2008 Математические основы компьютерной безопасности № 1(1)
DOI 10.17223/20710410/1/12
УДК 004.94

ПРИМЕНЕНИЕ ДП-МОДЕЛЕЙ ДЛЯ АНАЛИЗА ЗАЩИЩЕННОСТИ СЕТЕЙ

Д.Н. Колегов

Томский государственный университет

E-mail: kden@sibmail.com

В статье вводится понятие замыкания ДП-моделей. Описываются и обосновываются алгоритмы их построения. Рассматривается применение таких моделей для анализа защищенности компьютерных сетей. Предлагаются методы анализа путей распространения прав доступа и информационных потоков. В рамках теории ДП-моделей формализуются модели топологического анализа защищенности REM и VTG.

Ключевые слова: *ДП-модель*, *ФАС ДП-модель*, *модель Take-Grant*, *замыкание*, *топологический анализ защищенности*, *графы атак*, *моделирование атак.*

В современных методологиях анализа защищенности компьютерных систем (КС) учитывается «топология» КС – взаимосвязь объектов и субъектов КС, их свойств и характеристик. Такой анализ защищенности называется *топологическим*, а автоматизированные средства, его выполняющие, *топологическими сканерами безопасности*.

Основной математической моделью топологического анализа защищенности является граф атак, который строится на основе модели нарушителя, конфигурации сети (правил фильтрации межсетевого экрана (МЭ), маршрутизации, обнаружения и предотвращения атак, достижимости хостов и т.д.), результатов сканирования сети, анализа уязвимостей или теста на вторжение. Центральными задачами являются синтез графа атак и его анализ (вероятностный, минимизационный и т.д.).

Граф атак содержит все известные траектории (сценарии, пути) атак реализации нарушителем угроз. Результатом его анализа может являться:

- перечень успешных атак, не обнаруживаемых IDS;
- соотношение реализуемых мер безопасности и уровня защищенности сети;
- перечень наиболее критичных уязвимостей;
- перечень мер, позволяющих предотвратить использование уязвимостей в ПО, для которого отсутствуют обновления;
- наименьшее множество мер, реализация которых сделает сеть защищенной.

Графы атак также используются при расследовании компьютерных инцидентов, для анализа рисков и корреляций предупреждений систем обнаружения атак. Ключевой проблемой построения графа атак является масштабируемость – возможность построения графа атак для сети с большим числом хостов и уязвимостей.

В настоящее время представлено значительное многообразие подходов к синтезу и анализу графов атак [1]. Большинство работ посвящено синтезу эффективных методов построения графов атак и лишь единицы – формальным моделям анализа современных КС, исследованию и математическому обоснованию их свойств. Так, Shahriary, Sadoddin, Jalili и Zakeri [2] использовали положения классической модели Take-Grant для синтеза полиномиального алгоритма построения графа атак путем построения замыкания графа доступов модели Take-Grant. В то же время известно [3], что модель Take-Grant является неадекватной для анализа безопасности современных КС. Одной из современных моделей анализа безопасности КС с дискреционным управлением доступа является развитие модели Take-Grant – ДП-модель и ее расширения.

Основой всех ДП-моделей является базовая ДП-модель. В ней моделируемая КС представляется абстрактной иерархической системой, состоящей из сущностей, каждое состояние которой представляется графом доступов. Переход из состояния в состояние осуществляется в результате применения одного из правил преобразования графа доступа: *take_right*(), *grant_right*(), *remove_right*(), *own_take*(), *own_remove*(), *create_subject*(), *delete_entity*(), *rename_entity*(), *access_read*(), *access_write*(), *access_append*(), *flow*(), *find*(), *post*(), *pass*() и *control*(). Для описания условий возникновения информационных потоков $write_m$, $write_t$ между сущностями используются права доступа $read_r$, $write_r$, $append_r$, $execute_r$, own_r и виды доступа $read_a$, $write_a$, $append_a$. Для описания условий передачи между сущностями КС прав доступа используется право доступа own_r. Субъекты могут создавать новые сущности и получать к ним любые права доступа.

С помощью ДП-модели анализируются условия передачи прав доступа и реализации информационных потоков по памяти и по времени при кооперации субъектов и без кооперации доверенных и недоверенных субъектов, а также предлагаются методы их предотвращения. На основе базовой ДП-модели путем изменения правил преобразования состояний строятся: ДП-модель без кооперации субъектов (БК ДП-модель), ДП-

модель с блокирующими доступами доверенных субъектов (БД ДП-модель) и ДП-модель с функционально-ассоциированными сущностями (ФАС ДП-модель). Последняя позволяет анализировать условия получения прав доступа к объекту при реализации информационных потоков по памяти, что и позволяет успешно использовать ее при анализе защищенности современных КС.

1. Обозначения и определения

В соответствии с [3] будем использовать следующие обозначения и определения: $E = O \cup C$ – множество сущностей, где O – множество объектов, C – множество контейнеров и $O \cap C = \varnothing$; $S \subseteq E$ – множество субъектов; $R_r = \{read_r, write_r, append_r, execute_r, own_r\}$ – множество видов прав доступа, где $read_r$ – право доступа на чтение из сущности, $write_r$ – право доступа на запись в сущность, $append_r$ – право доступа на запись в конец сущности, $execute_r$ – право доступа на выполнение (активизацию) сущности, own_r – право доступа на владение сущностью, позволяющее субъекту-владельцу передавать права доступа к сущности другим субъектам или удалить сущность; $R_a = \{read_a, write_a, append_a\}$ – множество видов доступа, где $read_a$ – доступ на чтение из сущности, $write_a$ – доступ на запись в сущность, $append_a$ – доступ на запись в конец слова, содержащегося в сущности; $R_f = \{write_m, write_t\}$ – множество видов информационных потоков, где $write_m$ – информационный поток по памяти на запись в сущность, $write_t$ – информационный поток по времени на запись в сущность; $R_{raf} = R_r \cup R_a \cup R_f$ – множество видов прав доступа, видов доступа и видов информационных потоков, при этом множества R_r, R_a, R_f попарно не пересекаются.

Для заданного на E отношения частичного порядка $\leq$ функция H: $E \to 2^E$ называется *функцией иерархии сущностей*, если она удовлетворяет условиям:

1) для любой сущности $e \in H(c)$ имеет место $e < c$ и не существует сущности-контейнера $d \in C$, такой, что $e < d$ и $d < c$;

2) для любых сущностей $e_1, e_2 \in E$ если $e_1 \neq e_2$, то $H(e_1) \cap H(e_2) = \varnothing$, и если $e_1 < e_2$, то или $e_1, e_2 \in E \setminus S$, или $e_1, e_2 \in S$;

3) $H(o) = \varnothing$ для любого $o \in O$; $H(e) \subset E \setminus S$ для любого $e \in E \setminus S$; $H(s) \subset S$ для всех $s \in S$.

Пусть определены множества S, E, $R \subseteq S \times E \times R_r$, $A \subseteq S \times E \times R_a$, $F \subseteq E \times E \times R_f$ и функция иерархии сущностей H. Конечный помеченный ориентированный граф без петель $G = (S, E, R \cup A \cup F, H)$, где элементы множеств S, E являются вершинами графа, а элементы множества $R \cup A \cup F$ ребрами, называют *графом доступов*.

Пусть также $\Sigma(G^*, OP)$ есть система, каждое состояние которой представляется графом доступов и где G^* – множество всех возможных состояний, OP – множество правил преобразования состояний [3]. Обозначим $G \vdash_{op} G'$ – переход системы $\Sigma(G^*, OP)$ из состояния G в состояние G' с использованием правила преобразования состояний $op \in OP$ и $\Sigma(G^*, OP, G_0)$ – систему $\Sigma(G^*, OP)$ с начальным состоянием G_0.

Пусть, наконец, подмножества $L_a \subset S \times O \times R_a$, $L_r \subset S \times O \times R_r$, $L_f \subset O \times O \times R_f$ являются множествами соответственно разрешенных доступов субъектов к объектам, разрешенных прав доступа субъектов к объектам, разрешенных информационных потоков между объектами и подмножества $N_a \subset S \times O \times R_a$, $N_r \subset S \times O \times R_r$ и $N_f \subset O \times O \times R_f$ суть множества соответственно запрещенных доступов субъектов к объектам, запрещенных прав доступа субъектов к объектам и запрещенных информационных потоков между объектами КС. При этом выполняются условия: $L_a \cap N_a = \varnothing$, $L_a \cup N_a = S \times O \times R_a$, $L_r \cap N_r = \varnothing$, $L_r \cup N_r = S \times O \times R_r$, $L_f \cap N_f = \varnothing$ и $L_f \cup N_f = O \times O \times R_f$.

Сущность называется *функционально-ассоциированной* с субъектом, если от нее зависит вид преобразования данных, реализуемого субъектом в данном или некотором последующем состоянии системы $\Sigma(G^*, OP)$.

Элементы множеств N_a, N_r, N_f будем называть *угрозами* нарушений безопасности в КС. *Нарушением безопасности* в КС называют ее переход в состояние, в котором либо получен запрещенный доступ из N_a, либо произошла утечка запрещенного права доступа из N_r, либо реализован запрещенный информационный поток из N_f.

2. Построение замыкания ДП-моделей

Для проверки истинности предикатов *can_share*(), *can_write_memory*(), *can_steal*() и др. предикатов в ДП-моделях [3] наиболее эффективно использовать алгоритмы, позволяющие осуществлять проверку истинности указанных предикатов для всех пар вершин одновременно и проводить анализ путей распространения прав доступа и информационных потоков. Такие алгоритмы реализуют преобразование графа доступов в его замыкание.

При построении замыкания графа доступов необходимо выполнить два стандартных для данной задачи шага [4]. Первый шаг направлен на устранение возможности применения немонотонных правил преобразования состояний и обосновывается тем, что при анализе условий передачи прав доступа, реализации информационных потоков по памяти и по времени возможно использование только монотонных правил преобразования состояний [3]. Поэтому при построении замыкания графа доступов немонотонные правила можно

исключить из рассмотрения. После этого граф доступов может только расти. Для того чтобы число вершин в графе не увеличивалось до бесконечности, каждый субъект должен иметь объект с правом владения на него. Этого будет достаточно для передачи субъектом прав доступа, создания субъектов или реализации информационных потоков, после чего в графе число вершин увеличиваться не будет, а число ребер между парой вершин будет ограничено числом прав доступа, видов доступа и информационных потоков.

На основе схемы построения замыкания графа доступов модели Take-Grant [5] дадим определения, построим и обоснуем алгоритмы замыкания графа доступов ДП-моделей.

Пусть $G = (S, E, R \cup A \cup F, H)$ – граф доступов, где для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $(s, z, append_r) \in R$, существует $y \in E$, что $y < z$ и $(s, y, own_r) \in R$. Тогда *own-замыканием* графа G будем называть граф $G^{own} = (S, E, R^{own} \cup A \cup F^{own}, H)$, полученный из G применением последовательности правил *take_right*() и *grant_right*(), если при этом выполнены следующие условия:

- для всех ребер $(a, b, \alpha) \in R^{own} \setminus R$: $\alpha = own_r$;
- для всех ребер $(c, d, \beta) \in F^{own} \setminus F$: $\beta = write_t$;
- все ребра $(c, d, \beta) \in F^{own} \setminus F$ получены в результате применения правил *take_right*() и *grant_right*();
- применение последовательности правил *take_right*() и *grant_right*() не приводит к появлению новых ребер указанных видов.

Пусть $G = (S, E, R \cup A \cup F, H)$ – граф доступов, где для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $(s, z, append_r) \in R$, существует $y \in E$, что $y < z$ и $(s, y, own_r) \in R$. Тогда *tgo-замыканием* графа G будем называть граф $G^{tgo} = (S, E, R^{tgo} \cup A \cup F^{tgo}, H)$, полученный из G применением последовательности правил *take_right*(), *grant_right*() и *own_take*(), если при этом выполнены условия:

- для всех ребер $(a, b, \alpha) \in R^{tgo} \setminus R$: $\alpha \in R_r$;
- для всех ребер $(c, d, \beta) \in F^{tgo} \setminus F$: $\beta = write_t$;
- все ребра $(c, d, \beta) \in F^{tgo} \setminus F$ получены в результате применения правил *take_right*(), *grant_right*() и *own_take*();
- применение последовательности правил *take_right*(), *grant_right*() и *own_take*() не приводит к появлению новых ребер указанных видов.

Пусть $G = (S, E, R \cup A \cup F, H)$ – граф доступов, где для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $(s, z, append_r) \in R$, существует $y \in E$, что $y < z$ и $(s, y, own_r) \in R$. Тогда *замыканием* (*access-замыканием*) графа G будем называть граф $G^{access} = (S, E, R^{access} \cup A^{access} \cup F^{access}, H)$, полученный из G применением последовательности правил *take_right*(), *grant_right*(), *own_take*(), *access_read*(), *access_write*(), *access_append*(), *flow*(), *find*(), *post*() и *pass*(), если при этом применение к графу G^{access} указанных правил не приводит к появлению в нем новых ребер.

Пусть $G = (S, E, R \cup A \cup F, H)$ – граф доступов, где для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $s, z, append_r) \in R$, существует $y \in E$, что $y < z$ и $(s, y, own_r) \in R$. Тогда *control-замыканием* графа G будем называть граф $G^{control} = (S, E, R^{control} \cup A^{control} \cup F^{control}, H)$, полученный из G применением последовательности правил *take_right*(), *grant_right*(), *own_take*(), *access_read*(), *access_write*(), *access_append*(), *flow*(), *find*(), *post*(), *pass*() и *control*(), если при этом применение к графу $G^{control}$ указанных правил не приводит к появлению в нем новых ребер.

Опишем и обоснуем алгоритмы построения замыкания графов доступов ДП-моделей.

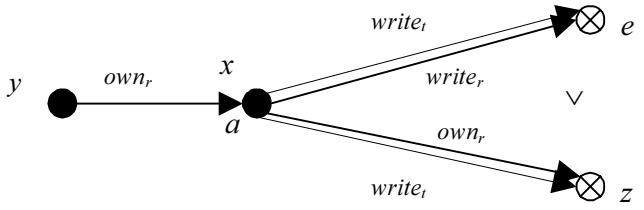
2.1. Построение замыкания графа доступов базовой ДП-модели

Алгоритм 1 построения *own*-замыкания графа доступов.

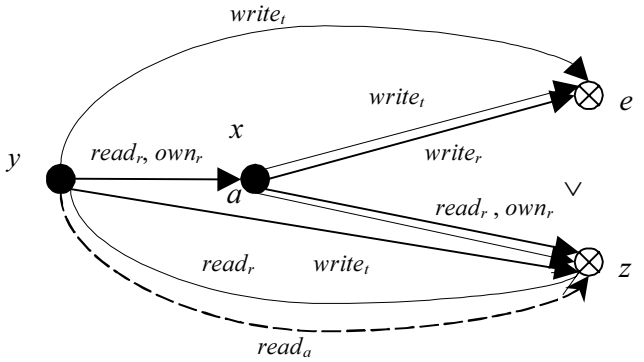
1. Для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $(s, z, append_r) \in R$, последовательно применить правила *create_entity*(x, y, z), *own_take*$(execute_r, x, y)$, *create_subject*(x, v, y). Новые сущности и ребра занести в S, E, R и F, определить H с учетом изменения иерархии сущностей.
2. $L = \{(x, y, own_r) \in R\}$, $N = \varnothing$.
3. Выбрать из списка L первое ребро (x, y, own_r), занести x и y в N, удалить ребро (x, y, own_r) из L.
4. Для всех вершин $z \in N$ проверить возможность применения правил *take_right*() и *grant_right*() на вершинах x, y, z с использованием ребра (x, y, own_r). Если в результате применения правил получаются новые ребра вида (a, b, own_r), где $\{a, b\} \subset \{x, y, z\}$, то занести их в конец L и R^{own}, а ребра вида $(a, b, write_t)$ в F^{own}.
5. Если L не пусто, перейти на шаг 2.

Замечание. При построении замыкания графа доступов в модели Take-Grant первый шаг алгоритма построения *tg*-замыкания заключается в применении правила *create*$(s, o, \{t, g, r, w\})$ для каждого $s \in S$, что достаточно для анализа передачи прав доступа и информационных потоков. Так как ДП-модель является иерархической, то для создания сущности z в сущности-контейнере e субъектом x необходимо, чтобы $(x, e, write_r) \in R$ или $(x, e, append_r) \in R$. При этом будут реализованы соответствующие информационные потоки по времени. В состоянии, изображенном на рис.1, субъект y может сначала получить права доступа к сущно-

сти-контейнеру e, а затем с помощью них создать сущность. Однако можно видеть, что субъект y может получить все права доступа к сущности z и реализовать все возможные информационные потоки по времени, поэтому для субъекта y нет необходимости осуществлять правило преобразования *create_entity*(). Точно так же в случае с созданием субъектов на шаге 1 *create_subject*(x, v, y) можно реализовать информационные потоки по времени от v к y с помощью правила *post*(y, x, v). Таким образом, шаг 1 в алгоритме является достаточным.



+ *own_take* ($read_r$, x, z), *take_right* ($read_r$, y, x, z), *access_read*(y, z)



+ *post* (y, x, v)

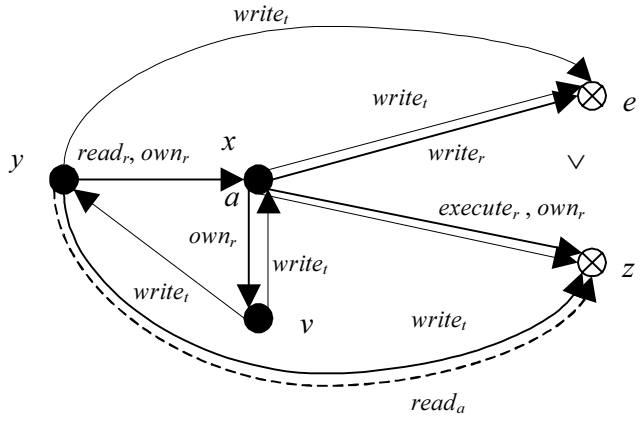


Рис. 1

Теорема 1. Алгоритм 1 корректно строит *own*-замыкание графа доступов.

Доказательство. Пусть $G' = (S, E, R' \cup A \cup F', H)$ – граф доступов, полученный из $G = (S, E, R \cup A \cup F, H)$ в результате применения алгоритма 1. Пусть $G^{own} = (S, E, R^{own} \cup A \cup F^{own}, H)$ есть *own*-замыкание графа G. Необходимо доказать, что $G' = G^{own}$. Покажем, что $R^{own} = R'$. Доказательство проведем методом от противного. Пусть существует ребро $(x, y, own_r) \in R^{own} \setminus R'$. Тогда существуют ребра (a, b, own_r), (c, d, own_r) из R^{own}, применение к которым правила *take_right* или *grant_right* привело к появлению ребра (x, y, own_r). Если (a, b, own_r), $(c, d, own_r) \in R'$, то, согласно описанию алгоритма 1, $(x, y, own_r) \in R'$ и получаем противоречие. Значит, или $(a, b, own_r) \in R^{own} \setminus R'$, или $(c, d, own_r) \in R^{own} \setminus R'$. Вместе с этим граф G^{own} получен из G в результате применения конечной последовательности правил *take_right*() и *grant_right*(), и ребра (a, b, own_r), (c, d, own_r) должны быть получены раньше, чем ребро (x, y, own_r). Проводя рассуждения подобным образом для

ребер, с использованием которых получены ребра (a, b, own_r), (c, d, own_r), придем к противоречию в связи с тем, что все ребра графа G изначально содержатся в графе G'. Таким образом, $R^{own} = R'$. А так как ребро $(x, y, write_t)$ получается в результате применений правил *take_right*() или *grant_right*(), то и $F^{own} = F'$. ■

Алгоритм 2 построения *tgo*-замыкания графа доступов.

1. Выполнить алгоритм 1.
2. Для всех ребер вида $(x, y, own_r) \in R$ применить правило *own_take*(α_r, x, y), $\alpha_r \in R_r \setminus \{own_r\}$, и если полученное ребро $(x, y, \alpha_r) \notin R$, занести его в R^{tgo}.
3. Для каждой пары ребер вида (x, y, own_r), $(y, z, \alpha_r) \in R$ применить правило *take_right*(α_r, x, y, z), и если полученное ребро $(x, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(y, x, write_t)$ – в F^{tgo}.
4. Для каждой пары ребер вида (x, y, own_r), $(x, z, \alpha_r) \in R$ применить правило *grant_right*(α_r, x, y, z), и если полученное ребро $(y, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(x, y, write_t)$ – в F^{tgo}.
5. Для каждой пары ребер вида (x, y, own_r), $(y, z, \alpha_r) \in R$ применить правило *take_right*(α_r, x, y, z), и если полученное ребро $(x, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(y, x, write_t)$ – в F^{tgo}.

Теорема 2. Алгоритм 2 корректно строит *tgo*-замыкание графа доступов.

Доказательство. Для построения *tgo*-замыкания графа доступов всем субъектам графа доступов после выполнения алгоритма 1 необходимо выполнить следующие шаги:

1) используя правило *own_take*(), забрать права доступа;

2) используя правило *grant_right*(), раздать права доступа и, используя правило *take_right*(), забрать права доступа;

3) используя правило *take_righ*()*t*, раздать права доступа и, используя правило *grant_right*(), забрать права доступа;

Объединяя шаги 2 – 3, получаем алгоритм 2. ■

Алгоритм 3 построения *access*-замыкания графа доступов.

1. Выполнить алгоритм 2.
2. Для всех ребер вида $(x, z, write_r) \in R$, где $x \in S$, а $y, z \in E$ и $y \in H(z)$, применить правило *rename_entity*(x, y, z), все полученные новые ребра занести в F^a.
3. Для всех ребер вида $(x, y, read_r) \in R$, $(x, y, read_a) \notin A$ применить правило *access_read*(x, y), полученные новые ребра $(x, y, read_a)$ и $(y, x, write_m)$, $\{(x, e, write_t)$: $e \in E$, $x \neq e$ и $y \leq e\}$ занести в A^a и F^a соответственно.
4. Для всех ребер вида $(x, y, write_r) \in R$, $(x, y, write_a) \notin A$ применить правило *access_write*(x, y), полученные ребра $(x, y, write_a)$ и $(x, y, write_m)$, $\{(x, e, write_t)$: $e \in E$, $x \neq e$ и $y \leq e\}$ занести в A^a и F^a соответственно.
5. Для всех ребер вида $(x, y, append_r) \in R$, $(x, y, append_a) \notin A$ применить правило *access_append*(x, y), полученные ребра $(x, y, append_a)$ и $(x, y, write_m)$, $\{(x, e, write_t)$: $e \in E$, $x \neq e$ и $y \leq e\}$ занести в A^a и F^a соответственно.
6. $L = \{(x, y, \alpha) \in R \cup F\}$, $N = \varnothing$.
7. Выбрать из списка L первое ребро (x, y, α), занести x и y в N, удалить ребро (x, y, α) из L.
8. Для всех вершин $z, y' \in N$ проверить возможность применения правила *flow*() на четверке вершин x, y, y', z с использованием ребер (x, y, α) и (z, y', β) и правил *post*(), *pass*(), *find*() на тройке вершин x, y, z с использованием ребра (x, y, α). Если в результате применения правил получаются новые ребра вида (a, b, γ), где $\{a, b\} \subset \{x, y, y', z\}$, $\gamma \in \{write_m, write_t\}$, то занести их в конец L и F^a.
9. Если L не пусто, перейти на шаг 6.

Теорема 3. Алгоритм 3 корректно строит *access*-замыкание графа доступов.

Доказательство. После выполнения шагов 1 – 5 алгоритма 3 в графе доступов будут все ребра вида $(x, y, \alpha) \in A$, где $\alpha \in \{read_a, write_a, append_a\}$. Шаги 5 – 8 алгоритма 3 аналогичны шагам алгоритма 1, в силу корректности которого по теореме 1 приходим к истинности теоремы 3. ■

2.2. Построение замыкания графа доступов БК ДП-модели

Перечислим основные положения БК ДП-модели:

- субъект $u \in L_S$ называют *доверенным*, если он обладает правом доступа владения к каждой сущности системы;

- каждый субъект системы $\Sigma(G^*, OP)$ является либо доверенным, либо недоверенным;

- доверенные субъекты системы $\Sigma(G^*, OP)$ не участвуют в реализации информационных потоков по времени, т.е. во всех состояниях отсутствуют информационные потоки по времени, исходящие из доверенных субъектов;

- правила преобразования состояний *take_right*(α_r, x, y, z), *grant_right*(α_r, x, y, z) могут быть использованы только недоверенными субъектами.

В соответствии с данными положениями модифицируются условия и результаты применения правил преобразования состояний базовой ДП-модели [3, табл. 2.1]. Опишем и обоснуем алгоритмы построения замыкания графа доступов БК ДП-модели.

Алгоритм 4 построения *own*-замыкания графа доступов.

1. Для всех $s \in S$, $z \in E \setminus S$, таких, что $(s, z, write_r) \in R$ или $(s, z, append_r) \in R$, последовательно применить правила *create_entity*(x, y, z), *own_take*$(execute_r, x, y)$, *create_subject*(x, v, y). Новые сущности и ребра занести в S, E, R и F, определить H с учетом изменения иерархии сущностей.
2. $L = \{(x, y, own_r) \in R\}$, где $x \in N_S$, $N = \varnothing$.
3. Выбрать из списка L первое ребро (x, y, own_r), занести x и y в N, удалить ребро (x, y, own_r) из L.
4. Для всех вершин $z \in N \cap N_S$ проверить возможность применения правил *take_right* и *grant_right* на вершинах x, y, z с использованием ребра (x, y, own_r). Если в результате применения правил получаются новые ребра вида (a, b, own_r), где $\{a, b\} \subset \{x, y, z\}$, то занести их в конец L и R^{own}, а ребра вида $(a, b, write_t)$ для $a \in N_S$ – в F^{own}.
5. Если L не пусто, перейти на шаг 2.

Теорема 4. Алгоритм 1 корректно строит *own*-замыкание графа доступов.

Доказательство теоремы выполняется аналогично доказательству теоремы 1 с учетом того, что только недоверенные субъекты могут инициировать выполнение правил преобразования состояний *take_right*()и *grant_right*(). ■

Алгоритм 5 построения *tgo*-замыкания графа доступов.

1. Выполнить алгоритм 4.
2. Для всех ребер вида $(x, y, own_r) \in R$ применить правило *own_take* (α_r, x, y), $\alpha_r \in R_r \setminus \{own_r\}$; если полученное ребро $(x, y, \alpha_r) \notin R$, занести его в R^{tgo}.
3. Для каждой пары ребер вида (x, y, own_r), $(y, z, \alpha_r) \in R$, где $x \in N_S$, применить правило *take_right* (α_r, x, y, z), и если полученное ребро $(x, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(y, x, write_t)$, где $y \in N_S$, – в F^{tgo}.
4. Для каждой пары ребер вида (x, y, own_r), $(x, z, \alpha_r) \in R$, где $x \in N_S$, применить правило *grant_right* (α_r, x, y, z), и если полученное ребро $(y, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(x, y, write_t)$, где $x \in N_S$, – в F^{tgo}.
5. Для каждой пары ребер вида (x, y, own_r), $(y, z, \alpha_r) \in R$, где $x \in N_S$, применить правило *take_right* (α_r, x, y, z), и если полученное ребро $(x, z, \alpha_r) \notin R$, то занести его в R^{tgo}, а ребро $(y, x, write_t)$, где $y \in N_S$, – в F^{tgo}.

Теорема 5. Алгоритм 5 корректно строит *tgo*-замыкание графа доступов.

Доказательство теоремы выполняется аналогично доказательству теоремы 2 с учетом того, что инициировать выполнение правила *own_take* могут все субъекты, а правил *take_right*() и *grant_right*() – только недоверенные субъекты. ■

Алгоритм 6 построения *access*-замыкания графа доступов.

1. Выполнить алгоритм 5.
2. Для всех ребер вида $(x, z, write_r) \in R$, где $x \in N_S$, а $y, z \in E$ и $y \in H(z)$, применить правило *rename_entity*(x, y, z), при получении новых ребер занести их в F^a.
3. Для всех ребер вида $(x, y, read_r) \in R$, $(x, y, read_a) \notin A$ применить правило *access_read*(x, y), полученные ребра $(x, y, read_a)$ и $(y, x, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E, x \neq e$ и $y \leq e\}$.
4. Для всех ребер вида $(x, y, write_r) \in R$, $(x, y, write_a) \notin A$ применить правило *access_write*(x, y), полученные ребра $(x, y, write_a)$ и $(x, y, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E, x \neq e$ и $y \leq e\}$.
5. Для всех ребер вида $(x, y, append_r) \in R$, $(x, y, append_a) \notin A$ применить правило *access_append*(x, y), полученные ребра $(x, y, append_a)$ и $(x, y, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E, x \neq e$ и $y \leq e\}$.
6. $L = \{(x, y, \alpha) \in R \cup F\}$, $N = \varnothing$.
7. Выбрать из списка L первое ребро (x, y, α), занести x и y в N, удалить ребро (x, y, α) из L.
8. Для всех вершин $z, y' \in N$ проверить возможность применения правила *flow* на четверке вершин x, y, y', z с использованием ребер (x, y, α) и (z, y', β) и правил *post*, *pass*, *find* на тройке вершин x, y, z с использованием ребра (x, y, α). Новые ребра вида (a, b, γ), где $\{a, b\} \subset \{x, y, y', z\}$, $\gamma \in \{write_m, write_t\}$, получающиеся в зависимости от правила преобразования состояния БК ДП-модели и принадлежности субъектов к множествам L_S и N_S, занести в конец L и F^a.
9. Если L не пусто, перейти на шаг 6.

Теорема 6. Алгоритм 6 корректно строит *access*-замыкание графа доступов.

Доказательство теоремы выполняется аналогично доказательству теоремы 3 с учетом правил преобразования БК ДП-модели. ■

2.3. Построение замыкания графа доступов БД ДП-модели

Перечислим основные положения и некоторые обозначения БД ДП-модели:

- любой доступ любого доверенного субъекта к сущности называется блокирующим;
- $E_B \subset E$ – множество сущностей, к которым имеют доступ доверенные субъекты системы $\Sigma(G^*, OP)$;
- H_B: $E \to 2^E$ – функция иерархии сущностей с учетом блокирующих доступов доверенных субъектов, сопоставляющая каждой сущности $c \in E$ множество сущностей $H_B(c) \subset H(c)$, удовлетворяющее условию $H_B(c) = \{e \in H(c): e <_B c\}$, где $<_B$ есть заданное на множестве сущностей E отношение частичного порядка;
- блокирующие доступы доверенных субъектов к сущностям системы $\Sigma(G^*, OP)$ препятствуют реализации информационных потоков по времени с использованием данных сущностей и иерархии сущностей, в которую входит данная сущность, за исключением случая, когда сущность является субъектом и участвует в реализации данного информационного потока по времени.

В соответствии с данными положениями модифицируются условия и результаты применения правил преобразования состояний базовой ДП-модели и БК ДП-модели [3, табл. 3.2].

Опишем и обоснуем алгоритмы построения замыкания графа доступов БД ДП-модели. Для построения *own*-замыкания и *tgo*-замыкания графа доступов БД ДП-модели применяются алгоритмы 4 и 5, так как в соответствии с положениями БД ДП-модели правила *take_right*(), *grant_right*(), *own_take*() применяются с условиями и результатами табл. 2.1 БК ДП-модели из [3].

Алгоритм 7 построения *access*-замыкания графа доступов.

1. Выполнить алгоритм 5.
2. Для всех ребер вида $(x, z, write_r) \in R$, где $x \in N_S$, а $y, z \in E$ и $y \in H(z)$, применить правило *rename_entity*(x, y, z), при получении новых ребер с учетом блокирующих доступов занести их в F^a: $F^a = F^a \cup \{(x, z, write_t)$, если $z \in E \setminus E_B\} \cup \{(x, e, write_t)$: $e \in E \setminus E_B$, $x \neq e$ и $e \le_B y\} \cup \{(x, s, write_t)$: $s \in S$, $x \neq s$ и $(s, e, \alpha_r) \in R$, где $e \in E \setminus E_B$, $e \le_B y$ и $\alpha_r \in R_r\}$.
3. Для всех ребер вида $(x, y, read_r) \in R$, $(x, y, read_a) \notin A$ применить правило *access_read*(x, y), полученные ребра $(x, y, read_a)$ и $(y, x, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E \setminus E_B$, $x \neq e$ и $y \le_B e\}$. Если $x \in L_S$, то $E'_B = E_B \cup \{y\}$ и для всех $e \in E$, таких, что $y \le_B e$, положить $H'_B(e) = \varnothing$.
4. Для всех ребер вида $(x, y, write_r) \in R$, $(x, y, write_a) \notin A$ применить правило *access_write*(x, y), полученные ребра $(x, y, write_a)$ и $(x, y, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E \setminus E_B$, $x \neq e$ и $y \le_B e\}$. Если $x \in L_S \cap S$, то $E'_B = E_B \cup \{y\}$ и для всех $e \in E$, таких, что $y \le_B e$: $H'_B(e) = \varnothing$.
5. Для всех ребер вида $(x, y, append_r) \in R$, $(x, y, append_a) \notin A$ применить правило *access_append*(x, y), полученные ребра $(x, y, append_a)$ и $(x, y, write_m)$ занести в A^a и F^a соответственно. Если $x \in N_S$, то $F^a = F^a \cup \{(x, e, write_t)$: $e \in E \setminus E_B$, $x \neq e$ и $y \le_B e\}$. Если $x \in L_S$, то $E'_B = E_B \cup \{y\}$ и для всех $e \in E$, таких, что $y \le_B e$, положить $H'_B(e) = \varnothing$.
6. $L = \{(x, y, \alpha) \in R \cup F\}$, $N = \varnothing$.
7. Выбрать из списка L первое ребро (x, y, α), занести x и y в N, удалить ребро (x, y, α) из L.
8. Для всех вершин $z, y' \in N$ проверить возможность применения правила *flow* на четверке вершин x, y, y', z с использованием ребер (x, y, α) и (z, y', β) и правил *post*, *pass*, *find* на тройке вершин x, y, z с использованием ребра (x, y, α). Новые ребра вида (a, b, γ), где $\{a, b\} \subset \{x, y, y', z\}$, $\gamma \in \{write_m, write_t\}$, получающиеся в зависимости от правила преобразования состояния БД ДП-модели и принадлежности субъектов к множествам L_S и N_S, занести в конец L и F^a.
9. Если L не пусто, перейти на шаг 6.

Теорема 7. Алгоритм 6 корректно строит *access*-замыкание графа доступов.

Доказательство теоремы выполняется аналогично доказательству теоремы 3 с учетом правил преобразования БД ДП-модели. ■

2.4. Построение замыканий графа доступов ФАС ДП-модели

Перечислим основные положения ФАС ДП-модели:

- сущность $e \in E$ называется функционально-ассоциированной с субъектом $s \in S$ в состоянии G, если данные в сущности e влияют на вид преобразования данных, реализуемого субъектом s в состоянии G;
- $[s] \subset E$ – множество всех сущностей, функционально-ассоциированных с субъектом s, при этом выполняется условие $s \in [s]$;
- только информационный поток по памяти к сущности, функционально-ассоциированной с субъектом, приводит к изменению вида преобразования данных, реализуемого этим субъектом;
- функционально-ассоциированными с субъектом являются сущности, от которых зависит вид преобразования данных, реализуемого субъектом в данном или некотором последующем состоянии системы $\Sigma(G^*,$

OP), множество сущностей, функционально-ассоциированных с субъектом, не изменяется в процессе функционирования системы;

- если субъект реализовал информационный поток по памяти от себя к сущности, функционально-ассоциированной с другим субъектом, то первый субъект получает право доступа владения ко второму субъекту (правило *control*(), табл. 4.1).

В соответствии с данными положениями модифицируются условия и результаты применения правил преобразования состояний БК ДП-модели [3, табл. 2.1].

Алгоритм 8 построения *control*-замыкания графа доступов.

1. $L = \{(y, z): z \in [y], y \in S, z \in E\}$.
2. Выбрать из списка L первую пару (y, z) . Удалить пару (y, z) из L.
3. Для всех вершин $x \in S$ применить правило *control*(x, y, z). Если в результате этого появятся новые ребра вида (x, y, own_r), то занести их в $R^{control}$.
4. Применить алгоритм 6 к графу $G^{control} = (S, E, R^{control} \cup A^{control} \cup F^{control}, H)$.
5. Если L не пусто, перейти на шаг 2.

Теорема 8. Алгоритм 8 корректно строит *control*-замыкание графа доступов.

Доказательство теоремы выполняется аналогично доказательству теоремы 3 с учетом правил преобразования ФАС ДП-модели. ■

3. Анализ путей распространения прав доступа и информационных потоков

Пусть $G_0 = (S_0, E_0, R_0 \cup A_0 \cup F_0, H_0)$ – начальное состояние системы $\Sigma(G^*, OP)$ и $\alpha \in R_{raf}$, $x, y \in E_0$, где $x \neq y$. Пусть существуют состояния $G_1, \ldots, G_N = (S_N, E_N, R_N \cup A_N \cup F_N, H_N)$ и правила преобразования состояний $op_1, \ldots, op_N$ из OP такие, что $G_0 \vdash_{op1} G_1 \vdash_{op2} \ldots \vdash_{opN} G_N$ и $(x, y, \alpha) \in R_N \cup A_N \cup F_N$, где $N \geq 0$. Обозначим множество ребер, к которым применялись правила преобразования состояний $B \subseteq R_N \cup A_N \cup F_N$, а множество применяемых правил – OP_B. Пусть $P_{pr} \subseteq B \times OP_B$ и $P_{ps} \subseteq OP_B \times B$. *Графом анализа* системы $\Sigma(G^*, OP)$ в состоянии G_N будем называть ориентированный граф $G^A(x, y, \alpha) = (B \cup OP_B, P_{pr} \cup P_{ps})$. Условия применения правил, соответствующие условиям отношения иерархии сущностей системы, в граф анализа включать не будем.

Алгоритм 9 построения графа G^A при известной последовательности правил преобразования состояний $op_1, \ldots, op_N$.

1. $L = \{op_1, \ldots, op_N\}$, $OP_B = L$, $P_{pr} \cup P_{ps} = \varnothing$, $B = \varnothing$, $j = 0$, $j = j + 1$.
2. Выбрать из списка L первое правило op. Удалить правило op из L.
3. Если ребра $r_1, \ldots, r_m$ графов G_i для $i = 0, \ldots, j-1$ необходимы для применения правила op, а ребра $p_1, \ldots, p_l$ появляются после его применения, то $B = B \cup \{r_1, \ldots, r_m\} \cup \{p_1, \ldots, p_l\}$, $P_{pr} = P_{pr} \cup \{(r_1, op), \ldots, (r_m, op)\}$, $P_{ps} = P_{ps} \cup \{(op, p_1), \ldots, (op, p_l)\}$.
4. Если L не пусто, перейти на шаг 1.

Рассмотрим следующую задачу. Пусть $G_0 = (S_0, E_0, R_0 \cup A_0 \cup F_0, H_0)$ – начальное состояние системы $\Sigma(G^*, OP)$ и $\alpha \in R_{raf}$, $x, y \in E_0$, где $x \neq y$, и пусть в некотором состоянии системы $\Sigma(G^*, OP)$ выполнено условие: $(x, y, \alpha) \in R \cup A \cup F$. Требуется найти все такие правила преобразования состояний $op_1, \ldots, op_N$, что $G_0 \vdash_{op1} G_1 \vdash_{op2} \ldots \vdash_{opN} G_N$ и $(x, y, \alpha) \in R_N \cup A_N \cup F_N$ для некоторого $N \geq 0$. Для нахождения всех таких правил преобразования состояний ДП-моделей необходимо сначала преобразовать граф доступов в его замыкание, а затем с помощью следующих алгоритмов построить граф анализа $G^A(x, y, \alpha)$. Для краткости не будем писать элементы, добавляемые в множество B, так как их легко определить по ребрам в P_{pr}. Алгоритмы 10 – 13 строят граф $G^A(x, y, \alpha)$, который содержит всевозможные траектории получения прав доступа, доступов и реализации информационных потоков соответственно для базовой, БК, БД и ФАС ДП-моделей.

Алгоритм 10.

1. $OP_B = \varnothing$, $P_{pr} \cup P_{ps} = \varnothing$, $B = (x, y, \alpha)$.
2. Если $(x, y, \alpha) \in R_0 \cup A_0 \cup F_0$, то выход.
3. Если $\alpha \in R_r$:
   - 3.1. Если $\alpha = own_r$, существует $z \in E \setminus S$, что $y \in H(z)$, $(x, z, \beta) \in R$, $\beta \in \{write_r, append_r\}$, и для всех $e \in E$, что $y \leq e$, выполнено $(x, e, write_l) \in F$, то $OP_B = OP_B \cup create\_entity(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), create\_entity(x, y, z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, y, z), (x, y, \alpha))$.
   - 3.2. Если $\alpha = own_r$, $y \in S$, $y \in H(x)$, $(y, x, write_l) \in F$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, и для всех $e \in E$, что $z \leq e$, выполнено $(x, e, write_l) \in F$, положить $OP_B = OP_B \cup create\_subject(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z, y), (x, y, \alpha))$.
   - 3.3. Если $\alpha \neq own_r$ и $(x, y, own_r) \in R$, то $OP_B = OP_B \cup own\_take(\alpha, x, y)$, $P_{pr} = P_{pr} \cup ((x, y, own_r), own\_take(\alpha, x, y))$, $P_{ps} = P_{ps} \cup (own\_take(\alpha, x, y), (x, y, \alpha))$.
   - 3.4. Для всех $z \in S$, таких, что (x, z, own_r), $(z, y, \alpha) \in R$, положить $OP_B = OP_B \cup take\_right(\alpha, x, z, y)$, $P_{pr} = P_{pr} \cup ((x,z,own_r), take\_right(\alpha,x,z,y)) \cup ((z, y, \alpha), take\_right(\alpha, x, z, y))$, $P_{ps} = P_{ps} \cup (take\_right(\alpha, x, z, y), (x, y, \alpha))$.

3.5. Для всех $z \in S$, таких, что (z, x, own_r), $(z, y, \alpha) \in R$, положить $OP_B = OP_B \cup grant\_right(\alpha, z, x, y)$, $P_{pr} = P_{pr} \cup ((z, y, own_r), grant\_right(\alpha, z, x, y)) \cup ((z, y, \alpha), grant\_right(\alpha, z, x, y))$, $P_{ps} = P_{ps} \cup (grant\_right(\alpha, z, x, y), (x, y, \alpha))$.

4. Если $\alpha \in R_a$:
   4.1. Если $\alpha = read_a$ и $(x, y, read_r) \in R$, то $OP_B = OP_B \cup access\_read(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, read_r), access\_read(x, y))$, $P_{ps} = P_{ps} \cup (access\_read(x, y), (x, y, \alpha))$.
   4.2. Если $\alpha = write_a$ и $(x, y, write_r) \in R$, то $OP_B = OP_B \cup access\_write(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), access\_write(x, y))$, $P_{ps} = P_{ps} \cup (access\_write(x, y), (x, y, \alpha))$.
   4.3. Если $\alpha = append_a$ и $(x, y, append_r) \in R$, то $OP_B = OP_B \cup access\_append(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, append_r), access\_append(x, y))$, $P_{ps} = P_{ps} \cup (access\_append(x, y), (x, y, \alpha))$.
5. Если $\alpha = write_m$:
   5.1. Если $(x, y, read_a) \in A$, то $OP_B = OP_B \cup access\_read(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, read_r), access\_read(x, y))$, $P_{ps} = P_{ps} \cup (access\_read(x, y), (x, y, \alpha))$.
   5.2. Если $(x, y, write_a) \in A$, то $OP_B = OP_B \cup access\_write(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), access\_write(x, y))$, $P_{ps} = P_{ps} \cup (access\_write(x, y), (x, y, \alpha))$.
   5.3. Если $(x, y, append_a) \in A$, то $OP_B = OP_B \cup access\_append(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, append_r), access\_append(x, y))$, $P_{ps} = P_{ps} \cup (access\_append(x, y), (x, y, \alpha))$.
   5.4. Для всех $z \in S$, таких, что (x, z, β), $(z, y, \gamma) \in R \cup F$ и $\beta, \gamma \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup find(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), find(x, z, y)) \cup ((z, y, \gamma), find(x, z, y))$, $P_{ps} = P_{ps} \cup (find(x, z, y), (x, y, \alpha))$.
   5.5. Для всех $z \in E$, таких, что (x, z, β), $(y, z, read_r) \in R \cup F$ и $\beta \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup post(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), post(x, z, y)) \cup ((y, z, read_r), post(x, z, y))$, $P_{ps} = P_{ps} \cup (post(x, z, y), (x, y, \alpha))$.
   5.6. Для всех $z \in S$, таких, что $(z, x, read_r) \in R \cup F$, $(z, y, \beta) \in R \cup F$ и $\beta \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup pass(x, z, y)$, $P_{pr} = P_{pr} \cup ((z, x, read_r), pass(x, z, y)) \cup ((z, y, \beta), pass(x, z, y))$, $P_{ps} = P_{ps} \cup (pass(x, z, y), (x, y, \alpha))$.
6. Если $\alpha = write_t$:
   6.1. Для всех $z \in E$, таких, что $(x, z, read_a) \in A$ и $z \le y$, положить $OP_B = OP_B \cup access\_read(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, read_r), access\_read(x, z))$, $P_{ps} = P_{ps} \cup (access\_read(x, z), (x, y, \alpha))$.
   6.2. Для всех $z \in E$, таких, что $(x, z, write_a) \in A$ и $z \le y$, положить $OP_B = OP_B \cup access\_write(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, write_r), access\_write(x, z))$, $P_{ps} = P_{ps} \cup (access\_write(x, z), (x, y, \alpha))$.
   6.3. Для всех $z \in E$, таких, что $(x, z, append_a) \in A$ и $z \le y$, положить $OP_B = OP_B \cup access\_append(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, append_r), access\_append(x, z))$, $P_{ps} = P_{ps} \cup (access\_append(x, z), (x, y, \alpha))$.
   6.4. Если $(y, x, own_r) \in R$, то для всех $z \in E$, таких, что $R_x \cap R_y \neq \varnothing$, где $R_y = \{\beta: (y, z, \beta) \in R\}$, $R_x = \{\beta: (x, z, \beta) \in R\}$, положить $OP_B = OP_B \cup take\_right(y, x, z)$, $P_{pr} = P_{pr} \cup ((y, x, own_r), take\_right(y, x, z))$, $P_{ps} = P_{ps} \cup (take\_right(y, x, z), (x, y, \alpha))$.
   6.5. Если $(x, y, own_r) \in R$, то для всех $z \in E$, таких, что $R_x \cap R_y \neq \varnothing$, где $R_y = \{\beta: (y, z, \beta) \in R\}$, $R_x = \{\beta: (x, z, \beta) \in R\}$, положить $OP_B = OP_B \cup grant\_right(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, y, own_r), grant\_right(x, y, z))$, $P_{ps} = P_{ps} \cup (grant\_right(x, y, z), (x, y, \alpha))$.
   6.6. Если $(x, y, write_r) \in R$, то для всех $z \in H(y)$, таких, что $\{(x, e, write_t): e \in E, x \neq e$ и $e \le z\} \cup \{(x, s, write_t): s \in S, x \neq s$ и $(s, e, \alpha_r) \in R$, где $e \in E, e \le z$ и $\alpha_r \in R_r\} \subseteq F$, положить $OP_B = OP_B \cup rename\_entity(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), rename\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, z, y), (x, y, \alpha))$.
   6.7. Для всех $e \in E$, таких, что $y \le e$, $e \in H(z)$, $(x, z, write_r) \cup (x, z, write_t) \cup \{(x, s, write_t): s \in S, x \neq s$ и $(s, e', \alpha_r) \in R$, где $e' \in E, e' \le e$ и $\alpha_r \in R_r\} \subseteq R \cup F$, положить $OP_B = OP_B \cup rename\_entity(x, e, z)$, $P_{pr} = P_{pr} \cup ((x, z, write_r), rename\_entity(x, e, z))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, e, z), (x, y, \alpha))$.
   6.8. Если $y \in S$, то для всех $z, z' \in E$, таких, что $z \in H(z')$, $(x, z', write_r)$, $(x, z', write_t) \in R \cup F$, если $e \le z$ и $(y, e, \alpha_r) \in R$, где $e \in E$, $\alpha_r \in R_r$, то положить $OP_B = OP_B \cup rename\_entity(x, z, z')$, $P_{pr} = P_{pr} \cup ((x, z', write_r), rename\_entity(x, z, z')) \cup ((y, e, \alpha_r), rename\_entity(x, z, z'))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, z, z'), (x, y, \alpha))$.
   6.9. Для всех $z, z' \in E$, что $z \in H(z')$, $(x, z, own_r) \in R$, $(x, z', \beta) \in R$, $\beta \in \{write_r, append_r\}$ и $z \le y$, положить $OP_B = OP_B \cup create\_entity(x, z', z)$, $P_{pr} = P_{pr} \cup ((x, y, \beta), create\_entity(x, z', z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z', z), (x, y, \alpha))$.
   6.10. Для всех $z, z' \in E$, что $(x, z, execute_r) \in R$, $z' \in H(x)$, $(x, z', own_r) \in R$ и $z \le y$, положить $OP_B = OP_B \cup create\_subjecty(x, z, z')$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_subject(x, z, z'))$, $P_{ps} = P_{ps} \cup (create\_subject(x, z, z'), (x, y, \alpha))$.
   6.11. Если $y \in H(x)$, $(x, y, own_r) \in R$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, положить $OP_B = OP_B \cup create\_subjecty(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_subject(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_subject(x, z, y), (x, y, \alpha))$.

6.12. Если $y \in S$, $(y, x, write_t) \in F$, то для всех $z, z' \in E$, что $z \le z'$ (x, z, α_r), $(y, z', \beta_r) \in R$, где $\alpha_r, \beta_r \in R_r$, положить $OP_B = OP_B \cup flow(x, z, z', y)$, $P_{pr} = P_{pr} \cup ((x, z, \alpha_r), flow(x, z, z', y)) \cup ((y, z', \beta_r), flow(x, z, z', y))$, $P_{ps} = P_{ps} \cup (flow(x, z, z', y), (x, y, \alpha))$.

6.13. Если $y \in S$, $(y, x, write_t) \in F$, то для всех $z, z' \in E$, что $z \le z'$ (y, z, α_r), $(x, z', \beta_r) \in R$, где $\alpha_r, \beta_r \in R_r$, положить $OP_B = OP_B \cup flow(x, z, z', y)$, $P_{pr} = P_{pr} \cup ((x, z', \alpha_r), flow(x, z, z', y)) \cup ((y, z, \beta_r), flow(x, z, z', y))$, $P_{ps} = P_{ps} \cup (flow(x, z, z', y), (x, y, \alpha))$.

6.14. Для всех $z \in S$, если (x, z, γ), $(z, y, \beta) \in F$, $write_t \in \{\gamma, \beta\}$, то $OP_B = OP_B \cup find(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \gamma), find(x, z, y)) \cup ((z, y, \beta), find(x, z, y))$, $P_{ps} = P_{ps} \cup (find(x, z, y), (x, y, \alpha))$.

6.15. Для всех $z \in E$, если $(x, z, write_t)$, $(y, z, read_r) \in R \cup F$, то $OP_B = OP_B \cup post(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, write_t), post(x, z, y)) \cup ((y, z, read_r), post(x, z, y))$, $P_{ps} = P_{ps} \cup (post(x, z, y), (x, y, \alpha))$.

6.16. Для всех $z \in S$, если $(z, x, read_r)$, $(z, y, \beta) \in R \cup F$, где $\beta \in \{write_r, append_r, write_m, write_t\}$, то $OP_B = OP_B \cup pass(x, z, y)$, $P_{pr} = P_{pr} \cup ((z, x, read_r), pass(x, z, y)) \cup ((z, y, \beta), pass(x, z, y))$, $P_{ps} = P_{ps} \cup (pass(x, z, y), (x, y, \alpha))$.

7. Для всех элементов вида $((a, b, \beta), op)$, добавленных на шагах 3 – 6 алгоритма в множество P_{pr} выполнить шаги 2–7 алгоритма 10 для (x, y, α), где $x = a$, $y = b$, $\alpha = \beta$.

Алгоритм 11 построения графа анализа $G^A(x, y, \alpha)$ для БК ДП-модели.

1. $OP_B = \emptyset$, $P_{pr} \cup P_{ps} = \emptyset$, $B = (x, y, \alpha)$.
2. Если $(x, y, \alpha) \in R_0 \cup A_0 \cup F_0$, то выход.
3. Если $x \in N_S$, выполнить шаги 2 – 6 алгоритма 10.
4. Если $x \in L_S$, $\alpha \in R_r$:
   4.1. Если $\alpha = own_r$ и существует $z \in E \backslash S$, что $y \in H(z)$, $(x, z, \beta) \in R$, $\beta \in \{write_r, append_r\}$, то $OP_B = OP_B \cup create\_entity(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), create\_entity(x, y, z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, y, z), (x, y, \alpha))$.
   4.2. Если $\alpha = own_r$, $y \in S$, $y \in H(x)$, $(y, x, write_t) \in F$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, положить $OP_B = OP_B \cup create\_subject(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z, y), (x, y, \alpha))$.
   4.3. Если $\alpha \ne own_r$, то $OP_B = OP_B \cup own\_take(\alpha, x, y)$, $P_{pr} = P_{pr} \cup ((x, y, own_r), own\_take(\alpha, x, y))$, $P_{ps} = P_{ps} \cup (own\_take(\alpha, x, y), (x, y, \alpha))$.
5. Если $x \in L_S$, $\alpha \in R_a$:
   5.1. Если $\alpha = read_a$ и $(x, y, read_r) \in R$, то $OP_B = OP_B \cup access\_read(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, read_r), access\_read(x, y))$, $P_{ps} = P_{ps} \cup (access\_read(x, y), (x, y, \alpha))$.
   5.2. Если $\alpha = write_a$ и $(x, y, write_r) \in R$, то $OP_B = OP_B \cup access\_write(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), access\_write(x, y))$, $P_{ps} = P_{ps} \cup (access\_write(x, y), (x, y, \alpha))$.
   5.3. Если $\alpha = append_a$ и $(x, y, append_r) \in R$, то $OP_B = OP_B \cup access\_append(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, append_r), access\_append(x, y))$, $P_{ps} = P_{ps} \cup (access\_append(x, y), (x, y, \alpha))$.
6. Если $x \in L_S$, если $\alpha = write_m$:
   6.1. Если $(x, y, read_a) \in A$, то $OP_B = OP_B \cup access\_read(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, read_r), access\_read(x, y))$, $P_{ps} = P_{ps} \cup (access\_read(x, y), (x, y, \alpha))$.
   6.2. Если $(x, y, write_a) \in A$, то $OP_B = OP_B \cup access\_write(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), access\_write(x, y))$, $P_{ps} = P_{ps} \cup (access\_write(x, y), (x, y, \alpha))$.
   6.3. Если $(x, y, append_a) \in A$, то $OP_B = OP_B \cup access\_append(x, y)$, $P_{pr} = P_{pr} \cup ((x, y, append_r), access\_append(x, y))$, $P_{ps} = P_{ps} \cup (access\_append(x, y), (x, y, \alpha))$.
   6.4. Для всех $z \in S$, таких, что (x, z, β), $(z, y, \gamma) \in R \cup F$ и $\beta, \gamma \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup find(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), find(x, z, y)) \cup ((z, y, \gamma), find(x, z, y))$, $P_{ps} = P_{ps} \cup (find(x, z, y), (x, y, \alpha))$.
   6.5. Для всех $z \in E$, таких, что (x, z, β), $(y, z, read_r) \in R \cup F$ и $\beta \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup post(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), post(x, z, y)) \cup ((y, z, read_r), post(x, z, y))$, $P_{ps} = P_{ps} \cup (post(x, z, y), (x, y, \alpha))$.
   6.6. Для всех $z \in S$, таких, что $(z, x, read_r) \in R \cup F$, $(z, y, \beta) \in R \cup F$ и $\beta \in \{write_r, append_r, write_m\}$, положить $OP_B = OP_B \cup pass(x, z, y)$, $P_{pr} = P_{pr} \cup ((z, x, read_r), pass(x, z, y)) \cup ((z, y, \beta), pass(x, z, y))$, $P_{ps} = P_{ps} \cup (pass(x, z, y), (x, y, \alpha))$.
7. Для всех элементов вида $((a, b, \beta), op)$, добавленных на шагах 3 – 6 алгоритма 11 в множество P_{pr}, выполнить шаги 2–7 алгоритма 11 для (x, y, α), где $x = a$, $y = b$, $\alpha = \beta$.

Алгоритм 12 построения графа анализа $G^A(x, y, \alpha)$ для БК ДП-модели.

1. $OP_B = \emptyset$, $P_{pr} \cup P_{ps} = \emptyset$, $B = (x, y, \alpha)$.

2. Если $(x, y, \alpha) \in R_0 \cup A_0 \cup F_0$, то выход.
3. Если $x \in L_S$, $\alpha \in R_r$:
   3.1. Если $\alpha = own_r$ и существует $z \in E \setminus S$, что $y \in H_B(z)$, $(x, z, \beta) \in R$, $\beta \in \{write_r, append_r\}$, то $OP_B = OP_B \cup create\_entity(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), create\_entity(x, y, z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, y, z), (x, y, \alpha))$.
   3.2. Если $\alpha = own_r$, $y \in S$, $y \in H_B(x)$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, положить $OP_B = OP_B \cup create\_subject(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z, y), (x, y, \alpha))$.
   3.3. Если $\alpha \neq own_r$, то $OP_B = OP_B \cup own\_take(\alpha, x, y)$, $P_{pr} = P_{pr} \cup ((x, y, own_r), own\_take(\alpha, x, y))$, $P_{ps} = P_{ps} \cup (own\_take(\alpha, x, y), (x, y, \alpha))$.
4. Если $x \in L_S$, $\alpha \in R_a$, выполнить шаг 5 алгоритма 11.
5. Если $x \in L_S$, если $\alpha = write_m$:
   5.1. Выполнить шаги 6.1 – 6.3 алгоритма 11.
   5.2. Для всех $z \in S$, таких, что $(x, z, write_m)$, $(z, y, write_m) \in F$, положить $OP_B = OP_B \cup find(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, write_m), find(x, z, y)) \cup ((z, y, write_m), find(x, z, y))$, $P_{ps} = P_{ps} \cup (find(x, z, y), (x, y, \alpha))$.
   5.3. Для всех $z \in E$, таких, что $(x, z, write_m)$, $(y, z, read_a) \in A$, положить $OP_B = OP_B \cup post(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), post(x, z, y)) \cup ((y, z, read_a), post(x, z, y))$, $P_{ps} = P_{ps} \cup (post(x, z, y), (x, y, \alpha))$.
   5.4. Для всех $z \in S$, таких, что $(z, x, read_a) \in A$, $(z, y, write_m) \in F$, положить $OP_B = OP_B \cup pass(x, z, y)$, $P_{pr} = P_{pr} \cup ((z, x, read_a), pass(x, z, y)) \cup ((z, y, write_m), pass(x, z, y))$, $P_{ps} = P_{ps} \cup (pass(x, z, y), (x, y, \alpha))$.
6. Если $x \in N_S$, $\alpha \in R_r$:
   6.1. Если $\alpha = own_r$, существует $z \in E \setminus S$, что $y \in H_B(z)$, $(x, z, \beta) \in R$, $\beta \in \{write_r, append_r\}$, и для всех $e \in E \setminus E_B$, что $y \leq_B e$, выполнено $(x, e, write_t) \in F$, то $OP_B = OP_B \cup create\_entity(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, z, \beta), create\_entity(x, y, z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, y, z), (x, y, \alpha))$.
   6.2. Если $\alpha = own_r$, $y \in S$, $y \in H_B(x)$, $(y, x, write_t) \in F$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, и для всех $e \in E \setminus E_B$, что $z \leq_B e$, выполнено $(x, e, write_t) \in F$, положить $OP_B = OP_B \cup create\_subject(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z, y), (x, y, \alpha))$.
   6.3. Выполнить шаги 3.3 – 3.5 алгоритма 10.
7. Выполнить шаг 4 алгоритма 10.
8. Если $x \in N_S$, $\alpha = write_m$, выполнить шаги 5.1 – 5.3 алгоритма 10 и 5.2 – 5.4 алгоритма 12.
9. Если $x \in N_S$, $\alpha = write_t$:
   9.1. Для всех $z \in E$, таких, что $(x, z, read_a) \in A$ и $y \in E \setminus E_B$, $x \neq y$, $z \leq_B y$, положить $OP_B = OP_B \cup access\_read(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, read_r), access\_read(x, z))$, $P_{ps} = P_{ps} \cup (access\_read(x, z), (x, y, \alpha))$.
   9.2. Для всех $z \in E$, таких, что $(x, z, write_a) \in A$ и $y \in E \setminus E_B$, $x \neq y$, $z \leq_B y$, положить $OP_B = OP_B \cup access\_write(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, write_r), access\_write(x, z))$, $P_{ps} = P_{ps} \cup (access\_write(x, z), (x, y, \alpha))$.
   9.3. Для всех $z \in E$, таких, что $(x, z, append_a) \in A$ и $y \in E \setminus E_B$, $x \neq y$, $z \leq_B y$, положить $OP_B = OP_B \cup access\_append(x, z)$, $P_{pr} = P_{pr} \cup ((x, z, append_r), access\_append(x, z))$, $P_{ps} = P_{ps} \cup (access\_append(x, z), (x, y, \alpha))$.
   9.4. Выполнить шаги 6.4 – 6.5 алгоритма 10.
   9.5. Если $(x, y, write_r) \in R$, то для всех $z \in H(y)$, таких, что $\{(x, e, write_t)$: $e \in E \setminus E_B$, $x \neq e$ и $e \leq_B z\} \cup \{(x, s, write_t)$: $s \in S$, $x \neq s$ и $(s, e, \alpha_r) \in R$, где $e \in E \setminus E_B$, $e \leq_B z$ и $\alpha_r \in R_r\} \subseteq F$, положить $OP_B = OP_B \cup rename\_entity(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, y, write_r), rename\_entity(x, z, y))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, z, y), (x, y, \alpha))$.
   9.6. Для всех $e \in E$, таких, что $y \leq_B e$, $e \in H(z)$, $(x, z, write_r) \cup (x, z, write_t) \cup \{(x, s, write_t)$: $s \in S$, $x \neq s$ и $(s, e', \alpha_r) \in R$, где $e' \in E \setminus E_B$, $e' \leq_B e$ и $\alpha_r \in R_r\} \subseteq R \cup F$, положить $OP_B = OP_B \cup rename\_entity(x, e, z)$, $P_{pr} = P_{pr} \cup ((x, z, write_r), rename\_entity(x, e, z))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, e, z), (x, y, \alpha))$.
   9.7. Если $y \in S$, то для всех $z, z' \in E$, таких, что $z \in H(z')$, $(x, z', write_r)$, $(x, z', write_t) \in R \cup F$, если $e \leq_B z$ и $(y, e, \alpha_r) \in R$, где $e \in E \setminus E_B$, $\alpha_r \in R_r$, то положить $OP_B = OP_B \cup rename\_entity(x, z, z')$, $P_{pr} = P_{pr} \cup ((x, z', write_r), rename\_entity(x, z, z')) \cup ((y, e, \alpha_r), rename\_entity(x, z, z'))$, $P_{ps} = P_{ps} \cup (rename\_entity(x, z, z'), (x, y, \alpha))$.
   9.8. Для всех $z, z' \in E$, что $z \in H_B(z')$, $(x, z, own_r) \in R$, $(x, z', \beta) \in R$, $\beta \in \{write_r, append_r\}$ и $z \leq_B y$, положить $OP_B = OP_B \cup create\_entity(x, z', z)$, $P_{pr} = P_{pr} \cup ((x, y, \beta), create\_entity(x, z', z))$, $P_{ps} = P_{ps} \cup (create\_entity(x, z', z), (x, y, \alpha))$.
   9.9. Для всех $z, z' \in E$, что $(x, z, execute_r) \in R$, $z' \in H(x)$, $(x, z', own_r) \in R$ и $z \leq_B y$, положить $OP_B = OP_B \cup create\_subjecty(x, z, z')$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_subject(x, z, z'))$, $P_{ps} = P_{ps} \cup (create\_subject(x, z, z'), (x, y, \alpha))$.
   9.10. Если $y \in H_B(x)$, $(x, y, own_r) \in R$, то для всех $z \in E$, что $(x, z, execute_r) \in R$, положить $OP_B = OP_B \cup create\_subjecty(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, execute_r), create\_subject(x, z, y))$, $P_{ps} = P_{ps} \cup (create\_subject(x, z, y), (x, y, \alpha))$.

9.11. Если $y \in S$, $(y, x, write_t) \in F$, то для всех $z, z' \in E$, что $z \le_B z'$, (x, z, α_r), $(y, z', \beta_r) \in R$, где $\alpha_r, \beta_r \in R_r$, положить $OP_B = OP_B \cup flow(x, z, z', y)$, $P_{pr} = P_{pr} \cup ((x, z, \alpha_r), flow(x, z, z', y)) \cup ((y, z', \beta_r), flow(x, z, z', y))$, $P_{ps} = P_{ps} \cup (flow(x, z, z', y), (x, y, \alpha))$.

9.12. Если $y \in S$, $(y, x, write_t) \in F$, то для всех $z, z' \in E$, что $z \le_B z'$, (y, z, α_r), $(x, z', \beta_r) \in R$, где $\alpha_r, \beta_r \in R_r$, положить $OP_B = OP_B \cup flow(x, z, z', y)$, $P_{pr} = P_{pr} \cup ((x, z', \alpha_r), flow(x, z, z', y)) \cup ((y, z, \beta_r), flow(x, z, z', y))$, $P_{ps} = P_{ps} \cup (flow(x, z, z', y), (x, y, \alpha))$.

9.13. Для всех $z \in S$, если (x, z, γ), $(z, y, \beta) \in F$, $\gamma, \beta \in \{write_t, write_m\}$, то $OP_B = OP_B \cup find(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, \gamma), find(x, z, y)) \cup ((z, y, \beta), find(x, z, y))$, $P_{ps} = P_{ps} \cup (find(x, z, y), (x, y, \alpha))$.

9.14. Для всех $z \in E$, если $(x, z, write_t)$, $(y, z, read_a) \in A \cup F$, то $OP_B = OP_B \cup post(x, z, y)$, $P_{pr} = P_{pr} \cup ((x, z, write_t), post(x, z, y)) \cup ((y, z, read_r), post(x, z, y))$, $P_{ps} = P_{ps} \cup (post(x, z, y), (x, y, \alpha))$.

9.15. Для всех $z \in S$, если $(z, x, read_a)$, $(z, y, \beta) \in A \cup F$, где $\beta \in \{write_r, append_r, write_m, write_t\}$, то $OP_B = OP_B \cup pass(x, z, y)$, $P_{pr} = P_{pr} \cup ((z, x, read_r), pass(x, z, y)) \cup ((z, y, \beta), pass(x, z, y))$, $P_{ps} = P_{ps} \cup (pass(x, z, y), (x, y, \alpha))$.

10. Для всех элементов вида $((a, b, \beta), op)$, добавленных на шагах 3 – 6 алгоритма 12 в множество P_{pr}, выполнить шаги 2 – 10 алгоритма 12 для (x, y, α), где $x = a$, $y = b$, $\alpha = \beta$.

Для построения графа анализа $G^A(x, y, \alpha)$ ФАС ДП-модели модифицируем алгоритм 11 построения графа анализа БК ДП-модели.

Алгоритм 13 построения графа анализа $G^A(x, y, \alpha)$ ФАС ДП-модели получается из алгоритма 11 добавлением в него следующего шага, выполняемого после шага 2:

3. Если $\alpha = own_r$, то для всех $z \in E$, $z \in [y]$, что $(x, z, write_m) \in R$, положить $OP_B = OP_B \cup control(x, y, z)$, $P_{pr} = P_{pr} \cup ((x, z, write_m), control(x, y, z))$, $P_{ps} = P_{ps} \cup (control(x, y, z), (x, y, \alpha))$.

4. Пример построения графа анализа ФАС ДП-модели

Рассмотрим сеть со следующей конфигурацией (рис. 2):

- на узле *Fw* установлена уязвимая версия *ssh*-службы, позволяющая получить права пользователя *root* через переполнение буфера;
- на узле *Ws* установлена уязвимая версия *web*-сервера, запущенного с привилегиями пользователя *apache*;
- пользователь *apache* имеет право доступа на чтение к базе данных *db*;
- в начальном состоянии нарушитель *A* на узле *Attacker* имеет доступ только к службе *ssh*, запущенной на *Fw*.

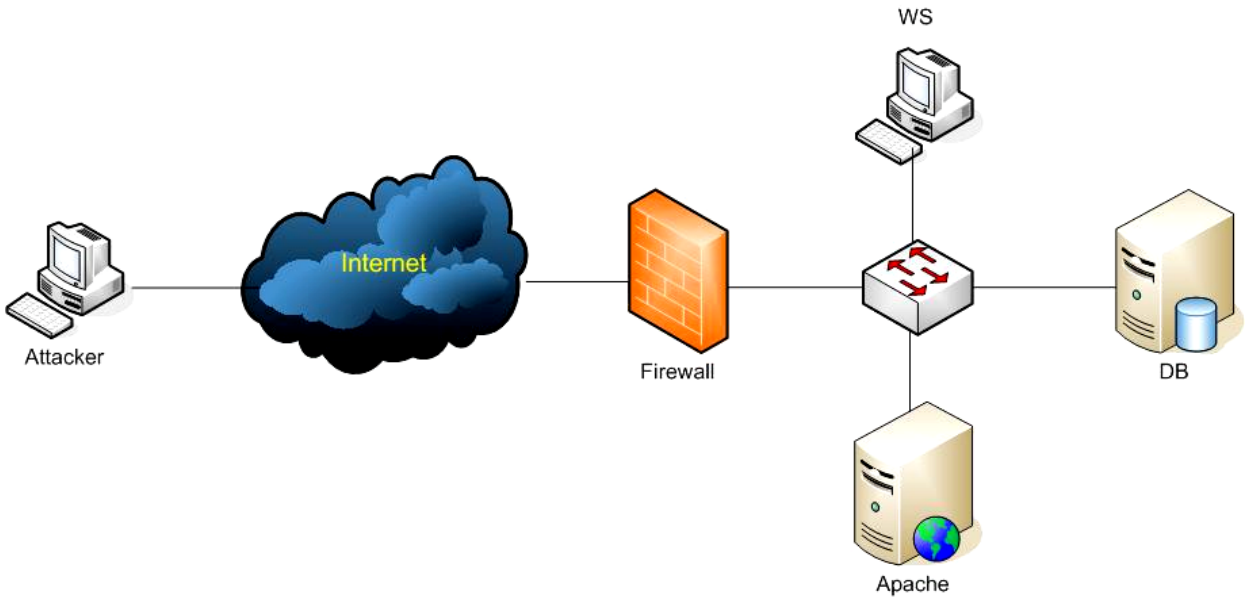


Рис. 2

На рис. 3 показана последовательность правил преобразований, в результате применения которой возможна утечка права доступа $(A, sw, read_r)$, после чего нарушитель может, используя уязвимость web-сервера, получить доступ к базе данных *db*. Далее строится граф анализа $G^A(A, sw, read_r)$ с помощью алгоритма 13. Результат представлен на рис. 4.

Примечание. Построение замыкания графа доступов на рисунках не изображено.

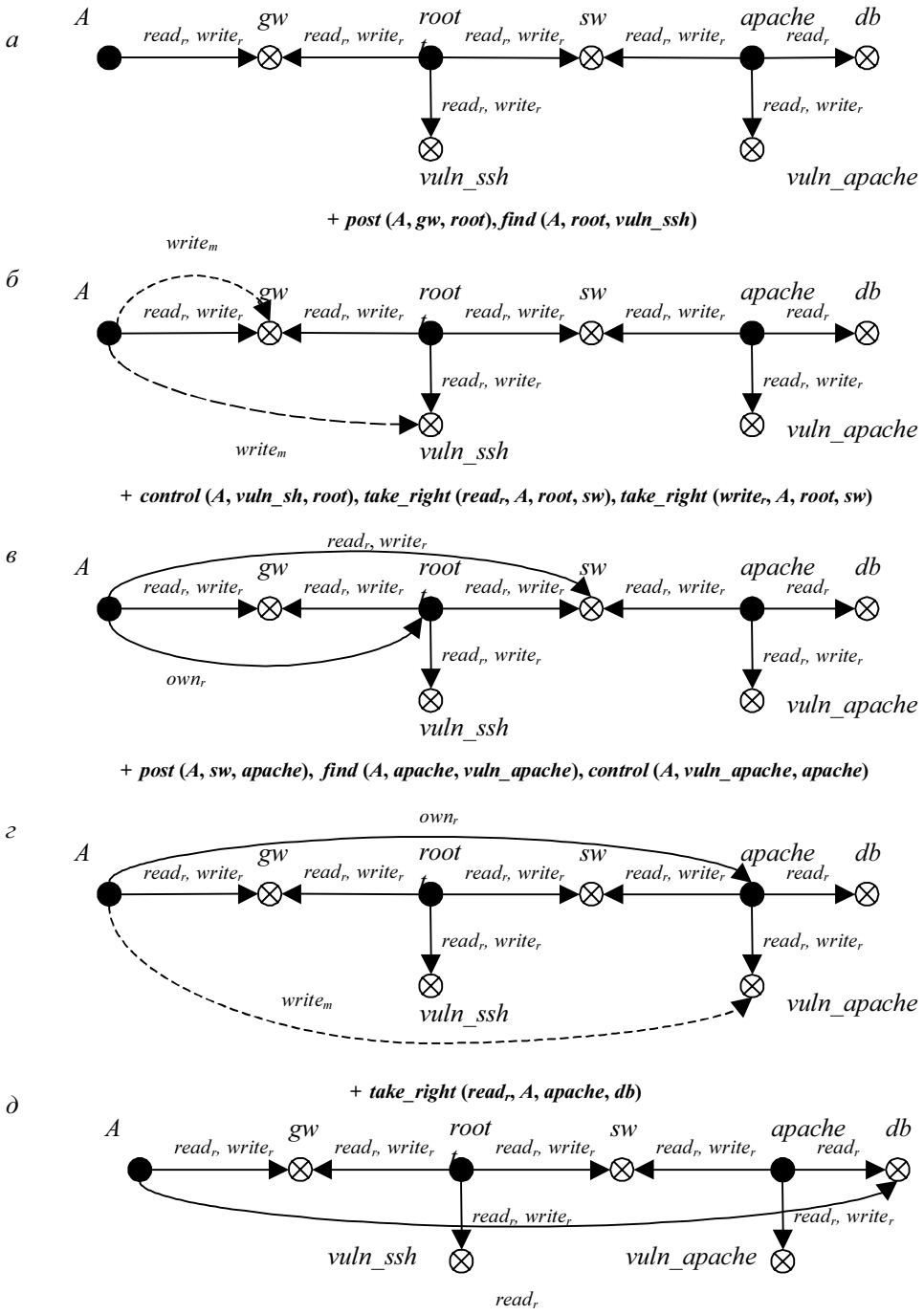


Рис. 3

5. Предотвращение утечки прав доступа и реализации запрещенных информационных потоков путем управления правами доступами в начальном состоянии ДП-модели

Пусть в системе $\Sigma(G^*, OP, G_0)$ возможна утечка права доступа (x, y, α) или реализация запрещенного информационного потока (x', y', α'). Тогда по определению возможен переход системы в состояние $G' = (S', E', R' \cup A' \cup F', H')$, в котором $(x, y, \alpha) \in R'$, $(x', y', \alpha') \in F'$, где $x \in S_0$, $x', y', y \in E_0$, $x \neq y$, $x' \neq y'$, $\alpha \in R_r$, $\alpha' \in R_f$ и $(x, y, \alpha) \notin R_0$, $(x', y', \alpha') \notin F_0$.

Формализуем и применим идею метода рекурсивного алгебраического анализа графов атак [6, 7] для предотвращения утечки прав доступа и реализации запрещенных информационных потоков.

Идея метода заключается в следующем. Правила преобразования, права доступа, доступы и информационные потоки интерпретируются логическими переменными. Взаимосвязь между ними характеризуется логическим выражением. Над логическими переменными, соответствующими правам доступа и информационным потокам исходного состояния некоторого правила преобразования, выполняется операция конъ-

юнкция. Над логическими переменными, соответствующими правилам преобразования, выполняется операция дизъюнкция, так как получение права доступа или реализация информационного потока возможно в результате применения нескольких правил преобразования. Такая последовательность действий производится рекурсивно, пока не будут достигнуты права доступа начального состояния. В результате проделанных действий будем иметь ДНФ некоторой булевой функции. Если ее приравнять к 0, найти все решения, то исключение прав доступа, соответствующих найденным решениям из множества R_0, позволяет предотвратить утечки прав доступа и реализации запрещенных информационных потоков.

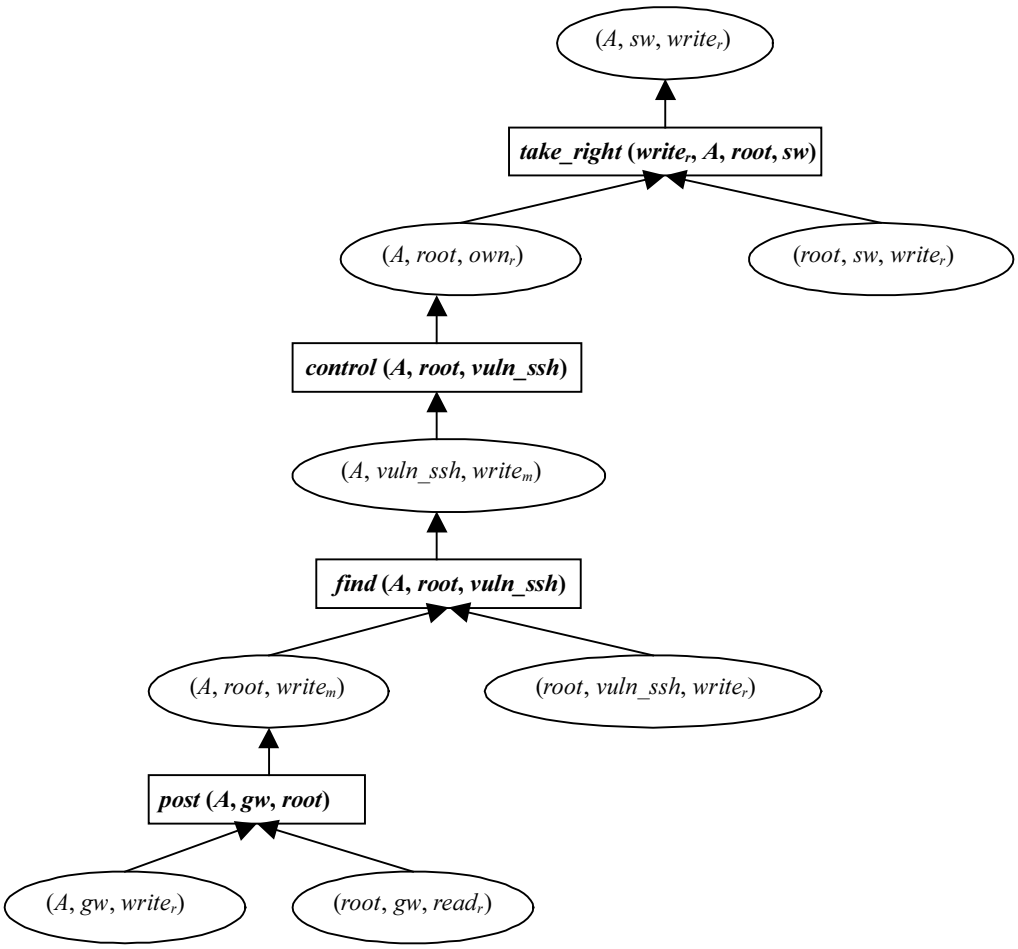


Рис. 4

Алгоритм 14.

1. Построить замыкание графа доступов ДП-модели и граф анализа $G^A(x, y, \alpha)$, вершину (x, y, α) обозначить c, положить $D = c$.
2. Ребра (op_1, c), …, (op_m, c) обозначить через $d_1,\ldots, d_m$, $L = \varnothing$.
3. Положить $c = d_1 \vee \ldots \vee d_m$.
4. Для всех $i = 1, \ldots, m$, ребра вида $((a_j, b_j, \alpha_j), op_i)$ обозначить c_{ij}, (a_j, b_j, α_j) добавить в L.
5. Каждое d_i в дизъюнкции c заменить на конъюнкцию $c_{i1}\, c_{i2\ldots}c_{in}$.
6. Изменить c в D.
7. Выполнить шаги 2–7 для всех $c \in L \backslash R_0$.
8. Решить уравнение $D = 0$ относительно переменных c_{ij}.

В качестве примера рассмотрим, какие права доступа необходимо исключить из R_0 для предотвращения утечки права $(A, sw, write_r)$ в рассмотренном выше примере (рис. 4).

В результате применения алгоритма 14 и переименования переменных получим уравнение $c_1c_2c_3c_4 = 0$, где c_1 соответствует праву $(root, sw, write_r)$, c_2 – $(root, vuln\_ssh, write_r)$, c_3 – $(root, gw, read_r)$, c_4 – $(A, gw, write_r)$. Соответственно получаем следующие возможные решения:

1) $c_1 = 0$, $c_3 = 0$ – отключение *ssh*-службы;
2) $c_2 = 0$ – устранение уязвимости *vuln_ssh*;
3) $c_3 = 0$ – настройка правил фильтрации МЭ.

6. Формальная модель анализа защищенности сетей REM

В работе [8] представлено неформальное описание модели анализа защищенности сетей REM (Requirements, Effects, Modifies), используемой в топологическом сканере безопасности *NetSPA*. Основными положениями данной модели являются:

- описание атаки состоит из трех частей – предусловий атаки (requirements), постусловий атаки (effects), координат нарушителя (modifies);
- при описании используются переменные *host*, *software*, *user*, которые соответственно принимают значения (<*current_host*>, <*target_host*>), (*vendor*: *product name*: *common version*: *exact version*), (*none*, *nologin*, *user*, *root*);
- предусловиями атак служат предикаты: $host_i$ RUNS (ALSO_RUNS) $software_j$ [DOS] [TROJAN] – установка на узле $host_i$ программного обеспечения $software_j$, которое может быть установлено нарушителем или находиться в неработоспособном состоянии; $host_i$ CAN_CONNECT $host_j$ – связь между узлами; [$user_i$ on $host_i$] HAS_ACCESS $user_j$ on $host_j$ – пользователь $user_i$ с узла $host_i$ имеет доступ на узле $host_j$ с правами $user_j$;
- изменения состояний сети описываются предикатами $host_i$ RUNS $software_j$, [$user_i$ on $host_i$] HAS_ACCESS $user_j$ on $host_j$, TRUST_RELATIONSHIP – перехват данных в сети;
- изменения состояния нарушителя соответствуют изменениям <*current_host*> = <*target_host*>, <*current_user*> = <*target_user*>.

Опишем формальную REM ДП-модель анализа защищенности сетей на основе базовой и ФАС ДП-моделей.

Пусть $G = (S, E, R \cup A \cup F, H)$ – состояние системы $\Sigma(G^*, OP)$. Конкретизируем назначение некоторых элементов системы.

По определению являются: узел (компьютер) – сущностью-контейнером; коммуникационный порт – сущностью-контейнером или сущностью-объектом; файл – либо сущностью-контейнером, либо сущностью-объектом; пользователь или процесс пользователя – сущностью-субъектом.

Обозначим: E – множество сущностей системы, EC – множество узлов КС, CC – множество коммуникационных каналов, $CC \subset E$, $EC \subset E$; S – множество субъектов системы, при этом для каждого узла $c \in EC$ существует «пользователь ОС» $os_c \in S$ данного компьютера, U – множество пользователей, $U \subset S$;

R – множество ребер графа-состояния G, соответствующих правам доступа пользователей к сущностям;

A – множество ребер графа-состояния G, соответствующих доступам пользователей к сущностям;

F – множество ребер графа-состояния G, соответствующих информационным потокам между сущностями;

H: $E \to 2^E$ – функция иерархии сущностей, ее значения на множестве узлов EC соответствуют заданной в системе иерархии подчиненности компьютеров, и каждая сущность системы $e \in E$

- либо является узлом сети ($e \in EC$),
- либо размещена на некотором единственном для каждой сущности узле (для сущности e существует единственный компьютер $c \in EC$, такой, что $e < c$),
- либо является коммуникационным каналом ($c \in EC$),
- либо является пользователем ($e \in S$).

Аналогично [3] будем считать, что для каждого узла $c_1 \in EC$ определены доверенные пользователи $os_{c1} \in S$, обладающие правом доступа владения к каждой сущности, размещенной на данном компьютере.

В соответствии с положениями модели REM [8]:

- субъект-процесс p, выполняющийся от имени субъекта-пользователя u, будем считать функционально ассоциированным с ним, так как при наличии уязвимостей p нарушитель может получить права доступа пользователя u ко всем сущностям КС;
- наличие уязвимостей определяется по значению переменной *software* = (*vendor*: *product name*: *common version*: *exact version*) и дополнительно установленного программного обеспечения;
- уязвимые сетевые сервисы, позволяющие получать доступ к чувствительной информации (например, через нулевые сессии в ОС класса Windows), функционально ассоциированы с субъектом *nologin*;
- субъект *nologin* может быть функционально ассоциированным с субъектом-пользователем, информацию о котором он предоставляет.

Для состояния $G = (S, E, R \cup A \cup F, H)$ системы $\Sigma(G^*, OP)$ формально определим предикаты *run_soft*(x, y, G), *run_soft_dos*(x, y, G), *can_access_net*(x, y, z, v, G), *can_connect*(x, y, z, v,G), соответствующие предикатам RUNS (ALSO_RUNS), RUNS_DOS, HAS_ACCESS, CAN_CONNECT модели REM.

Предикат *run_soft*(x, y, G) является истинным тогда и только тогда, когда $y < x$ в состоянии G, где $y \in S$, $x \in EC$. При этом предполагается, что на узле x существует некоторый пользователь $u \in S$, что (u, y, own_r)$\in R$.

Предикат *can_connect*(x, y, z, v, G) является истинным тогда и только тогда, когда в состоянии G существует $c \in CC$, что (y, c, α_r)$\in R$, (v, c, α_r)$\in R$, где $\alpha_r \in \{read_r, write_r\}$, $y < x$, $v < z$, y, $v \in S$, x, $z \in EC$.

Предикат *run_soft_dos*(x, y, G) является истинным тогда и только тогда, когда предикат *run_soft*(x, y, G) истинен, а предикат *can_connect*(x, y, z, v, G) ложен для любых $v \in S$, $z \in EC$.

Предикат *has_access*(x, y, z, v, G) является истинным тогда и только тогда, когда в состоянии G истинен предикат *can_connect*(x, y, z, v, G) и (y, v, own_r)$\in R$, где $y < x$, $v < z$, y,$v \in S$, x, $z \in EC$.

Предикат *trust_relationship*(x, y, G) является истинным тогда и только тогда, когда для всех $c \in CC$, что (v, c, α_r)$\in R$, где $\alpha_r \in \{read_r, write_r\}$, $v < x$, истинно (y, c, $read_r$)$\in R$, где $y < x$, v, $y \in S$.

Изменение состояния нарушителя отображает последовательное получение доступа нарушителя к различным узлам сети с правами доступа различных пользователей. Если в состоянии G системы нарушитель, имея права доступа пользователя $a \in U$ на узле $b \in EC$, в результате выполнения последовательности преобразований система перейдет в состояние G', а нарушитель может получить права доступа пользователя $c \in U$ на узле $d \in EC$, то <*current_host*> = b, <*current_user*> = a, <*target_host*> = d, <*target_user*> = c. *Координатами нарушителя* в состоянии G' будем называть пару (<*target_host*>, <*target_user*>).

Определенные выше предикаты используются для проверки исходного состояния модели в соответствии с базой описания атак или становятся истинными в результате перехода из одного состояния в другое, что соответствует выполнению последовательности преобразований базовой или ФАС ДП-моделей.

Графом атак модели REM будем называть конечный помеченный ориентированный граф без петель $G_{\text{REM}} = (G^* \times EC \times U, E_{\text{REM}})$, в котором элементы множества $G^* \times EC \times U$ являются вершинами графа атак, а элемент $((G, x \times y), (G', z \times v)) \in E_{\text{REM}}$ тогда и только тогда, когда:

- в состоянии G истинны предикаты, соответствующие предусловиям атаки, в состоянии G' истинны предикаты, соответствующие постусловиям атаки;
- (z, v) – координаты нарушителя в соответствии с описанием атаки;
- существуют состояния $G_1, \ldots, G_N = (S_N, E_N, R_N \cup A_N \cup F_N, H_N)$ и правила преобразования состояний $op_1, \ldots, op_N$, такие, что $G \vdash_{op1} G_1 \vdash_{op2} \ldots {}_{opN} G_N = G'$, где $N \geq 0$, и если правило op зависит от сущности $e \in E \backslash CC$, то $e < x$ или $e < z$.

Ниже показан граф атак модели REM сети, изображенной на рис. 2, где состояниям G, G_1, G' соответствуют графы доступов, изображенные на рис. 3, *а*, *в*, *г* соответственно.

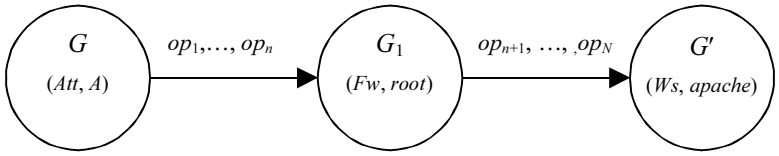


Рис. 5

7. Формальная модель анализа защищенности сетей VTG

Shahriary, Sadoddin, Jalili и Zakeri [2] на основе подходов, используемых в классической модели Take-Grant, предложили модель VTG (vulnerability Take-Grant) для анализа защищенности сетей. Модель VTG является расширением модели Take-Grant, в котором вводятся дополнительные права доступа, понятие уязвимости и новые правила преобразования – vulnerability rewriting rules.

Опишем основные положения модели VTG:

- каждому узлу сети соответствует некоторое множество известных уязвимостей;
- возможны следующие права доступа: h – размещения, x – выполнения, o – владения, r, w, t, g – как в классической модели Take-Grant;
- уязвимость сущностей обозначается с помочью их метки;
- рассматриваются 3 класса уязвимостей – переполнение буфера, слабые пароли, отношение доверия;
- для первых двух классов вводятся новые правила преобразования состояния модели: *buffer overflow* (*bof_rule*) и *password cracking* (*pc_rule*) rewriting rule;
- для проверки истинности предиката *can_access*(α, x, y, VTG_0) строится замыкание графа доступов;
- предикат *can_access*(α, x, y, VTG_0) является истинным тогда и только тогда, когда существуют состояния $\text{VTG}_1, \ldots, \text{VTG}_N$ и правила преобразования состояний $op_1, \ldots, op_N \in \{bof\_rule, pc\_rule, take\}$, такие, что $\text{VTG}_0 \vdash_{op1} \text{VTG}_1 \vdash_{op2} \ldots \vdash_{opN} \text{VTG}_N$ и ребро (x, y) помечено α в VTG_N, где $N \geq 0$.

Опишем формальную VTG ДП-модель анализа защищенности сетей на основе базовой и ФАС ДП-моделей. Неформально, под уязвимостью принято понимать некоторое нежелательное свойство КС, которое может быть использовано нарушителем при реализации атаки и может привести к осуществлению угрозы. *Уязвимостями* будем называть сущности, функционально ассоциированные с другими субъектами КС, если при реализации информационных потоков к ним происходит нарушение безопасности. Субъекты, имеющие такие сущности, назовем *уязвимыми*. По определению уязвимость является сущностью-объектом КС.

Будем использовать введенные ранее обозначения. Через V обозначим множество уязвимостей системы.

Правам r, w, x модели VTG соответствуют права доступа $read_r$, $write_r$, $execute_r$ ДП-моделей; вместо прав t, g, o используется право own_r, дуге в VTG-графе из вершины x в y, помеченной правом h, соответствует отношение $y<x$ в ДП-модели.

В соответствии с положениями модели VTG:

- при активации пользователем *u* некоторого процесса *p* последний наследует все права пользователя *u*;

- уязвимости процесса, связанные с переполнением буфера, функционально ассоциированы с пользователем, от имени которого данный процесс запущен;

- уязвимость вида «слабый пароль» функционально ассоциирована с пользователем-владельцем пароля, что позволяет проводить анализ условий передачи прав доступа и реализации информационных потоков по памяти, но не по времени.

Вместо правил *bof_rule* и *pc_rule* модели VTG будем использовать правило преобразования *control*() *ФАС* ДП-модели. Если пользователь *u* доверяет пользователю *v* в некотором состоянии $G = (S, E, R \cup A \cup F, H)$, то $(v, u, own_r) \in R$. Для построения замыкания VTG ДП-модели применим алгоритм 8 построения *control*-замыкания ФАС ДП-модели. *Граф атак* модели VTG – замыкание графа доступа VTG ДП-модели.

ЛИТЕРАТУРА

1. *Колегов Д.Н.* Проблемы синтеза и анализа графов атак // Вестник ТГУ. Приложение. 2007. № 23. С. 180 – 188.
2. *Shahriary H.*, *Sadoddin R.*, *Jalili R.*, *Zakeri R.* Network vulnerability analyses thorough vulnerability Take-Grant model // ce.sharif.edu/~shahriari/publications/ICICS2005.pdf.
3. *Девянин П.Н.* Анализ безопасности управления доступом и информационными потоками в компьютерных системах. М.: Радио и связь, 2006. 176 с.
4. *Bishop M.*, *Frank J.* Extending the Take-Grant protection system // http://citeseer.ist.psu.edu/frank96extending.html.
5. *Девянин П.Н.* Модели безопасности компьютерных систем: Учеб. пособие для студ. высш. учеб. заведений. М.: Издательский центр «Академия», 2005. 144 с.
6. *Jajodia S.*, *Noel S.*, *O'Berry B.* Managing Cyber Threats: Issues, Approaches and Challenges, ch. Topological Analysis of Network Attack Vulnerability. Kluwer Academic Publisher, 2003.
7. *Jajodia S.*, *Noel S.*, *et al.* Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs // Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas, NV, USA, December 2003.
8. *Artz M.* NETspa, A Network Security Planning Architecture, M.S. Thesis. Cambridge: Massachusetts Institute of Technology, May 2002.