функционирования компьютерной системы;

$C$  — множество компьютеров системы;

$DC \subset C$  — множество контроллеров домена (отождествим контроллеры домена и вспомогательные контроллеры одного домена), рабочих станций членов рабочих групп;

$WS \subset C$  — множество рабочих станций и серверов, являющихся членами домена; при этом  $DC \cup WS = C$ ,  $DC \cap WS = \emptyset$ ;

$U$  — множество пользователей компьютерной системы;

$os_c$  — «пользователь ОС» компьютера  $c \in C$ ;

$R_c$  — множество привилегий и прав пользователей на компьютере  $c \in C$ . Например: {«Access this computer from network», «Debug programs», «Load and unload device drivers», «право на запись в файл SAM реестра ресурсов», «право на чтение файла ntoskrnl.exe»}  $\subset R_c$ .

**Исходное предположение 1.** Пусть множества  $C$ ,  $DC$ ,  $WS$ ,  $U$  и  $R_c$  для  $c \in C$  не изменяются со временем.

Данное условие не ограничивает общности последующих положений, так как можно считать, что все пользователи, привилегии пользователей, компьютеры и роли компьютеров, когда-либо имевшиеся в компьютерной системе, были определены в ней изначально. Если в некотором выражении для значения функции или множества не указан момент времени, то будем считать, что найдется такой момент времени, в котором данное выражение будет выполняться.

Используем следующие обозначения:

$write: U \rightarrow 2^C$  — функция, определяющая для каждого пользователя множество компьютеров, на которых он может разместить свои ресурсы (файловые данные или процесс от своего имени);

$read: U \rightarrow 2^C$  — функция, определяющая для каждого пользователя множество компьютеров, к которым он обращается с целью получения данных локально или по сетевым коммуникационным каналам;

$right: U \times C \rightarrow 2^{R_c}$  — функция, определяющая множество прав и привилегий пользователя на компьютере.

**Исходное предположение 2.** В начальный момент времени ( $t=0$ ) пользователи не размещают свои ресурсы на компьютерах и не обращаются к компьютерам, т.е. для каждого  $u \in U$  выполняются следующие условия:

$$write(u)^0 = \emptyset; read(u)^0 = \emptyset.$$

**Исходное предположение 3.** Пользователь  $os_c$  компьютера  $c \in C$  обладает всеми привилегиями  $R_c$  на данном компьютере, т.е. для  $c \in C$  справедливо равенство

$$right(os_c, c) = R_c.$$

**Исходное предположение 4.** Если у пользователя имеются некоторые особые права или привилегии на данном компьютере, например «Debug programs», «Load and unload device drivers», или «право на запись в файл SAM реестра ресурсов», то он может получить все привилегии на данном компьютере, т.е. верна импликация

$$\{«Debug programs», «Load and unload device drivers», «право на запись в файл SAM реестра ресурсов»\} \cap right(u, c) \neq \emptyset \Rightarrow (right(u, c) = R_c).$$

**Определение 7.1.** Пользователей  $u \in U$  компьютера  $c \in C$ , для которых в момент времени  $t=0$  справедливо равенство  $right(u, c) = R_c$ , назовем доверенными пользователями данного компьютера. Множество доверенных пользователей компьютера  $c \in C$  обозначим через  $U_c$ . Пользователей из  $U \setminus U_c$  назовем недоверенными пользователями компьютера  $c \in C$ .

**Определение 7.2.** Пусть  $c_1, c_2 \in C$ . Будем говорить, что компьютер  $c_2$  непосредственно подчинен компьютеру  $c_1$ , если выполняется одно из следующих условий:

- $c_1 = c_2$ ;
- $c_1$  — контроллер домена,  $c_2$  — рабочая станция этого домена;
- $c_1$  — контроллер первого домена,  $c_2$  — контроллер второго домена, который доверяет первому.

**Определение 7.3.**  $G(C, E)$  — ориентированный граф подчиненности компьютеров в компьютерной системе, где  $C$  — множество вершин;  $E \subseteq C \times C$  — множество ребер графа подчиненности. Полагаем, что  $(c_1, c_2) \in E$  тогда и только тогда, когда компьютер  $c_2$  непосредственно подчинен компьютеру  $c_1$ . Обозначим в графе: • — вершины из множества  $DC$ ;  $\otimes$  — вершины из множества  $WS$ .

**Определение 7.4.** Пусть  $c_1, c_2 \in C$  и  $G(C, E)$  — граф подчиненности компьютеров. Компьютер  $c_2$  подчинен компьютеру  $c_1$  ( $c_1 \rightarrow \rightarrow c_2$ ) тогда и только тогда, когда в графе  $G(C, E)$  существует ориентированный путь от  $c_1$  к  $c_2$ .

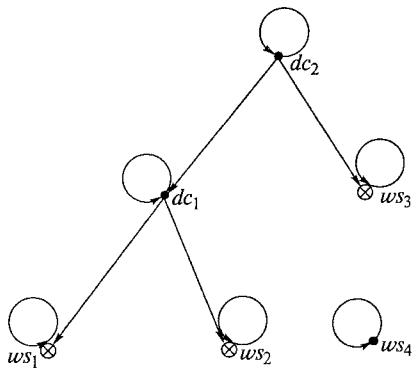


Рис. 7.5. Пример графа  $G(C, E)$  подчиненности компьютеров

**Пример 7.4.** Пусть  $dc_1$  — контроллер первого домена;  
 $ws_1, ws_2$  — рабочие станции первого домена;  
 $dc_2$  — контроллер второго домена, которому доверяет первый домен;  
 $ws_3$  — рабочая станция второго домена;  
 $ws_4$  — рабочая станция — член рабочей группы (рис. 7.5).

Тогда  $\{(dc_1, ws_1), (dc_1, ws_2), (dc_2, ws_3), (dc_2, dc_1)\} \subseteq E$ .

**Замечание 7.1.** Если компьютер

$c \in WS$ , то в графе  $G(C, E)$  ему соответствует вершина, в которую входит только одно ребро и из которого не выходит ни одного ребра в другие вершины, отличные от данной.

**Замечание 7.2.** Если компьютер является рабочей станцией и членом рабочей группы, то в графе  $G(C, E)$  ему соответствует изолированная вершина.

**Исходное предположение 5.** Наличие у пользователя всех привилегий на компьютере означает наличие у пользователя всех привилегий на компьютерах, подчиненных данному, т. е. для всех  $u \in U, c_1, c_2 \in C$  таких, что  $c_1 \rightarrow c_2$ , выполняются условия:

$$(right(u, c_1) = R_{c_1}) \Rightarrow (right(u, c_2) = R_{c_2}); \\ (u \in U_{c_1}) \Rightarrow (u \in U_{c_2}).$$

С использованием введенных обозначений запись рассматриваемых в рамках политики безопасного администрирования угроз 1 и 2 имеет следующий вид.

**Угроза 1.** Наличие у пользователя возможности разместить ресурсы на компьютере несет угрозу получения им всех привилегий на данном компьютере, т. е. для всех  $t_1 > 0, u \in U, c \in C$  выполняется условие

$$(c \in write(u)^{t_1}, right(u, c)^{t_1} = R_c) \Rightarrow (\text{существует } t_2 > t_1: right(u, c)^{t_2} = R_c).$$

**Угроза 2.** Обращение пользователя к компьютеру несет угрозу захвата его привилегий пользователем, разместившим свои ресурсы на этом компьютере, т. е. для всех  $t_1 > 0, u_1, u_2 \in U, c_1, c_2 \in C$  выполняется условие

$$(c_1 \in read(u_1)^{t_1}: c_1 \in write(u_2)^{t_1}, right(u_1, c_2)^{t_1} \subsetneq right(u_2, c_2)^{t_1}) \Rightarrow \\ \Rightarrow (\text{существует } t_2 > t_1: right(u_1, c_2)^{t_2} \subset right(u_2, c_2)^{t_2}).$$

**Замечание 7.3.** Угроза 2 предполагает, что все пользователи компьютерной системы активно работают и не все пользователи являются доверенными.

**Исходное предположение 6.** Пусть для  $t_1 > 0, u_1 \in U, c_1, c_2 \in C$  выполняется условие  $c_1 \in read(u_1)^{t_1}$ ,  $right(u_1, c_2)^{t_1} \neq \emptyset$  и не выполняется условие  $c_1 \rightarrow c_2$ . Тогда всегда найдется  $u_2 \in U$  такой, что  $c_1 \in write(u_2)^{t_1}$  и  $right(u_1, c_2)^{t_1} \subsetneq right(u_2, c_2)^{t_1}$ .

**Замечание 7.4.** Исходное предположение 6 является существенным дополнением к определению угрозы 2, необходимым для обеспечения соответствия этого определения условиям функционирования реальных компьютерных систем. Оно, в том числе, показывает, что всегда найдется не являющийся доверенным пользователь рабочей станции домена, который может разместить на ней свой ресурс.

Формальная запись основного постулата политики безопасности администрирования дается в следующем определении.

**Определение 7.5.** С использованием введенных обозначений основной постулат политики безопасного администрирования будет иметь следующий вид: компьютерная система, построенная на основе ОС семейств *Windows NT/2000/XP*, удовлетворяет требованиям безопасного администрирования тогда и только тогда, когда выполняются условия 1 и 2.

**Условие 1.** Для  $p_1 \in P, t_2 > t_1 > 0, u \in U, c_1, c_2 \in C$  таких, что  $c_1 \neq c_2$ , истинно:

$$(c_1 \in write(u)^{t_1}, right(u, c_2)^{t_2-1} \neq R_{c_2}, right(u, c_2)^{t_2} = R_{c_2}) \Rightarrow \\ \Rightarrow (\text{существует } p_2 \in P: right(u, c_2)^{t_2-2} \neq R_{c_2}, right(u, c_2)^{t_2-1} = R_{c_2}).$$

**Условие 2.** Для  $p_1 \in P, t_2 > t_1 > 0, u_1, u_2 \in U, c_1, c_2 \in C$  таких, что  $u_1 \neq u_2$ , истинно:

$$(c_1 \in read(u_1)^{t_1}, right(u_2, c_2)^{t_2-1} \neq R_{c_2}, right(u_2, c_2)^{t_2} = R_{c_2}) \Rightarrow \\ \Rightarrow (\text{существует } p_2 \in P: right(u_2, c_2)^{t_2-2} \neq R_{c_2}, right(u_2, c_2)^{t_2-1} = R_{c_2}).$$

**Замечание 7.5.** В определении основного постулата предполагается, что размещение пользователем ресурсов на компьютере или обращение его к компьютеру осуществляется не менее чем за один шаг функционирования компьютерной системы.

**Замечание 7.6.** Условие 1 требует, чтобы, размещая свои ресурсы на компьютере, пользователь не приобретал возможности получения всех привилегий на другом компьютере системы. Условие 2 требует, чтобы, обращаясь к компьютеру, пользователь не давал другим пользователям возможности получения всех привилегий на каком-либо компьютере системы (в том числе и на данном).

Непосредственная проверка условий 1, 2 требует рассмотрения всех траекторий функционирования системы, что является алгоритмически неразрешимой задачей в общем случае. Поэтому сформулируем эффективно проверяемые и практически реализуемые в рассматриваемых компьютерных системах условия, выполнение которых необходимо и достаточно для выполнения условий 1 и 2.

**Теорема 7.1.** Компьютерная система, построенная на основе ОС семейств *Windows NT/2000/XP*, удовлетворяет требованиям безопасного администрирования тогда и только тогда, когда выполняются условия 3 и 4.

**Условие 3.** Для  $u \in U$ ,  $c_1 \in C$  истинно:

$$(c_1 \in \text{write}(u)) \Rightarrow (\{c \in C: c_1 \rightarrow c\} \subset \{c_1\} \cup \{c \in C: u \in U_c\}).$$

**Условие 4.** Для  $u \in U$ ,  $c_1 \in C$  истинно:

$$(c_1 \in \text{read}(u)) \Rightarrow (\{c \in C: u \in U_c\} \subset \{c \in C: c_1 \rightarrow c\}).$$

**Замечание 7.7.** Условие 3 означает, что пользователь имеет возможность разместить свой ресурс на компьютере в двух случаях:

- если данному компьютеру не подчинен кроме него ни один компьютер;
- если пользователь является доверенным пользователем всех компьютеров, подчиненных данному.

Условие 4 означает, что пользователь имеет возможность обратиться к компьютеру только в случае, если все компьютеры, доверенным пользователем которых является данный пользователь, подчинены данному компьютеру.

**Доказательство.** Доказательство выполним методом от противного.

Докажем, что условие 1 выполняется тогда и только тогда, когда выполняется условие 3.

Докажем необходимость выполнения условия 3 для выполнения условия 1.

Пусть найдутся  $u \in U$ ,  $c_1 \in \text{write}(u)$ , для которых не выполняется условие 3 теоремы, т.е. существует  $c_2 \in C$  такой, что  $c_2 \neq c_1$ ,  $c_1 \rightarrow c_2$  и  $u \notin U_{c_2}$ .

Из исходных предположений 1 и 2 следует, что существуют  $t_1 > 0$  и  $p_1 \in P$ , для которых  $c_1 \in \text{write}(u)_{p_1}^{t_1}$ . Из исходного предположения 5 следует, что  $u \notin U_{c_1}$ , следовательно,  $\text{right}(u, c_1)_{p_1}^{t_1} \neq R_{c_1}$  и  $\text{right}(u, c_2)_{p_1}^{t_1} \neq R_{c_2}$ . Из определения 7.5 и определения угрозы 1 следует, что существует  $t_2 > t_1$ , для которого  $\text{right}(u, c_1)_{p_1}^{t_2} = R_{c_1}$ . Пусть  $t_2$  выбрано минимальным. Из исходного предположения 5 следует, что  $\text{right}(u, c_2)_{p_1}^{t_2} = R_{c_2}$ . Таким образом, размещение пользователем и своих ресурсов на компьютере  $c_1$  было использовано им для получения всех привилегий  $R_{c_2}$  на компьютере  $c_2$ . Так как  $t_2$  минимально, то невозможно найти иную траекторию функционирования компьютерной системы  $p_2 \in P$ , для которой бы выполнялись условия  $\text{right}(u, c_2)_{p_2}^{t_2-2} \neq R_{c_2}$ ,  $\text{right}(u, c_2)_{p_2}^{t_2-1} = R_{c_2}$ . Следовательно, условие 1 основного постулата не выполняется. Таким образом, выполнение условия 3 теоремы необходимо для выполнения условия 1 основного постулата.

Докажем достаточность выполнения условия 3 для выполнения условия 1.

Пусть найдутся  $p_1 \in P$ ,  $t_2 > t_1 > 0$ ,  $u \in U$ ,  $c_1$ ,  $c_2 \in C$  такие, что  $c_1 \neq c_2$ , для которых не выполняется условие 1, т.е.  $c_1 \in \text{write}(u)_{p_1}^{t_1}$  и  $\text{right}(u, c_2)_{p_1}^{t_2-1} \neq R_{c_2}$ ,  $\text{right}(u, c_2)_{p_1}^{t_2} = R_{c_2}$  и не существует  $p_2 \in P$  такой, что  $\text{right}(u, c_2)_{p_2}^{t_2-2} \neq R_{c_2}$ ,  $\text{right}(u, c_2)_{p_2}^{t_2-1} = R_{c_2}$ . Таким образом, именно размещение пользователем и своих ресурсов на компьютере  $c_1$  было использовано им для получения всех привилегий на компьютере  $c_2$ . Так как  $c_1 \neq c_2$ , то из исходного предположения 5 следует, что  $c_1 \rightarrow c_2$ . Так как  $\text{right}(u, c_2)_{p_1}^{t_2-1} \neq R_{c_2}$ , то  $u \notin U_{c_2}$ . Следовательно, условие 3 не выполняется. Таким образом, выполнения условия 3 достаточно для выполнения условия 1.

Докажем, что условие 2 выполняется тогда и только тогда, когда выполняется условие 4.

Докажем необходимость выполнения условия 4 для выполнения условия 2.

Пусть найдутся  $u_1 \in U$ ,  $c_1 \in \text{read}(u_1)$ , для которых не выполняется условие 4 теоремы, т.е. существует  $c_2 \in C$  такой, что  $c_2 \neq c_1$ ,  $u_1 \in U_{c_2}$  и  $c_2 \notin \{c \in C: c_1 \rightarrow c\}$ .

Из исходных предположений 1 и 2 следует, что существуют  $t_1 > 0$  и  $p_1 \in P$ , для которых  $c_1 \in \text{read}(u_1)_{p_1}^{t_1}$ . Из определения 7.5, определения угрозы 2 и исходного предположения 6 следует, что существует  $u_2 \in U$  такой, что  $c_1 \in \text{write}(u_2)_{p_1}^{t_1}$ ,  $R_{c_2} = \text{right}(u_1, c_2)_{p_1}^{t_1} \neq \text{right}(u_2, c_2)_{p_1}^{t_1}$ , и существует  $t_2 > t_1$ , для которого  $R_{c_2} = \text{right}(u_1, c_2)_{p_1}^{t_2} = \text{right}(u_2, c_2)_{p_1}^{t_2}$ . Пусть  $t_2$  выбрано минимальным. Таким образом, условие 2 не выполняется, так как обращение пользователя  $u_1$  к компьютеру  $c_1$  было использовано пользователем  $u_2$  для получения всех привилегий на компьютере  $c_2$ . Так как  $t_2$  минимально, то невозможно найти иную траекторию функционирования компьютерной системы  $p_2 \in P$ , для которой бы выполнялись условия  $\text{right}(u_2, c_2)_{p_2}^{t_2-2} \neq R_{c_2}$ ,  $\text{right}(u_2, c_2)_{p_2}^{t_2-1} = R_{c_2}$ . Следовательно, выполнение условия 4 необходимо для выполнения условия 2.

Докажем достаточность выполнения условия 4 для выполнения условия 2.

Пусть найдутся  $p_1 \in P$ ,  $t_2 > t_1 > 0$ ,  $u_1$ ,  $u_2 \in U$ ,  $c_1$ ,  $c_2 \in C$  такие, что  $u_1 \neq u_2$ , для которых не выполняется условие 2, т.е.  $c_1 \in \text{read}(u_1)_{p_1}^{t_1}$ ,  $\text{right}(u_2, c_2)_{p_1}^{t_2-1} \neq R_{c_2}$ ,  $\text{right}(u_2, c_2)_{p_1}^{t_2} = R_{c_2}$ , и не существует  $p_2 \in P$  такой, что  $\text{right}(u_2, c_2)_{p_2}^{t_2-2} \neq R_{c_2}$ ,  $\text{right}(u_2, c_2)_{p_2}^{t_2-1} = R_{c_2}$ . При этом реализовалась угроза 2 и обращение пользователя  $u_1$  к компьютеру  $c_1$  было использовано пользователем  $u_2$  для получения всех привилегий на компьютере  $c_2$ . Следовательно,  $\text{right}(u_1, c_2)_{p_1}^{t_1} = R_{c_2}$ ,  $c_1 \in \text{write}(u_2)_{p_1}^{t_1}$  и по определению 7.5 и определению угрозы 1  $\text{right}(u_2, c_1)_{p_1}^{t_1} = R_{c_1}$ . Так как  $t_2 > t_1$  и  $\text{right}(u_2, c_2)_{p_1}^{t_2-1} \neq R_{c_2}$ , из исходного предположения 5 следует, что  $c_2 \notin \{c \in C: c_1 \rightarrow c\}$ . Без ограничения общности будем считать, что  $u_1 \in U_{c_2}$ , так как все привилегии могут быть получены недоверенным пользователем через обращение к компьютеру либо от другого недоверенного пользователя, уже обладающего всеми привилегиями, либо от доверенного пользова-

теля. Но в начальном состоянии компьютерной системы не существует недоверенных пользователей, обладающих всеми привилегиями на каком-либо компьютере. Следовательно,  $c_2 \in \{c \in C: u_1 \in U_c\} \setminus \{c \in C: c_1 \rightarrow c\}$  и условие 4 не выполняется. Таким образом, выполнения условия 4 достаточно для выполнения условия 2. Техорема доказана.

### 7.3.3. Реализация политики безопасного администрирования

Политика безопасного администрирования реализуется на основе обеспечения выполнения в компьютерной системе ряда требований к настройке, конфигурированию и администрированию ОС.

**Требование 1.** Настроить параметры компьютерной системы защиты в соответствии с требованиями уровня адекватности *EAL4* и профиля защиты *Controlled Access Protection Profile Version 1.d «Общих критериев»* [10].

**Требование 2.** Администратор домена должен выполнять свои функции только на контроллере домена.

**Требование 3.** Аутентификационные данные пользователей домена (их бюджеты) должны быть труднодоступными для нарушителя, для чего размещение файловых данных и запуск процессов на контроллере домена должны быть разрешены только Администратору домена. На рабочих станциях домена локально не должны быть зарегистрированы пользователи.

**Требование 4.** В домене для каждой рабочей станции должен быть зарегистрирован выделенный пользователь для выполнения на ней административных функций, который администрирует и осуществляет вход в домен только с данной рабочей станции и не обращается к другим рабочим станциям домена или любым компьютерам доверяющих доменов.

**Требование 5.** При определении порядка обращения друг к другу компонент компьютерной системы необходимо исключить обращения от компьютеров с большим доверием к компьютерам с меньшим доверием.

Обоснуем требования. Требование 1 принимается априорно и обоснованию не подлежит.

Покажем, что из условий 3 и 4 теоремы 7.1 следуют сформулированные требования 2—5, реализующие политику безопасного администрирования.

Из условия 3 следует требование 3, так как условие 3 запрещает размещение пользователями (кроме Администратора домена) своих ресурсов (локальный вход, удаленный запуск процессов и размещение файловых данных) на контроллере домена (рис. 7.6).

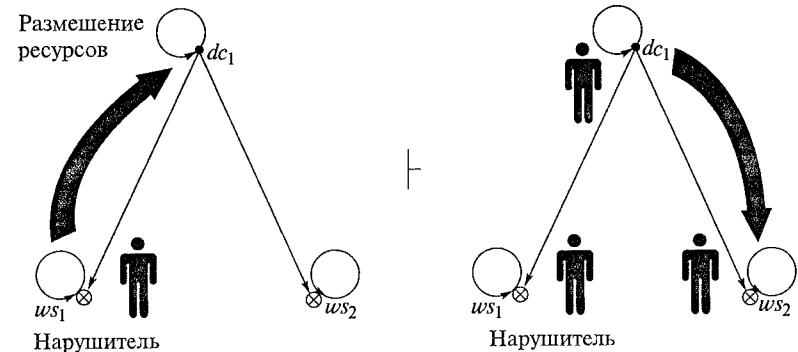


Рис. 7.6. Пример реализации атаки нарушителя по захвату всех привилегий в домене в случае невыполнения условия 3 теоремы 7.1

Условие 4 запрещает Администратору домена осуществлять удаленное обращение к рабочим станциям домена и любым компьютерам доверяющих доменов или входить на них локально (так как локальный вход также сопровождается чтением данных). Следовательно, Администратор домена должен выполнять свои функции только на контроллере домена, т. е. должно выполняться требование 2, и для выполнения административных функций на рабочей станции должен быть выделен пользователь, не являющийся Администратором домена. При этом из условия 4 следует, что такой пользователь является Администратором только одной рабочей станции и не должен входить в домен с других рабочих станций и любых компьютеров доверяющих доменов, т. е. должно выполняться требование 4 (рис. 7.7).

Также из условия 4 непосредственно следует требование 5.

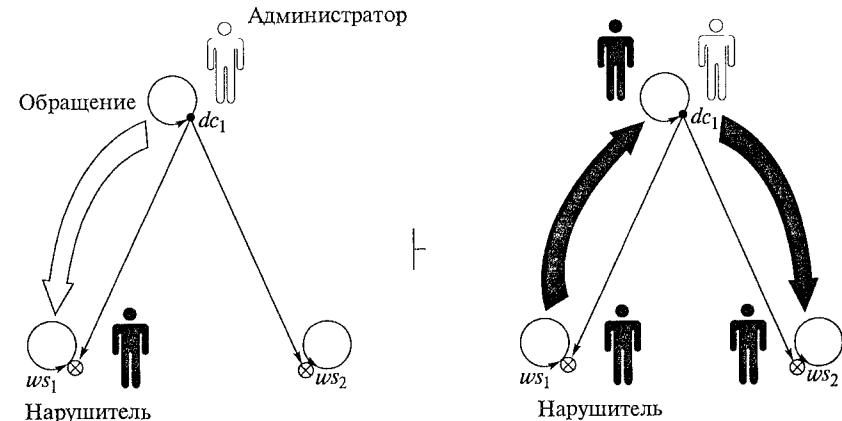


Рис. 7.7. Пример реализации атаки нарушителя по захвату всех привилегий в домене в случае невыполнения условия 4 теоремы 7.1

## Контрольные вопросы

1. В чем состоит основная проблема реализации системы защиты в произвольной компьютерной системе?
2. Возможна ли реализация атаки с использованием программных закладок вида «троянский конь», описанной в подразд. 7.2, в системах с мандатным разграничением доступа?
3. Известно, что при реализации мандатного разграничения доступа может допускаться использование правил дискреционного разграничения доступа (например, с использованием *ds*-свойства безопасности модели Белла — Лападула). Учитывая это обстоятельство, предложите подход по созданию неблагоприятного информационного потока от объекта с высоким уровнем конфиденциальности в объект с низким уровнем конфиденциальности, использующий кооперацию субъектов компьютерной системы.
4. Каким образом мандатная политика целостности Биба была использована для обоснования политики безопасного администрирования компьютерных систем, построенных на основе ОС семейств *Windows NT/2000/XP*?

5. При каких условиях будет соответствовать требованиям политики безопасного администрирования реализация системы централизованного сбора на контроллере домена данных журналов аудита рабочих станций домена?

## СПИСОК ЛИТЕРАТУРЫ

1. Гостехкомиссия России. Руководящий документ. Концепция защиты средств вычислительной техники и автоматизированных систем от несанкционированного доступа к информации. — М.: Военное издательство, 1992.
2. Грушо А.А., Тимонина Е.Е. Теоретические основы защиты информации. — М.: Издательство Агентства «Яхтсмен», 1996.
3. Зегжеда Д.П., Ивашко А.М. Основы безопасности информационных систем. — М.: Горячая линия — Телеком, 2000.
4. Кнут Д. Искусство программирования для ЭВМ. Т. 1. — М: Мир, 1976.
5. Носов В.А. Основы теории алгоритмов и анализа их сложности. Курс лекций. — М.: МГУ, 1992.
6. Теоретические основы компьютерной безопасности / П.Н.Девянин, О.О.Михальский, Д.И.Правиков, А.Ю.Щербаков. — М.: Радио и связь, 2000.
7. Щербаков А.Ю. Введение в теорию и практику компьютерной безопасности. — М.: Издатель Молгачева С.В., 2001.
8. Amman P.E., R. Sandhu. Safety Analyses for the Extend Schematic Protection Model. Prvc. IEEE Symposium of Research in Security and Privacy, 1991. — P. 87—97.
9. Bell D.E., LaPadula L.J. Secure Computer Systems: Unified Exposition and Multics Interpretation. — Bedford, Mass.: MITRE Corp., 1976. — MTR-2997 Rev. 1.
10. Common Criteria for Information Technology Security Evaluation. — Communication Security Establishment (Canada), Service Central de la Securite des Systemes d'Information (France), Bundesamt fur Sicherheit in der Informationstechnik (Germany), Netherlands National Communication Security Agency (Netherlands), Communication-Electronics Security Group (United Kingdom), National Institute of Standards and Technology (United States), National Security Agency (United States), 1998. — Version 2.0.
11. Database Security / Castano S., Fugini M.G., Martella G., Samarati P. — Addison Wesley Publishing Company, ACM Press, 1995. — P. 456.
12. Frank J., Bishop M. Extending The Take-Grant Protection System. — Department of Computer Science. — University of California at Davis, 1984.
13. Harrison M., Ruzzo W. Monotonic protection systems / In DeMillo R., Dobkin D., Jones A., Lipton R., editors // Foundation of Secure Computation. — New York: Academic Press, 1978. — P. 337—365.
14. Harrison M., Ruzzo W., Ullman J. Protection in operating systems. Communication of ACM, 19(8): 461—471, August 1976.

15. Information Technology Security Evaluation Criteria. Harmonized Criteria Of France-Germany-Netherlands-United Kingdom. — London: Department Of Trade and Industry, 1991.

16. Lanawehrm E., Heitmeyer L., McLean J. A Security Model for Military Message Systems. — ACM Trans. On Computer Systems. — Vol. 9, № 3. — P. 198—222.

17. Lipton R., Snyder L. On synchronization and security. In DeMillo R., Dobkin D., Jones A., Lipton R., editors, Foundation of Secure Computation. — New York: Academic Press, 1978. — P. 367—385.

18. McLean J., John D. Security Models and Information Flow, Proceedings of 1990, IEEE Symposium on Research in Security and Privacy. — IEEE Press, 1990.

19. McLean J., John D. The Specification and Modeling of Computer Security. Computer. — Vol. 23, No. 1, Jan. 1990.

20. Sandhu R. Rationale for the RBAC96 family of access control models. In Proceeding of the 1<sup>st</sup> ACM Workshop on Role-Based Access Control. — ACM, 1997.

21. Sandhu R. Role-Based Access Control, Advanced in Computers. — Vol. 46. — Academic Press, 1998.

22. Sandhu R. The typed access matrix model. In Proceeding of the IEEE Symposium on Research in Security and Privace. — Oakland, CA, May 1992. — P. 122—136.

23. Trusted Computer System Evaluation Criteria. — US Department Of Defense, 1985. — CSC-STD-001-83.

## ОГЛАВЛЕНИЕ

Предисловие .....	3
<b>Глава 1. Основные понятия и определения, используемые при описании моделей безопасности компьютерных систем .....</b>	<b>4</b>
1.1. Элементы теории защиты информации .....	4
1.1.1. Объект, субъект, доступ .....	4
1.1.2. Классификация угроз безопасности информации .....	5
1.1.3. Основные виды политики безопасности .....	7
1.2. Математические основы моделей безопасности .....	10
1.2.1. Основные понятия .....	10
1.2.2. Элементы теории автоматов .....	11
1.2.3. Элементы теории графов .....	12
1.2.4. Алгоритмически разрешимые и алгоритмически неразрешимые проблемы .....	13
1.2.5. Модель решетки .....	14
1.3. Основные виды моделей безопасности .....	17
<b>Глава 2. Модели систем дискреционного разграничения доступа .....</b>	<b>18</b>
2.1. Модель матрицы доступов XPU .....	18
2.1.1. Элементы модели XPU .....	18
2.1.2. Анализ безопасности систем XPU .....	20
2.1.3. Модель типизированной матрицы доступов .....	25
2.2. Модель распространения прав доступа <i>Take-Grant</i> .....	32
2.2.1. Основные положения классической модели <i>Take-Grant</i> .....	32
2.2.2. Расширенная модель <i>Take-Grant</i> .....	43
2.2.3. Представление систем <i>Take-Grant</i> системами XPU .....	51
<b>Глава 3. Модели систем мандатного разграничения доступа .....</b>	<b>55</b>
3.1. Модель Белла—ЛаПадула .....	55
3.1.1. Классическая модель Белла—ЛаПадула .....	55
3.1.2. Пример некорректного определения свойств безопасности .....	60
3.1.3. Политика <i>low-watermark</i> в модели Белла—ЛаПадула .....	61
3.1.4. Безопасность переходов .....	63

3.1.5. Модель мандатной политики целостности информации Биба .....	66	7.1.2. Гомоморфизм компьютерной системы и ее математической модели безопасности .....	122
3.2. Модель систем военных сообщений .....	68	7.2. Проблемы реализации политики безопасности .....	123
3.2.1. Общие положения и основные понятия .....	68	7.2.1. Реализация дискреционной политики безопасности .....	123
3.2.2. Неформальное описание модели СВС .....	70	7.2.2. Реализация мандатной политики безопасности .....	125
3.2.3. Формальное описание модели СВС .....	71	7.3. Обоснование политики безопасного администрирования ОС семейств <i>Windows NT/2000/XP</i> .....	128
<b>Г л а в а 4. Модели безопасности информационных потоков .....</b>	<b>78</b>	7.3.1. Общие положения .....	128
4.1. Автоматная модель безопасности информационных потоков .....	78	7.3.2. Математическая модель политики безопасного администрирования .....	129
4.2. Программная модель контроля информационных потоков .....	80	7.3.3. Реализация политики безопасного администрирования .....	136
4.2.1. Основные элементы модели .....	80		
4.2.2. Контролирующий механизм защиты .....	82		
4.3. Вероятностная модель безопасности информационных потоков .....	83		
4.3.1. Схема компьютерной системы .....	83		
4.3.2. Информационная невыводимость .....	84		
4.3.3. Информационное невлияние .....	84		
<b>Г л а в а 5. Модели ролевого разграничения доступа .....</b>	<b>88</b>		
5.1. Понятие ролевого разграничения доступа .....	88		
5.2. Базовая модель РРД .....	88		
5.3. Модель администрирования РРД .....	92		
5.3.1. Основные положения .....	92		
5.3.2. Администрирование множеств авторизованных ролей пользователей .....	93		
5.3.3. Администрирование множеств прав доступа, которыми обладают роли .....	97		
5.3.4. Администрирование иерархии ролей .....	98		
5.4. Модель мандатного РРД .....	101		
5.4.1. Защита от угрозы конфиденциальности информации .....	101		
5.4.2. Защита от угрозы конфиденциальности и целостности информации .....	103		
<b>Г л а в а 6. Субъектно-ориентированная модель изолированной программной среды .....</b>	<b>108</b>		
6.1. Основные понятия. Монитор безопасности объектов .....	108		
6.2. Монитор безопасности субъектов .....	112		
6.3. Изолированная программная среда .....	115		
<b>Г л а в а 7. Проблемы применения моделей безопасности при построении защищенных компьютерных систем .....</b>	<b>121</b>		
7.1. Проблема адекватности реализации модели безопасности в реальной компьютерной системе .....	121		
7.1.1. Общая постановка задачи построения системы защиты .....	121		