МАТЕМАТИЧЕСКИЕ ОСНОВЫ КОМПЬЮТЕРНОЙ БЕЗОПАСНОСТИ

DOI 10.17223/20710410/9/7

УДК 004.94

АНАЛИЗ БЕЗОПАСНОСТИ ИНФОРМАЦИОННЫХ ПОТОКОВ В ОПЕРАЦИОННЫХ СИСТЕМАХ СЕМЕЙСТВА *GNU/LINUX*[1]

М. А. Качанов

Томский государственный университет, г. Томск, Россия

E-mail: m.a.kachanov@gmail.com

В данной работе анализируется безопасность информационных потоков в операционных системах семейства *GNU/Linux*. Рассматриваются информационные потоки по времени с участием доверенных субъектов, приводятся примеры. Предлагается метод проверки возможности реализации информационного потока по памяти между сущностями компьютерной системы, защищенной с помощью средства *SELinux*.

Ключевые слова: *компьютерная безопасность, информационный поток, Linux.*

Введение

В настоящее время одной из актуальных задач теории компьютерной безопасности является анализ безопасности управления доступом и информационными потоками в компьютерных системах (КС) [1]. При её решении, в частности, необходимо идентифицировать информационные потоки разных типов, которые в реальных КС могут быть реализованы многими способами.

Основные виды информационных потоков по памяти и по времени были описаны и исследованы в работе [1] в рамках семейства ДП-моделей. В этих моделях предполагается, что доверенные субъекты не участвуют в реализации информационных потоков по времени, однако в реальных КС существуют процессы, в которых данные предположения частично нарушаются, то есть доверенные субъекты могут участвовать в реализации некоторых информационных потоков по времени, но это не означает, что всякий субъект, реализовавший поток от себя к доверенному субъекту или наоборот, сможет прочитать данные из любой сущности системы, к которой доверенный субъект имеет права доступа. Под участием в данном случае понимается возможность реализации потока от доверенного субъекта к некоторым заданным сущностям компьютерной системы при выполнении другими субъектами определенных действий. Подобное участие доверенных субъектов в реализации потоков может и не привести к утечке конфиденциальных данных, но тем не менее, как будет показано ниже, позволяет реализовать передачу данных между недоверенными субъектами. Данные потоки возникают из-за особенностей реализации доверенных субъектов в реальных компьютерных системах,

[1]Работа выполнена в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009–2013 гг. (гос. контракт № П1010).

заключающихся в том, что в процессе функционирования доверенные субъекты заносят данные о действиях субъектов системы в доступные на чтение другим субъектам сущности, к которым первые могут и не получать доступ. Поэтому представляется целесообразным построение модели безопасности, адекватной некоторым новым видам информационных потоков, с последующим описанием условий нарушения безопасности последних. Для этого необходимо описать такие информационные потоки в КС и исследовать механизмы их возникновения.

Кроме того, в рамках решения задачи анализа безопасности управления доступом и информационными потоками часто возникает задача анализа конфигурации средств, реализующих данное управление. Одним из таких средств является *SELinux*, применяемое в операционных системах (ОС) семейства *GNU/Linux* и реализующее различные виды управления доступом. Это средство является сложно конфигурируемым, поэтому в процессе его эксплуатации возникает задача построения формальной модели анализа реализуемых им политик безопасности в КС.

В данной работе проводится анализ безопасности информационных потоков в ОС семейства *GNU/Linux* и решаются две задачи. Первой из них является идентификация информационных потоков по времени в ОС семейства *GNU/Linux*, в том числе и потоков нового типа, а именно информационных потоков по времени с участием доверенных субъектов. Второй задачей является анализ возможности реализации информационного потока по памяти между сущностями компьютерной системы, защищенной с помощью средства *SELinux*, которое де-факто является наиболее распространенным при реализации современных политик безопасности в ОС семейства *GNU/Linux*.

Работа состоит из двух разделов. Первый раздел посвящен информационным потокам по времени в КС. Рассматривается новый вид информационного потока по времени — с участием доверенных субъектов, на примере таких компьютерных систем, как ОС *GNU/Linux* и система управления базами данных (СУБД) *MySQL*. Приводятся конкретные примеры возможности реализации информационных потоков по времени в ОС *GNU/Linux* с использованием виртуальной файловой системы, сокетов и времени последнего доступа к файлу.

Во втором разделе проводится анализ безопасности информационных потоков по памяти в *SELinux*. Приводится краткое описание данного средства, предлагается метод проверки возможности реализации информационного потока по памяти между сущностями КС и предлагается программное средство автоматизированного анализа политики безопасности *SELinux*. В заключении подводятся итоги работы.

1. Примеры информационных потоков по времени в *GNU/Linux*

1.1. Информационные потоки по времени без участия доверенных субъектов

Приводятся примеры информационных потоков по времени в *GNU/Linux* без участия доверенных субъектов. Хотя данный вид информационных потоков и был рассмотрен в рамках семейства ДП-моделей [1], в литературе редко приводятся примеры возможности их реализации на практике в реальных компьютерных системах, поэтому представляется целесообразным подробно описать способы реализации данных информационных потоков в *GNU/Linux*.

Далее приводятся примеры в следующем виде. В *описании* кратко излагается основная идея реализации информационного потока. В *цели* указывается желаемый результат использования потока. Далее уточняются *необходимые условия* для его реали-

зации. После этого описывается *метод реализации потока* с конкретными *примерами и результатами* его использования в $GNU/Linux$. Затем следуют некоторые *наблюдения*, облегчающие реализацию потока на практике. В конце предлагаются *механизмы защиты* ОС от реализации рассматриваемого информационного потока по времени.

Пример 1. Информационный поток по времени с использованием сокетов.

Описание. Используется тот факт, что если слушающий сокет уже создан одним процессом, то другой процесс сокет на том же порту создать не сможет, и ядро вернет ему код ошибки. По факту возможности/невозможности создания сокета передаются данные.

Цель. Передача данных между процессами, запущенными от имени разных пользователей.

Необходимые условия. Два процесса, имеющие право создать слушающий сокет на некотором порту.

Метод реализации потока. Если процесс P_1 хочет передать 1 процессу P_2, то он создает слушающий сокет на оговоренном порту, иначе не создает. P_2 пытается создать сокет на том же порту. Если это удается сделать, то была передана 1, иначе 0.

Примеры/Результаты. Для достижения высокой скорости передачи данных при создании сокета следует ядру указать, чтобы порт не блокировался на некоторое время после разрыва соединения.

Заключения/Наблюдения. Для успешной реализации потока желательно, чтобы порт был из высокого диапазона (непривилегированный).

Механизмы защиты. На уровне ядра запретить процессам создавать слушающие сокеты, кроме тех, что им действительно необходимы. Тем не менее, если два процесса в соответствии с политикой безопасности могут создать сокет на одном и том же порту, то такие меры не защитят от потока по времени. Можно также контролировать частоту создания сокета, но данная защита не является надежной. Другой вариант — изменить ядро так, чтобы по коду возврата системного вызова нельзя было определить, успешно ли завершилась операция, но на практике это может означать потерю работоспособности системы.

Пример 2. Информационный поток по времени с использованием времени последнего доступа к файлу.

Описание. Используется особенность файловой системы, заключающаяся в том, что в метаданных, относящихся к файлу, сохраняется время последних доступа и модификации.

Цель. Передача данных между процессами, запущенными от имени разных пользователей.

Необходимые условия. Файл $file$, доступный на открытие процессу P_1 и «видимый» процессом P_2. Под видимостью понимается возможность выполнить системный вызов `stat` с файлом $file$. Фактически P_2 в списке файлов директории видит файл $file$.

Метод реализации потока. Если процесс P_1 хочет передать 1 процессу P_2, то он осуществляет доступ к файлу $file$, в противном случае — не осуществляет. Процесс P_2 выполняет системный вызов `stat` с файлом $file$ и узнает время последнего доступа. Если оно изменилось с момента предыдущего выполнения вызова `stat`, то считаем, что P_1 передал 1, иначе — 0.

Примеры/Результаты. Пусть пользователь U_1 имеет доступ только к своей домашней директории, а пользователь U_2 «видит» (в оговоренном выше смысле) корневой каталог домашней директории. Тогда пользователь U_1 может передать данные

пользователю U_2, открывая/не открывая свой домашний каталог, а U_2 будет делать `stat` на домашнем каталоге U_1.

Заключения/Наблюдения. Поток имеет место, если к файлу не происходит обращений со стороны других процессов. В *GNU/Linux* отсутствует иерархичность во времени доступа к файлу. То есть, если получим доступ к корневой директории, то время доступа к файлу в ней не изменится.

Механизмы защиты. Можно монтировать файловую систему без учета времени доступа:

```
mount -noatime -nodiratime
```

1.2. Информационные потоки по времени с участием доверенных субъектов

Случаи возникновения информационных потоков по времени, описанные в работе [1], охватывают множество возможностей их реализации в реальных КС, но не всегда полно отражают действительность. Так, новый вид информационных потоков по времени, а именно с участием доверенных субъектов, был обнаружен в таких КС, как ОС *GNU/Linux*, а также СУБД *MySQL*.

Для реализации информационного потока по времени в ОС *GNU/Linux* используется виртуальная файловая система *proc*. Особенностью *proc* является то, что информация о действиях одного процесса может отображаться в файлах, доступных для чтения процессам, запущенным от имени других пользователей. Например, пусть имеются два процесса (P_1 и P_2) с идентификаторами $pid1$ и $pid2$ соответственно и пусть процесс P_2 имеет право чтения файла $/proc/pid1/status$. В этом файле, в частности, отображается количество нитей (*threads*), которыми оперирует процесс P_1. При создании либо удалении процессом P_1 нити информация об этом будет заноситься ядром ОС *GNU/Linux* в файл $/proc/pid1/status$. Читая данный файл, процесс P_2 может получить данные от процесса P_1. В данном файле, в частности, отображается количество нитей, которыми оперирует процесс P_1. Между процессами существует договоренность, что если P_1 хочет передать 0, то он создает одну нить, а если хочет передать 1, то создает две нити. Создание нити выполняется с помощью вызова функции `pthread_create` библиотеки `pthread`. P_2 читает информацию о количестве нитей из файла $/proc/pid1/status$ и, таким образом, получает данные от P_1. На первый взгляд может показаться, что данный способ реализации информационного потока по времени подпадает под уже описанные в рамках ДП-моделей случаи, но это не так. Существенной особенностью приведенного выше примера является то, что при создании нити P_1 не осуществляет никаких обращений к файловой системе, а данные в файл $/proc/pid1/status$ записывает ядро ОС, которое является доверенным субъектом. Таким образом, P_1 может вообще не иметь никаких прав доступа в файловой системе, но тем не менее информационный поток по времени может быть реализован. Стоит уточнить, что реализация *proc* в ОС *GNU/Linux* такова, что пользователь, от имени которого запущен P_1, хоть и является владельцем файла $/proc/pid1/status$, но тем не менее не может менять права доступа к нему и не может открыть этот файл на запись. Для предотвращения возможности реализации подобного информационного потока по времени может быть использовано средство *SELinux*, позволяющее наложить дополнительные ограничения на стандартную политику безопасности *GNU/Linux* и запретить чтение файла $/proc/pid1/status$ всем процессам, кроме P_1.

В случае СУБД *MySQL* аналогичная ситуация возникает, когда некоторый пользователь осуществляет запросы к базе данных (БД). Ядро СУБД, являясь доверенным

субъектом, ведет статистику о количестве и типах запросов, об объеме принятых и переданных данных и некоторых других параметрах. Например, при всяком запросе пользователя `show databases` ядро СУБД будет увеличивать текущее значение счетчика подобных запросов на единицу. Стоит отметить, что даже пользователь с минимальными правами к БД может тем или иным образом влиять на параметры, статистику о которых ведет ядро СУБД. С помощью запроса `show status` пользователи системы могут получить полный отчет о накопленной статистике и увидеть текущие значения параметров, в том числе количество определенных запросов всех пользователей системы. Таким образом, один пользователь БД может передать данные другому пользователю, лишь совершая запросы к БД, разрешенные ему политикой безопасности, причем второй пользователь может не иметь никаких прав доступа к таблицам БД, с которыми работает первый пользователь.

Оба приведенных примера объединяет то, что информационные потоки по времени, возникающие в результате осуществления описанных действий, реализуются за счет отображения ядром системы, которое является доверенным субъектом, информации о её функционировании в сущностях, к которым субъекты системы непосредственно не получали доступа. Ядро системы само заносит данные о действиях субъектов системы в доступные на чтение другим субъектам сущности, причем первые могут и не иметь никаких прав доступа к данным сущностям.

В связи с обнаружением информационных потоков по времени с участием доверенных субъектов возникает необходимость учета данных потоков при анализе защищенности КС. В рамках семейства ДП-моделей возможно введение нового вида ассоциированных сущностей, указывающих на возможность реализации к ним информационных потоков по времени в зависимости от выполняемых субъектом действий. Кроме того, возможно введение новых правил преобразований, а также формулирование и обоснование необходимых и достаточных условий возможности реализации информационных потоков по времени между сущностями КС.

2. Анализ безопасности информационных потоков по памяти в *SELinux*

2.1. Описание средства *SELinux*

В настоящее время распространенным механизмом управления доступом и информационными потоками в ОС семейства *GNU/Linux* является средство *SELinux*. Данное средство представляет собой набор патчей для ядра *Linux* и входит в его стандартную поставку. Стоит отметить, что *SELinux*, а также утилиты его администрирования включены в дистрибутив ОС *Red Hat Enterprise Linux*, сертифицированный ФСТЭК. Средство *SELinux* изначально разрабатывалось Агентством национальной безопасности США, но в конце 2000 года его исходный текст был открыт под лицензией *GPL*, и проект был передан в разработку мировому сообществу.

SELinux позволяет реализовать принудительный контроль доступа в ОС класса *Unix* поверх стандартной дискреционной политики безопасности. С помощью данного средства возможна реализация дискреционного, ролевого, а также мандатного управления доступом. Работа средства *SELinux* основана на сопоставлении каждой программе или процессу, ресурсам (файлу, директории, сокету и т. д.) определенного типа. Тип, сопоставленный процессу, принято называть доменом. Каждый домен представляет собой множество прав доступа, достаточных для нормального функционирования процесса, но не более того. Например, домен может быть ограничен в определенных действиях с заданными файлами. Для того чтобы иметь возможность устанавливать подобные ограничения для конкретных ресурсов, каждый файл помечен определен-

ным контекстом безопасности. Домен не может получить доступ к файлам, имеющим контекст безопасности, отличный от тех, к которым ему непосредственно разрешено получать доступ. При определенных условиях процесс, порождающий новый процесс с помощью запуска исполняемого файла, может покинуть свой домен и перейти в новый. Новый домен может иметь другие привилегии в системе, нежели исходный. Механизм $SELinux$, гарантирующий строгое следование предписанным правилам взаимодействия доменов и типов, получил название $type\ enforcment$. Также существуют и другие механизмы безопасности, в том числе и ролевое управление доступом ($RBAC$). Определения типов, контекстов безопасности, а также возможных переходов между доменами описываются в политике безопасности на собственном гибком языке. К сожалению, политики зачастую довольно объемны и сложны, что затрудняет их комплексное исследование. В связи с этим возникает задача верификации политик безопасности $SELinux$.

Известно несколько подходов к верификации политик безопасности $SELinux$. В [2] вводится формальная модель описания правил политик безопасности, а также предлагается рекурсивный алгоритм проверки возможности получения субъектом определенного права доступа к объекту по начальному состоянию компьютерной системы. При этом данная модель не учитывает возможность реализации информационных потоков по памяти между сущностями КС и позволяет получить лишь примитивную информацию о возможных правах доступа субъекта. В работе [3] рассматривается программное средство *Apol*, включенное в набор утилит $SETools$ для администрирования $SELinux$. Данное средство способно отслеживать информационные потоки по памяти, находить все возможные пути реализации информационного потока между двумя сущностями КС, а также обнаруживать некоторые информационные потоки по времени. Однако для данного средства отсутствует формальная модель, а также нет документации, описывающей алгоритм проверки возможности реализации информационного потока по памяти. Кроме того, *Apol* не учитывает функционально и параметрически ассоциированные с субъектами сущности.

Таким образом, представляется целесообразным разработать средство для анализа политик безопасности $SELinux$, позволяющее анализировать возможность реализации информационного потока по памяти между сущностями КС, которое основано на формальной модели, учитывающей функционально ассоциированные с субъектами сущности. В дальнейшем подобный метод может быть предложен и для анализа возможности реализации информационных потоков по времени.

2.2. Анализ возможности реализации информационного потока по памяти в $SELinux$

В рамках семейства ДП-моделей [1] проводится анализ безопасности компьютерных систем с дискреционным, мандатным и ролевым управлением доступом, формулируются и обосновываются необходимые и достаточные условия получения недоверенным субъектом права доступа владения к доверенному субъекту, а также предлагаются алгоритмы построения замыканий, позволяющих определить истинность предиката can_share для всех вершин и прав доступа одновременно. К сожалению, на практике при автоматизированном анализе защищенности реальных компьютерных систем данные алгоритмы малоприменимы из-за их вычислительной сложности. Так, в работе [4] был предложен алгоритм построения замыкания базовой ролевой ДП-модели, а также было показано, что он имеет полиномиальную сложность. Известно, что для построения замыкания с помощью ЭВМ для состояния КС с 60 сущностями, 60 ролями и

30 недоверенными пользователями потребовалось 7 мин. В реальных КС количество сущностей несоизмеримо больше, что затрудняет применение вышеуказанного алгоритма для анализа их защищенности. В связи с этим возникает задача построения алгоритма, пригодного для анализа безопасности компьютерных систем на практике.

Рассмотрим средство управления доступом и информационными потоками в ОС класса $Unix-SELinux$ и предложим метод проверки возможности реализации информационного потока по памяти между сущностями КС, защищенной с помощью данного средства, пригодный для практического применения.

Как было отмечено выше, для описания политик безопасности в $SELinux$ используется собственный язык. Данный язык имеет множество синтаксических конструкций. Ввиду того, что разбор политики безопасности не является основной задачей данной работы, будем рассматривать лишь некоторые конструкции языка, субъективно наиболее важные для проверки возможности реализации информационного потока по памяти между сущностями КС. Такими конструкциями были выбраны определения типов и доменов, а также векторов доступа. Конструкции, отвечающие за определение атрибутов, ограничений, протоколирование, были исключены из рассмотрения. Кроме того, язык политик безопасности $SELinux$ позволяет задавать принудительные переходы субъектов КС между доменами. Данные конструкции языка далее также рассматривать не будем. Такие довольно сильные ограничения на исходный текст политики безопасности возможно наложить ввиду того, что существуют работы (например, [2]), в которых предлагаются формальные модели, основанные именно на языке политик безопасности и учитывающие его значимые синтаксические конструкции. При расширении предлагаемого далее метода на весь язык политик можно воспользоваться результатами, изложенными в подобных работах. В связи с вышесказанным дальнейшие рассуждения будем вести в следующих предположениях. Будем рассматривать лишь конструкции языка описания политик безопасности $SELinux$, отвечающие за определение типов, а также векторов доступа и не будем рассматривать информационные потоки по времени между сущностями КС. В связи с тем, что информационные потоки по времени в КС не участвуют в порождении информационных потоков по памяти, а данный раздел посвящен последним, это предположение не ограничивает общности рассуждений.

Рассмотрим подробнее язык описания политик безопасности $SELinux$, а именно те его конструкции, которые были выбраны для анализа возможности реализации информационного потока по памяти.

Определение типа имеет одну из следующих форм (листинг 1):

```
1 type type_id;
2 type type_id, attribute_id;
3 type type_id alias alias_id;
4 type type_id alias alias_id, attribute_id;
```

Листинг 1. Определение типа в $SELinux$

В них
`type` — ключевое слово,
`type_id` — идентификатор типа,
`alias` — ключевое слово,
`alias_id` — необязательный псевдоним (один или несколько) типа `type_id`,
`attribute_id` — один или несколько идентификаторов атрибутов.

Как говорилось выше, конструкции языка, описывающие атрибуты, а также псевдонимы типов в данной работе не рассматриваются.

Определение вектора доступа имеет вид

```
rule_name source_type target_type : class perm_set;
```
(1)

В нём

`rule_name` — одно из ключевых слов `allow, dontaudit, auditallow, neverallow`;

`source_type, target_type` — один или несколько типов источника/назначения, либо идентификаторов атрибутов;

`class` — один или несколько классов объектов;

`perm_set` — права доступа типа источника к типу назначения.

В соответствии с данным вектором доступа типам из `source_type` будет разрешено обращаться к типам `target_type` как к объектам класса `class` с правами `perm_set`.

Также данная синтаксическая конструкция может включать метки, похожие на регулярные выражения, позволяющие кратко записать, например, все множество прав доступа либо множество без одного элемента.

Далее будем рассматривать лишь те определения векторов доступа, в которых используется ключевое слово `allow`, а права доступа записаны явно, без использования специальных меток.

Для того чтобы учесть функционально ассоциированные сущности при анализе возможности реализации информационного потока по памяти, в язык описания политик безопасности вводится новая синтаксическая конструкция, которая имеет вид

```
fas subj_types : assoc_types;
```
(2)

В ней

`subj_types` — один или несколько идентификаторов типов,

`assoc_types` — один или несколько идентификаторов типов.

Данная конструкция указывает на то, что типы, перечисленные в `assoc_types`, являются функционально ассоциированными с типами из `subj_types`.

Сам язык описания политик безопасности *SELinux* не включает в себя подобной синтаксической конструкции. Она служит лишь для анализа возможности реализации информационного потока по памяти и не является правилом, разграничивающим доступ между субъектами и объектами. Определения функционально ассоциированных сущностей с помощью вышеуказанной конструкции могут быть вынесены в отдельный файл таким образом, что оригинальная политика безопасности останется неизменной.

Опишем метод проверки возможности реализации информационного потока по памяти между сущностями КС, защищенной с помощью средства *SELinux*.

Метод состоит в построении по тексту политики безопасности графа информационных потоков по памяти между сущностями КС.

Поскольку сама политика безопасности *SELinux* (в оговоренных выше ограничениях) для разграничения доступа между типами включает лишь правила вида (1), то для анализа возможности реализации информационного потока по памяти будем отталкиваться именно от этих правил. Особенностью данных правил является то, что множество прав доступа не ограничивается традиционными правами на чтение, запись и исполнение. Права доступа основываются на системных вызовах ядра *Linux*, то есть, грубо говоря, определение вектора доступа указывает на то, какие системные вызовы тип источника может применять к типу назначения. Например, возможны следующие определения векторов доступа:

```
allow initrc_t acct_exec_t : file { getattr read execute };
allow ftpd_t initrc_t : fifo_file { getattr read write append ioctl lock };
```

Более подробную информацию по этому вопросу можно найти в [5].

Идея метода состоит в том, чтобы абстрагироваться от прав доступа субъектов к объектам и построить ориентированный граф, в котором вершины будут сопоставлены сущностям КС, а дуги — информационным потокам по памяти между сущностями, сопоставленными вершинам.

Применительно к политике *SELinux* это означает, что по векторам доступа строится ориентированный граф, в котором вершины будут сопоставлены типам, указанным в векоторах доступа, а дуги — возможным информационным потокам по памяти между типами, сопоставленными вершинам. Здесь нужно уточнить, как по политике *SELinux*, а именно по векторам доступа, определить возможные информационные потоки по памяти между типами, указанными в векторах доступа. Поскольку в векторах доступа указывается множество прав доступа между типами, то предлагается определить множество прав доступа, при обладании которыми возможна реализация информационного потока по памяти от домена либо к нему. Для этого предлагается ввести конструкцию вида

```
write_m direction : class perm_set;                    (3)
```

В ней

`write_m` — ключевое слово;

`direction` — одно из ключевых слов `to`, `from`;

`class` — идентификатор класса;

`perm_set` — множество прав доступа.

Данная конструкция указывает на то, что при обладании доменом любым из прав в `perm_set` возможна реализация информационного потока по памяти между доменом и типом в направлении `direction` (`to` соответствует направлению от домена к типу, `from` — наоборот).

Например, возможны следующие определения информационных потоков по памяти:

```
write_m to : file { write append };
write_m to : fifo_file { write append };
write_m from : chr_file { read };
```

Язык описания политик безопасности *SELinux* не включает в себя подобной конструкции, и она вводится лишь для анализа возможности реализации информационного потока по памяти между типами. Данные конструкции могут быть вынесены в отдельный файл так, что оригинальная политика безопасности останется неизменной.

Далее будем использовать термины и обозначения ФАС ДП-модели:

E — множество сущностей;

$S \subset E$ — множество субъектов;

$[s] \subset E$ — множество всех сущностей, функционально ассоциированных с субъектом s (при этом по определению выполняется условие $s \in [s]$, и для каждого субъекта множество сущностей, функционально с ним ассоциированных, не изменяется в процессе функционирования системы);

$R_f = \{write_m\}$ — множество видов информационных потоков, где $write_m$ — информационный поток по памяти на запись в сущность;

$F_a \subset E \times E \times R_f$ — множество информационных потоков между сущностями;

$F \subset E \times E$ — множество информационных потоков по памяти.

Пусть определены множества S, E, F, и пусть для всех $s \in S$ определено множество $[s]$. Определим $G = (E, F)$ — конечный ориентированный граф, в котором элементы множества E являются вершинами, а элементы множества F — дугами.

Метод 1 проверки возможности реализации информационного потока по памяти между сущностями КС.

Пусть определен граф $G = (E, F)$.

1. Для всех $s \in S$, для всех $e \in [s] \setminus \{s\}$ положить $F = F \cup \{(e, s)\}$.

2. Для всех $s \in S$, для всех $f \in [s]$, для всех $e \in E$, если существует путь из e в f, положить $F = F \cup \{(s, e)\}$.

3. Реализация информационного потока по памяти от сущности $e_1 \in E$ к сущности $e_2 \in E, e_2 \neq e_1$, возможна тогда и только тогда, когда в графе G существует путь из e_1 в e_2 (если путь есть — ответ «да», иначе — «нет»).

Покажем теперь, как данный метод может быть применен для анализа возможности реализации информационного потока по памяти между типами, указанными в политике безопасности $SELinux$.

Пусть мы имеем текст политики безопасности $SELinux$ и определения функционально ассоциированных сущностей с помощью конструкции (2) и информационных потоков по памяти с помощью конструкции (3).

Задача ставится следующим образом: для данных двух типов, указанных в политике безопасности, определить возможность реализации информационного потока по памяти от первого типа ко второму.

Предложим метод решения данной задачи.

Метод 2 проверки возможности реализации информационного потока по памяти в $SELinux$.

1. Положить $S = E = F = \varnothing$ и $G = (E, F)$.

2. Для каждого определения типа вида листинг 1 (строка 1): если тип является типом субъекта (доменом), то добавить соответствующий ему элемент в множество S, иначе — в множество E. Для каждого определения функционально ассоциированных сущностей вида (2) добавить элементы, соответствующие ассоциированным типам, в множество $[s]$, где s — элемент, соответствующий типу субъекта.

3. Для каждого вектора доступа вида (1): если в E отсутствуют элементы, соответствующие типам, указанным в векторе доступа, то аналогично п. 2 добавить эти типы в множества S или E. Для каждого определения информационного потока вида (3): если вектор доступа содержит хотя бы одно из прав, указанных в (3), то для каждого типа источника и каждого типа назначения в векторе добавить дуги (т. е. элементы в F) по направлению, указанному в определении информационного потока.

4. Для графа $G = (E, F)$ применить метод 1.

5. Реализация информационного потока по памяти от типа — источника к типу — приемнику возможна тогда и только тогда, когда в п. 4 метод 1 выдаёт ответ «да» для пары вершин, соответствующих этим типам.

Пример.

Пусть задана политика безопасности $SELinux$:

```
allow user_t tmp_t : file {read write append};
allow ftpd_t tmp_t : file {write append};
allow ftpd_t ftpd_tmpfs_t:file { create open getattr setattr read write };
allow user_t etc_t : file {getattr};
allow eva_t etc_t : file {write};
```

И пусть заданы определения информационных потоков по памяти и функционально ассоциированных сущностей:

```
write_m to : file {write append};
write_m from : file {read};
fas user_t : {etc_t};
```

Построенный в п. 3 метода 2 граф будет выглядеть, как показано на рис. 1.

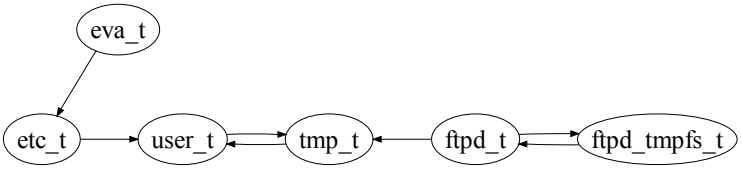

Рис. 1. Граф, полученный в п. 3 метода 2

Если в п. 5 метода 2 рассмотреть не одна пару вершин, а все возможные упорядоченные пары и всякий раз добавлять в граф дугу тогда и только тогда, когда метод 1 выдаёт ответ «да» для данной пары вершин, то получим граф, изображенный на рис. 2.

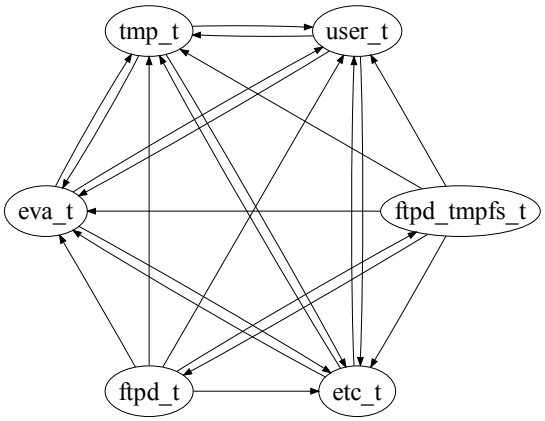

Рис. 2. Граф всех возможных информационных потоков по памяти между типами

2.3. Практическая реализация

Для анализа политик безопасности *SELinux* было разработано программное средство, основанное на методе 2 проверки возможности реализации информационного потока по памяти в *SELinux*. Данное средство позволяет по политике безопасности, а также по определениям информационных потоков по памяти вида (3) и функционально ассоциированных с субъектами сущностей вида (2) ответить на вопрос, возможна ли реализация информационного потока по памяти между типами, указанными в политике безопасности *SELinux*. Входными данными для данного средства являются текст политики безопасности *SELinux*, а также определения собственных конструкций. Кроме того, на вход могут быть поданы имена типов, для которых необходимо проверить возможность реализации информационного потока по памяти. На выходе средство дает информацию о возможности реализации потока либо между двумя указанными типами, либо между всеми типами, указанными в политике безопасности *SELinux*. Также средство способно вывести граф информационных потоков в виде

изображения. Данное средство написано на языке программирования *Python* с использованием библиотек *sepolgen* и *pygraph*. В библиотеке *sepolgen* исправлены некоторые ошибки, а также добавлена новая функциональность.

С помощью данного средства проанализирована часть реальной политики безопасности *SELinux* из стандартного пакета для дистрибутива *Ubuntu* 9.04. Для анализа была выбрана модульная политика для *ftp*-сервиса *ftpd*. По тексту данной политики, а также по определениям информационных потоков по памяти и функционально ассоциированных сущностей построен граф информационных потоков (рис. 3).

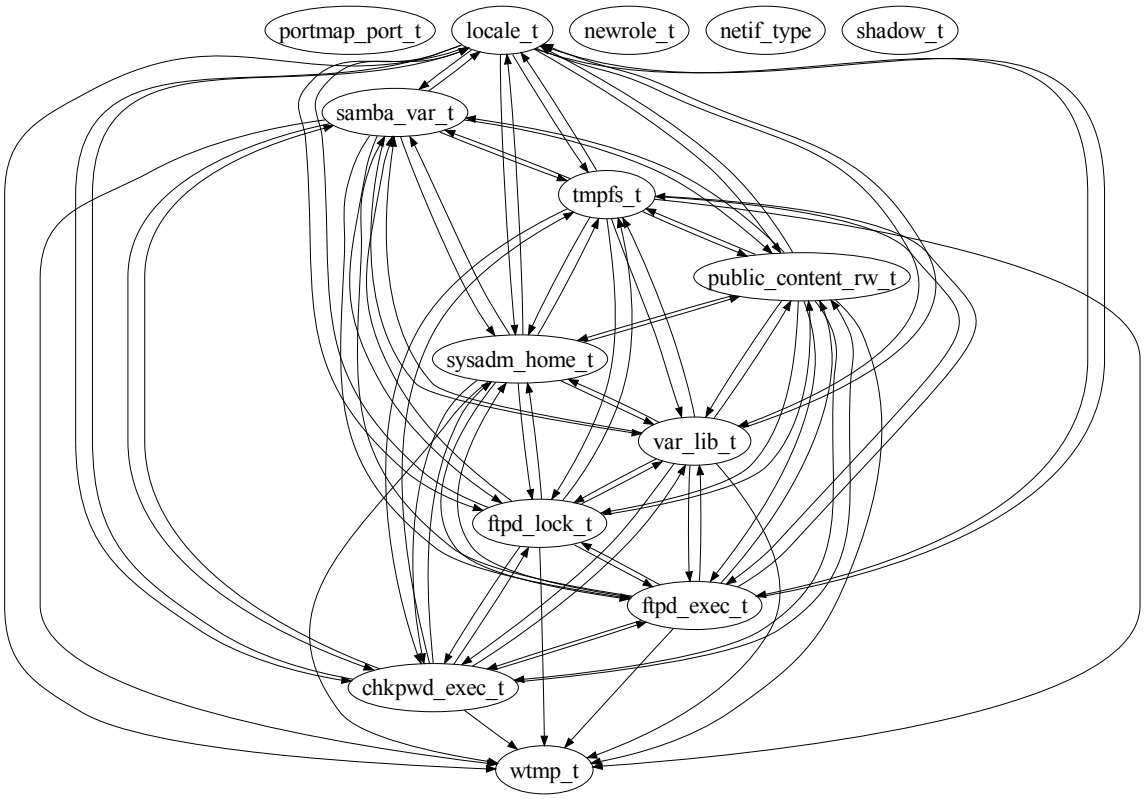

Рис. 3. Часть графа информационных потоков по памяти для политики *ftpd*

Таким образом, для всех типов, указанных в политике безопасности для *ftpd*, удалось определить возможность реализации информационных потоков по памяти между ними.

Заключение

В данной работе рассмотрены вопросы анализа безопасности информационных потоков в ОС семейства *GNU/Linux*. Исследованы информационные потоки по времени в ОС *GNU/Linux*, в том числе и новые информационные потоки по времени с участием доверенных субъектов. Приведены конкретные примеры возможности реализации потоков в ОС *GNU/Linux* с использованием виртуальной файловой системы, сокетов и времени последнего доступа к файлу с подробным описанием и рекомендациями по защите. Рассмотрено распространенное средство реализации политик безопасности в ОС *GNU/Linux – SELinux*, предложен метод проверки возможности реализации инфор-

мационного потока по памяти между сущностями КС, а также описано программное средство автоматизированного анализа политики безопасности *SELinux* с примерами его практического применения.

ЛИТЕРАТУРА

1. *Девянин П. Н.* Анализ безопасности управления доступом и информационными потоками в компьютерных системах. М.: Радио и связь, 2006. 176 с.
2. *Zanin G., Mancini L.* Towards a formal model for security policies specification and validation in the selinux system // Proc. of the ninth ACM symposium on Access control models and technologies. NY, USA: ACM, 2004. P. 136–145.
3. `http://selinux-symposium.org/2005/presentations/session5/5-3-macmillan.pdf` — SELinux Symposium. 2005.
4. *Качанов М. А.* Замыкание базовой ролевой ДП-модели // Прикладная дискретная математика. 2009. Приложение № 1. С. 41–44.
5. `http://selinuxproject.org/page/Main_Page` — SELinux Project Wiki. 2010.