УДК 681.3

Улучшенная ролевая модель управления доступом к объектам

Майоров А.В.

Аннотация

В статье предлагается новая модель управления доступом к объектам приложения. Говорится о том, какие преимущества она имеет по сравнению с классическими моделями, и показывается, как эти модели можно реализовать в ее рамках.

1. Введение

В статье освещаются общие принципы построения систем безопасности и дается описание базовых моделей, обычно использующихся для этой цели (раздел 3).

Детально рассматриваются ограничения ролевой модели, которая, из-за сравнительной простоты администрирования, является предпочтительной к использованию в прикладных программах (раздел 4).

Подробно описываются возможности и составные части разработанной нами модели, и показывается, что каждая из них лишена вышеупомянутых ограничений (раздел 5).

Доказывается, что в рамках предлагаемой модели можно реализовать базовые ролевую и дискреционную модели, а также обсуждаются возможные пути соединения ее с мандатной моделью (раздел 6).

В заключении подчеркивается основное отличительное свойство нашей модели и кратко говорится о преимуществах, которые оно несет.

2. Постановка задачи и результат

Перед нами стояла задача разработать модель управления доступом к объектам приложения, позволяющую реализовывать любую разумную политику безопасности приложения, оставаясь при этом максимально удобной в администрировании.

В результате получена модель, которая, с одной стороны, совместима с известными базовыми моделями, а с другой стороны, вводит дополнительную степень свободы – привязку пользовательских полномочий к иерархии объектов системы. Подобная привязка позволяет ограничить область действия выданных пользователю полномочий и, как следствие, упростить схемы доступа к объектам.

3. Базовые модели

Общим подходом для всех моделей управления доступом является разделение множества сущностей, составляющих систему, на множества объектов и субъектов. При этом определения понятий «объект» и «субъект» могут существенно различаться. Мы будем подразумевать, что объекты являются некоторыми контейнерами с информацией, а субъекты – пользователи, которые выполняют различные операции над этими объектами.

Безопасность обработки информации обеспечивается путем решения задачи управления доступом субъектов к объектам в соответствии с заданным набором правил и ограничений, которые образуют политику безопасности.

Можно выделить три основные модели управления доступом к объектам: мандатную, дискреционную и ролевую.

3.1. Мандатная модель

Классической мандатной моделью считается модель Белла-ЛаПадулы [1]. Она базируется на правилах секретного документооборота, использующегося в правительственных учреждениях. В этой модели каждому объекту и субъекту (пользователю) системы назначается свой уровень допуска. Все возможные уровни допуска системы четко определены и упорядочены по возрастанию секретности. Действуют два основных правила:

1. Пользователь может читать только объекты с уровнем допуска не выше его собственного.
2. Пользователь может изменять только те объекты, уровень допуска которых не ниже его собственного.

Цель первого правила очевидна каждому, второе может вызвать недоумение. Смысл же его в том, чтобы воспрепятствовать пользователю с высоким уровнем доступа, даже случайно, раскрыть какие-то известные ему тайны.

Одной из проблем этой модели считается беспрепятственность обмена информацией между пользователями одного уровня, так как эти пользователи могут выполнять в организации разные функции, и то, что имеет право делать пользователь А, может быть запрещено для Б. Поэтому в практике мандатную модель обычно используют совместно с какой-нибудь другой [2] [3].

Из этих двух правил можно вынести несколько интересных наблюдений, указывающих на проблемы, которые могут проявиться в процессе адаптации модели к реальному приложению:

- Пользователи «снизу» могут попытаться передать информацию наверх, выложив ее на своем уровне. При этом они никогда не узнают, читал ли ее кто-либо «сверху» или нет, так как документ будет защищен от редактирования вышестоящими лицами.
- Еще пользователи могут попробовать «закинуть» данные на уровень выше. В этом случае верха будут иметь возможность вставить в полученный документ свои комментарии, но отправитель об этом также не узнает. Вообще, о существовании верхних уровней он может узнать только из документации к системе.
- У пользователей с высоким уровнем допуска нет никаких возможностей коммуникации с нижними уровнями. Возможно, наверху сидят очень умные люди, советы которых были бы просто бесценны, но мы об этом никогда не узнаем.

3.2. Дискреционная модель

В дискреционной модели безопасности управление доступом осуществляется путем явной выдачи полномочий на проведение действий с каждым из объектов системы. Например, в модели Харрисона-Руззо-Ульмана [4] для этого служит матрица доступа, в которой определены права доступа субъектов системы к объектам. Строки матрицы соответствуют субъектам, а столбцы – объектам. Каждая ячейка матрицы содержит набор прав, которые соответствующий субъект имеет по отношению к соответствующему объекту.

Как правило, создатель объекта обладает на него полными правами и может делегировать часть прав другим субъектам.

Дискреционный подход позволяет создать гораздо более гибкую схему безопасности, чем мандатный, но при этом он и гораздо более сложен в администрировании. С программной точки зрения его реализация очень проста, но при достаточно большом количестве объектов и субъектов система становится практически неуправляемой.

Для решения этой проблемы применяется, например, группировка пользователей. В этом случае права раздаются группам пользователей, а не каждому пользователю в отдельности. Для того чтобы пользователь получил соответствующие разрешения, нужно просто добавить его в одну или несколько групп.

Также можно использовать типизацию объектов. Каждому объекту назначается тип, а для каждого типа определяется свой набор прав (схема доступа). В этом случае столбцы матрицы доступа соответствуют не объектам, а типам объектов. Комбинирование этого подхода с группировкой пользователей позволяют существенно уменьшить матрицу доступа, а значит, и упростить ее администрирование.

В сущности, набор прав – это не что иное, как список известных системе операций, снабженных разрешением или запретом на выполнение данной операции. В крупном приложении количество известных операций может быть весьма большим. При этом большая часть операций

имеет смысл только для определенных типов объектов, а многие типовые процессы, осуществляемые пользователем в приложении, включают в себя выполнение нескольких элементарных операций над различными объектами. Поэтому, даже с уменьшенной матрицей доступа, продумать политику безопасности приложения, т.е. грамотно разделить полномочия между различными пользователями системы, достаточно сложно.

3.3. Ролевая модель

В ролевой модели [1] операции, которые необходимо выполнять в рамках какой-либо служебной обязанности пользователя системы, группируются в набор, называемый «ролью». Например, операции по регистрации документов могут быть сгруппированы в роль «регистратор».

Для того чтобы множества операций, связанных с различными ролями, не пересекались, вводится иерархическая зависимость между ролями. К примеру, роль «секретарь» может включать в себя роль «регистратор» и, плюс к тому, еще несколько дополнительных операций.

Каждый пользователь системы играет в ней одну или несколько ролей. Выполнение пользователем определенного действия разрешено, если в наборе его ролей есть нужная, и запрещено, если есть нежелательная.

В этой модели у объектов нет определенных хозяев. Вся информация расценивается как принадлежащая организации, владеющей системой. Соответственно, и роли пользователя внутри системы – это роли, которые он играет в данной организации. Как следствие, пользователю невозможно делегировать права на какой-то определенный объект. Либо у него есть доступ ко всем подобным объектам системы, либо нет.

Таким образом, преимуществом ролевой модели перед дискреционной является простота администрирования: назначение пользователей на роли и создание новых ролей не составляют никаких трудностей. В то же время она не позволяет управлять разными частями системы по отдельности, и тем более – делегировать какому-либо пользователю такие полномочия.

4. Ограничения ролевой модели

Выше мы говорили о том, что ролевая модель контроля доступа является компромиссным решением, обеспечивающим неплохие возможности в задании политики безопасности при достаточной простоте администрирования. Это позволяет рассматривать ролевую модель как наиболее подходящую для применения в прикладных программах. В то же время существующие ограничения в ряде случаев сильно затрудняют ее использование. Рассмотрим эти ограничения более подробно.

4.1. Глобальность ролей

Первое ограничение состоит в том, что пользователь принимает свои роли по отношению ко всей системе сразу. Соответственно, для системы нет разницы в правах между двумя пользователями, находящимися на одинаковой должности, даже если они занимают эти должности в разных отделах. Например, любой пользователь в роли «начальник отдела» имеет право управлять любым отделом своей организации, а это, конечно, неправильно.

Решением могло бы стать введение отдельных ролей «начальник отдела А», «начальник отдела Б» и т.п., что позволило бы нам решить проблему, не выходя за рамки ролевой модели. К сожалению, подобный вариант привносит гораздо больше проблем, чем решает.

Более правильным будет разбить все множество объектов системы на несколько подмножеств (доменов) и дать пользователям возможность играть разные роли в разных доменах системы. В нашем примере систему можно разбить на отделы, так что каждый из начальников отделов будет играть роль «начальник» только в домене, соответствующем его отделу. Такой подход применяется, например, в библиотеке Microsoft Authorization Manager [6].

4.2. Отсутствие владельца объекта

Второе препятствие перед использованием ролевой модели в ряде систем – это отсутствие в ней понятия владельца объекта. Другими словами, пользователь, создавший объект, не имеет на него никаких исключительных прав. Это вполне приемлемо для систем, поддерживающих, например, процесс купли-продажи, но перестает годиться, как только документы начинают содержать какие-то авторские материалы.

Зачастую для решения этой проблемы к объектам системы добавляют свойство «владелец», являющееся внешним по отношению к модели безопасности. В том случае, если пользователь является владельцем объекта, над которым он хочет совершить действие, проверка производится по специальным отдельным правилам, а не по тем общим, которые предусмотрены ролевой моделью.

Другой подход – ввести в ролевую модель элементы дискреционной и явным образом дать пользователю нужные права на созданный им объект.

Заметим, что оба эти варианта решают задачу, используя внешние по отношению к модели средства, и, соответственно, не снимают ограничений самой модели.

4.3. Операции принадлежат ролям

Казалось бы, группировка операций системы в роли, в рамках которых они выполняются, упрощает администрирование, но это снова верно не для всех типов систем. Предположим, что в нашей системе есть десять различных типов объектов, для каждого из которых определена операция «удалить». Тогда, если мы добавляем эту операцию в какую-либо из ролей, то любой пользователь, играющий эту роль, получает право удалять объекты любого типа. Очевидно, что это далеко не всегда является желательным поведением.

Можно попытаться решить эту проблему, введя десять различных операций, предназначенных для удаления объекта каждого из типов. Это будет выглядеть примерно так: «удалить статью», «удалить папку» и т.д. Такое решение оказывается не очень удачным, если типов не десять, а, например, сто.

Проблема еще более усугубляется, если в системе возможны различные схемы доступа к разным объектам одного типа. Например, объект типа «статья» может быть или открытым для публики, или совершенно секретным. Создавать действия «удалить публичную статью», «удалить секретную статью», «удалить публичную папку», «удалить секретную папку» и т.п. кажется совершенно неразумным.

В принципе, можно еще более увеличить количество операций, пытаясь использовать подобный подход в решении «проблемы владельца»: добавить операцию «удалить свою статью», и использовать ее для проверки прав на удаление статей, принадлежащих данному пользователю.

На практике, при использовании ролевой модели в сложных системах, разработчики обычно не пытаются декларативно задавать схему доступа к объектам системы. Вместо этого процедура проверки встраивается в нужное место программы, при этом проверяются как сведения, предоставляемые модулем ролевой безопасности (т.е. роли, в которых выступает пользователь), так и любые другие сведения об объекте (владелец объекта, уровень секретности и т.п.).

Подобный подход затрудняет изменение схемы доступа, так как для этого нужно исправлять код процедуры проверки и перекомпилировать приложение. Для упрощения этой процедуры некоторые библиотеки ролевой безопасности предлагают встроенные средства написания сценариев проверки. Эти сценарии являются фактически теми же процедурами, но написанными на другом языке программирования, и расположенными не в самой программе, а среди настроек библиотеки. В Microsoft Authorization Manager программирование сценариев ведется на VBScript и теоретически может производиться администратором системы, так как не требует перекомпилирования всей программы.

5. Модель Force-Field

Force-Field – это разработанная нами модель управления доступом, которая позволяет создавать простые в администрировании политики безопасности, является существенно более мощной, чем ролевая модель, и при этом лишена ее основных недостатков.

5.1. Дерево объектов

В разделе 4.1 обсуждалась проблема диапазона действия ролей в ролевой модели, описанной в [1], и вариант ее решения с использованием доменов. Недостаток этого решения в том, что домены нужно создавать вручную, и они явным образом не связаны ни с какими объектами системы. Фактически приложение само должно решить, к какому домену относится текущая проверка, и передать эту информацию библиотеке. К тому же отсутствует иерархия доменов, а значит, наборы ролей пользователя в разных доменах приложения совершенно независимы. Это создает трудности, если пользователь должен играть определенную роль во всей системе сразу – его придется назначить на эту роль в каждом домене в отдельности.

В модели Force-Field все объекты системы объединяются в единое дерево. У каждого объекта, кроме единственного корневого, есть один родительский объект, и любое количество дочерних. Роль может быть назначена пользователю в контексте любого объекта. При этом пользователь начинает играть назначенную роль во всей ветви дерева, которая образована этим объектом.

Отметим, что терминология, использующаяся в работе, подразумевает, что дерево объектов «растет» вниз. Самым верхним объектом является корень, а перемещение от него к ветвям – это движение вниз (или вглубь) иерархии.

Таким образом, любой объект приложения может образовать домен (см. раздел 4.1), в который будут входить он сам и все его дочерние объекты. Любой из его дочерних объектов может также образовать домен, являющийся подчиненным по отношению к домену родительского объекта. Список ролей, которые пользователь играет в определенном домене, состоит из ролей, назначенных ему в данном домене, плюс роли из домена более высокого уровня, и из домена еще более высокого уровня, и так далее, до корневого домена приложения. Роли, назначенные пользователю в корневом домене, имеют глобальный характер, т.е. действительны в контексте каждого объекта приложения.

Например, корневой объект «предприятие» имеет десять подчиненных объектов типа «отдел», под каждым из которых, в свою очередь, располагаются документы, относящиеся к данному отделу. Пользователь, которому назначена роль «начальник» в контексте конкретного отдела (или отделов), имеет полный доступ ко всем документам своего отдела, но не имеет доступа к документам других отделов, так как там он не играет соответствующей роли. Пользователь же, являющийся «начальником» корневого объекта системы, имеет полный доступ ко всем документам предприятия, что полностью соответствует соображениям элементарной логики.

5.2. Роль «владелец»

Вернемся еще раз к проблеме отсутствия владельцев у объектов в базовой ролевой модели (см. раздел 4.2). В качестве решения подойдет любой механизм, позволяющий выделить хозяина объекта и дать ему особые права на этот объект. В нашей модели для этого вводится роль «владелец», назначающаяся пользователям в контексте тех объектов, которыми они владеют.

Эта роль по своему поведению слегка отличается от обычных ролей. Во-первых, только один пользователь может играть роль «владелец» в контексте какого-то определенного объекта. Во-вторых, объект не должен наследовать роль «владелец» от родителя, если в его собственном контексте такая роль кому-либо назначена.

Первый принцип очевиден, второй поясним на примере. Пусть у нас есть иерархия объектов «дом» - «квартира». «Дом» является родительским объектом для нескольких «квартир». Если некий пользователь является владельцем «дома», то он также является и владельцем всех «квартир», у которых нет своих собственных владельцев. «Квартиры» же с явно указанным владельцем ему не принадлежат.

Развивая эту идею, заметим, что роль может быть ограничена не только одним актером, но и большим их количеством. Таким образом, если роль ограничена *n* пользователями, то в контексте

любого объекта системы не больше *n* пользователей могут играть эту роль. При этом, очевидно, что в системе в целом у этой роли может быть больше *n* назначений.

В базовой модели подобные ограничения называются кардинальностью роли и определены как максимальное количество пользователей, которые могут играть эту роль в рамках всей системы.

5.3. Класс доступа

В разделе 4.3 было показано, что, хотя распределение операций по ролям кажется логичным, оно весьма затрудняет разработку схемы безопасности. Использование же сценариев для проверки прав доступа усложняет администрирование системы.

Для создания гибкой схемы безопасности без ручного программирования сценариев проверки, в Force-Field введено понятие «класс доступа». Класс доступа содержит набор правил, задающих права выполнения определенных операций для определенных ролей. Например, правило может быть таким: «роли Администратор выполнять операцию Удаление Разрешено». Порядок следования правил важен, так как при проверке поиск делается сверху вниз и продолжается до тех пор, пока не будет найдено правило, подходящее проверяемой ситуации. Правило из нашего примера подойдет, если будет запрошено разрешение на удаление объекта, а пользователь в контексте этого объекта будет администратором.

Каждому объекту системы ставится в соответствие ровно один класс доступа, а любой класс доступа может быть назначен произвольному количеству объектов. Это позволяет иметь в системе несколько разных схем доступа к объектам, не заставляя нас связывать эти схемы с типами или какими-то другими признаками объектов. Отметим также, что назначения классов никак не связаны с иерархией объектов: дочерний объект может иметь любой класс доступа, независимо от того, какой класс назначен родительскому объекту.

Класс может базироваться на другом классе, так что, если подходящего правила в классе нет, будет просмотрен базовый класс, потом его базовый класс и так далее. Если правило так и не будет найдено, то операция считается запрещенной. Этот механизм также служит для упрощения администрирования системы.

Перечислим некоторые возможные варианты распределения классов по объектам:

1. У всех объектов одного типа один и тот же класс доступа. Следует применять в системах, где различные типы отличаются друг от друга по правилам контроля доступа, но все объекты одного типа ведут себя одинаково.
2. Есть несколько классов доступа, которые могут быть назначены любому объекту, независимо от его типа. Например, классы, определяющие уровень секретности информации (публичная, для служебного пользования, совершенно секретная).
3. Есть всего один класс доступа, который назначается всем объектам.

5.4. Иерархии

Помимо единой иерархии объектов, в нашей модели предусмотрена также иерархия ролей и иерархия операций.

Как и в базовой модели, роль может включать в себя любое количество других ролей (циклы запрещены). В этом случае, если пользователь играет в каком-то контексте некоторую роль, то он играет и все подчиненные ей роли. Корнем иерархии ролей является роль «любая роль».

Иерархия операций позволяет упростить администрирование, раздавая права не на каждую операцию в отдельности, а на целые группы операций. Например, имея операции «создать объект типа А», «… Б», «… В» и т.д., мы могли бы добавить объединяющую их операцию «создать объект». Пользователь, которому разрешена эта операция, будет иметь право создавать объекты любого типа. Как и с ролями, у этой иерархии есть корень – операция «любая операция».

5.5. Наследование

Мы уже говорили о двух механизмах наследования в нашей модели – это, во-первых, наследование списка ролей пользователя при движении вглубь по иерархии объектов, и, во-вторых, наследование правил из базовых классов. Не хватает еще одного – наследования правил

доступа от вышестоящих объектов. Подобный механизм часто встречается в файловых системах (дискреционная модель): файлы, лежащие в папке, могут не иметь своих собственных правил доступа, а наследовать эти правила у папки. При переносе этих файлов в другую папку, права пользователя на доступ к этим файлам могут поменяться.

В Force-Field это реализуется следующим образом: в любом правиле класса кроме резолюций «разрешено» и «запрещено» можно использовать вариант «как у родителя». В этом случае, для выдачи окончательного ответа будет проверено, можно ли данному пользователю выполнить запрашиваемое действие по отношению к родительскому объекту. Если можно, то и на первоначальный запрос ответ будет положительным, нет – отрицательным. Естественно, этот процесс может быть рекурсивным: если родительский объект также не имеет своего мнения, то будет проверен его родительский ответ и так далее. Если по достижении корня иерархии объектов явного разрешения или запрета так и не будет найдено, то действие считается запрещенным.

Чтобы продублировать описанное выше поведение файловой системы, достаточно создать класс, с которым объект наследует от родителя права на выполнение всех операций. В то же время наша модель позволяет и такой вариант, когда права на часть операций наследуются, а для другой части задаются явным образом.

6. Гибридность модели Force-Field

Докажем, что предлагаемая нами модель позволяет реализовать в своих рамках функциональность базовых ролевой (раздел 3.3) и дискреционной (3.2) моделей.

6.1. Реализация ролевой модели

Предположим, что в нашем приложении четыре объекта двух различных типов: A_1, A_2, B_1, B_2. Существуют две операции, которые можно выполнить над объектом типа A, и одна для объектов типа B: op_{A1}, op_{A2}, op_{B1}. Также в системе зарегистрировано два пользователя: U_1 и U_2.

Базовая ролевая модель предписывает нам ввести в систему роли и распределить операции между ролями. Введем следующие роли. Роль r_1 включает в себя операции op_{A2} и op_{B1}, а роль r_2 – операцию op_{A1}. Назначим на роли пользователей: U_1 играет в системе роль r_2, а U_2 – обе роли. Напомним, что пользователь, назначенный на определенную роль, имеет право выполнять операции, входящие в эту роль, по отношению к любому объекту системы.

Эта схема проиллюстрирована рисунком 1.

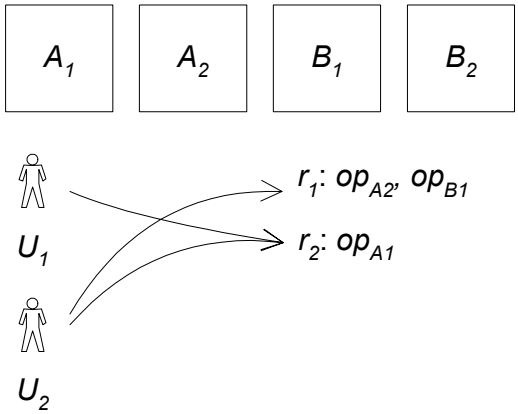


Рис. 1

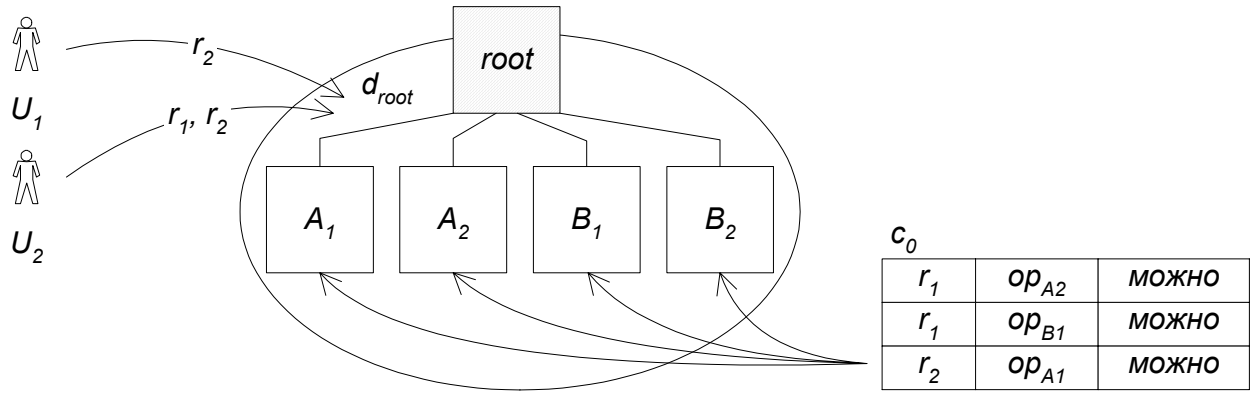
Чтобы реализовать ее в нашей модели, необходимо сначала свести все объекты приложения в единую иерархию. Для этого достаточно добавить фиктивный корневой объект – *root* – и сделать объекты его прямыми потомками. Для того чтобы все роли были глобальными (как того требует базовая модель) мы должны будем назначать пользователей на них только в контексте корневого объекта, или, другими словами, в корневом домене d_{root}. Мы вводим роли r_1 и r_2 и назначаем на них пользователей.

После введения в систему операций остается только одна нерешенная проблема: необходимо, чтобы роль определялась операциями, выполняемыми в ее рамках. Действительно, роль в

модели Force-Field, в общем случае, не соответствует этому требованию. Фактически она ничем не отличается от группы пользователей.

Решение заключается во вводе в систему единого для всех объектов класса доступа c_0, в котором операции, составляющие определенную роль, для этой роли явным образом разрешены. Очевидно, что это и есть нужный нам способ записи соответствия между ролями и операциями.

Принципиальная схема реализации приведена на рисунке 2.



c_0

r_1	op_{A2}	можно
r_1	op_{B1}	можно
r_2	op_{A1}	можно

Рис. 2

6.2. Реализация дискреционной модели

Для приложения, описанного в предыдущей задаче, система контроля доступа по модели Харрисона-Руззо-Ульмана будет подобна схеме на рисунке 3. В ней столбцы соответствуют объектам, строки – пользователям. В ячейках прописаны индивидуальные права пользователя на соответствующий объект. Отметим, что, хотя в этом и нет большой необходимости, в таблице явным образом запрещены операции, не имеющие смысла для соответствующих объектов.

	A_1	A_2	B_1	B_2
U_1	op_{A1} - да op_{A2} - нет op_{B1} - нет	op_{A1} - да op_{A2} - нет op_{B1} - нет	op_{A1} - нет op_{A2} - нет op_{B1} - нет	op_{A1} - нет op_{A2} - нет op_{B1} - нет
U_2	op_{A1} - да op_{A2} - да op_{B1} - нет	op_{A1} - да op_{A2} - да op_{B1} - нет	op_{A1} - нет op_{A2} - нет op_{B1} - да	op_{A1} - нет op_{A2} - нет op_{B1} - да

Рис. 3

Для реализации этой модели в Force-Field нужно создать четыре отдельных класса доступа (c_{A1}, c_{A2}, c_{B1} и C_{B2}) и назначить их соответствующим их объектам. В составляющих классы правилах разрешения будут даваться не ролям, а непосредственно пользователям. Необходимо подчеркнуть, что возможность указывать в правиле не роль, а пользователя, следует использовать только в крайних случаях, так как это может привести к созданию плохо управляемой политики безопасности.

В классы не включены правила, запрещающие не имеющие смысла операции, т.к. все явным образом не разрешенное автоматически считается запрещенным.

Результирующая схема приведена на рисунке 4.

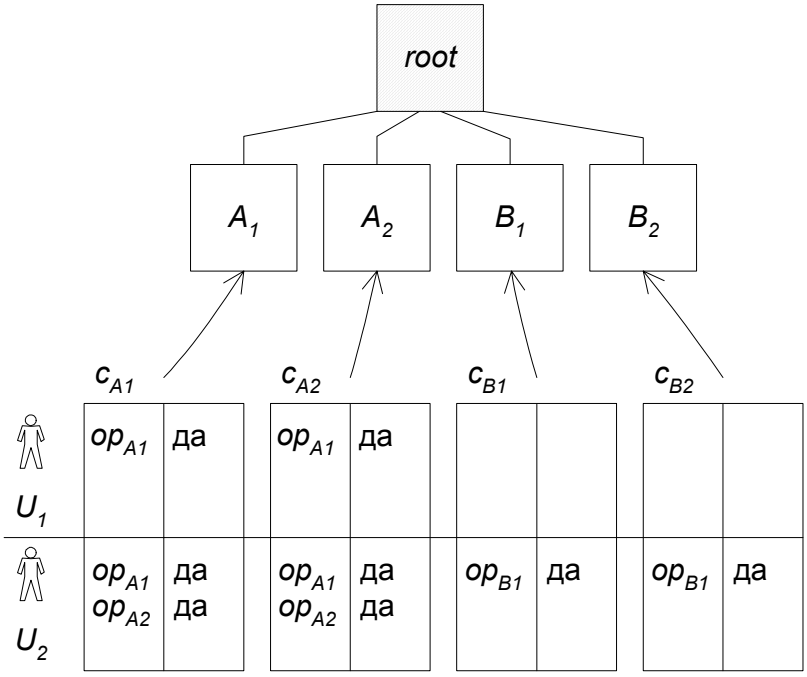


Рис. 4

Обратим внимание, что на схеме не показан корневой домен и назначения пользователей на роли. Это связано с тем, что модель, которую мы рассматривали, является слишком простой, и роли в ней не используются. Как мы уже обсуждали в разделе 3.2, в таком виде модель не годится для большинства реальных применений, поэтому расширим ее, добавив типы объектов и группы пользователей.

Хотя в нашем приложении изначально есть два типа объектов, они до сих пор находились вне разрабатываемой системы безопасности. Введем типы в систему, связав с ними классы доступа. Таким образом, количество классов безопасности уменьшается до двух: класс безопасности для объектов типа A (c_A) и класс для объектов типа B (c_B).

Для группировки пользователей мы можем использовать роли, назначаемые пользователям в корневом домене. Поэтому введем две роли: g_1 и g_2. Фактически это то же самое, что роли r_1 и r_2 из предыдущих моделей.

Изменим правила в классах доступа, с тем чтобы они выдавали разрешения группам пользователей (т.е. ролям), а не каждому пользователю в отдельности.

На рисунке 5 приведена схема реализации дискреционной модели, оптимизированной за счет типизации объектов и введения групп пользователей. В нашей модели можно реализовать и другие способы оптимизации. Например, наследование прав по аналогии с файловой системой легко реализуется за счет специального класса доступа на дочернем объекте. Он должен определять собственную схему доступа объекта и дополнять ее правилом «любая роль – любая операция – как у родителя».

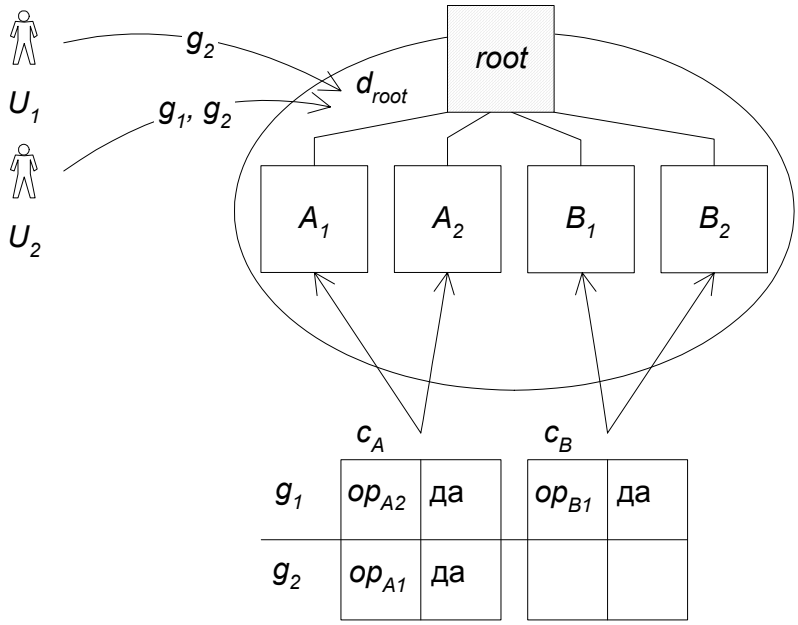


Рис. 5

6.3. Введение элементов мандатной модели

Самой первой из рассмотренных нами классических моделей была мандатная модель (см. раздел 3.1). В дальнейшем мы не уделяли ей достаточного внимания, так как она является весьма экзотичной, и крайне редко применяется в реальных приложениях. В связи с этим мы не ставили перед собой задачу охвата и этой модели, но наметили пути, по которым это может быть сделано.

Во-первых, классам доступа нужно назначить уровни секретности. Так как каждый объект приложения проассоциирован с определенным классом доступа, это автоматически назначит уровни секретности и всем объектам. При этом если уровень секретности классу все же не назначен, то можно считать, что класс и его объекты находятся на самом низком уровне.

Во-вторых, мы должны назначить уровни допуска пользователям системы. В соответствии с моделью, пользователь будет иметь право читать документы с уровнем секретности не выше его собственного, и изменять документы с уровнем не ниже.

Это подводит нас к третьему пункту. Необходимо разделить все операции системы на две группы: группу чтения и группу записи. Тогда, при проверке на допустимость операции, мы будем точно знать, какое правило применить. Дополнительно мы могли бы добавить и еще одну группу, принадлежность к которой означала бы, что операцию не нужно проверять по мандатной модели. Это позволит сделать, например, операцию уведомления о прочтении документа (ранее мы уже обсуждали, какие сложности возникают с этим в строгой мандатной модели).

Наконец, модифицируется процедура проверки прав. Если у объекта и пользователя разные уровни, то мы проводим проверки по стандартным принципам мандатной модели. Если уровень одинаковый или операция входит в «свободную» группу, то применяем обычные правила нашей модели.

Заметим, что нужно будет еще тщательно обдумать желаемое поведение системы в ситуации, когда по мандатной модели операция разрешена, а по модели Force-Field – запрещена. Ответ на этот вопрос определит направленность системы безопасности. Нужно решить, что приоритетней: полная свобода пользователям с высоким уровнем или ограничение пользователей с низким.

7. Заключение

Мы показали, что предлагаемая модель контроля доступа объединяет в себе базовые модели. Что более важно, она расширяет их возможностью назначения пользователей на роли в контексте любого объекта системы. Поэтому множество ролей, которые пользователь играет в

некий момент времени, не является одним и тем же для всех объектов приложения, а пополняется новыми ролями по мере спуска вглубь по объектной иерархии.

За счет этого появляется возможность максимально естественным образом ограничить область действия выданных пользователю полномочий определенной частью приложения. Это позволяет уменьшить необходимое количество ролей и упростить схемы доступа к объектам.

Таким образом, наша модель позволяет создавать легкие в администрировании политики безопасности, обладая, в то же время, необходимыми возможностями по ограничению несанкционированного доступа к объектам приложения.

8. Литература

1. Leonard J. LaPadula and D. Elliott Bell. Secure Computer Systems: A Mathematical Model // MITRE Corporation Technical Report 2547. Volume II. 31 May 1973.
2. Зегжда Д.П. Общая схема мандатных моделей безопасности и ее применение для доказательства безопасности систем обработки информации // Проблемы информационной безопасности. Компьютерные системы. СПбГТУ. 2000. 2.
3. Степанов П.Г. Принципы управления доступом к кесурсам в защищенной ОС «Феникс» // Проблемы информационной безопасности. Компьютерные системы. СПбГТУ. 1999. 4.
4. M. Harrison, W. Ruzzo, J. Uhlman Protection in operating systems // Communications of the ACM. 1976.
5. Баранов А.П. Зегжда Д.П., Зегжда П.Д., Ивашко А.М., Корт С.С. Теоретические основы информационной безопасности (Дополнительные главы). СПб.: СПбГТУ. 1998.
6. Mohan Rao Cavale. Role-Based Access Control Using Windows Server 2003 Authorization Manager. http://msdn.microsoft.com/library/en-us/dnnetserv/html/AzManRoles.asp