10. Лекция: Идентификация и аутентификация, управление доступом

Идентификация и аутентификация

Основные понятия

Идентификацию и *аутентификацию* можно считать основой программно-технических средств безопасности, поскольку остальные сервисы рассчитаны на обслуживание именованных субъектов. *Идентификация* и *аутентификация* – это первая линия обороны, "проходная" информационного пространства организации.

Идентификация позволяет субъекту (пользователю, процессу, действующему от имени определенного пользователя, или иному аппаратно-программному компоненту) назвать себя (сообщить свое имя). Посредством ***аутентификации*** вторая сторона убеждается, что субъект действительно тот, за кого он себя выдает. В качестве синонима слова "*аутентификация*" иногда используют словосочетание "проверка подлинности".

(Заметим в скобках, что происхождение русскоязычного термина "*аутентификация*" не совсем понятно. Английское "authentication" скорее можно прочитать как "аутентикация"; трудно сказать, откуда в середине взялось еще "фи" – может, из *идентификации*? Тем не менее, термин устоялся, он закреплен в Руководящих документах Гостехкомиссии России, использован в многочисленных публикациях, поэтому исправить его уже невозможно.)

Аутентификация бывает ***односторонней*** (обычно клиент доказывает свою подлинность серверу) и ***двусторонней*** (***взаимной***). Пример *односторонней аутентификации* – процедура входа пользователя в систему.

В сетевой среде, когда стороны *идентификации*/*аутентификации* территориально разнесены, у рассматриваемого сервиса есть два основных аспекта:

- что служит ***аутентификатором*** (то есть используется для подтверждения подлинности субъекта);
- как организован (и защищен) обмен данными *идентификации*/*аутентификации*.

Субъект может подтвердить свою подлинность, предъявив по крайней мере одну из следующих сущностей:

- нечто, что он знает (пароль, личный *идентификационный* номер, криптографический ключ и т.п.);
- нечто, чем он владеет (личную карточку или иное устройство аналогичного назначения);
- нечто, что есть часть его самого (голос, отпечатки пальцев и т.п., то есть свои биометрические характеристики).

В открытой сетевой среде между сторонами *идентификации*/*аутентификации* не существует доверенного маршрута; это значит, что в общем случае данные, переданные субъектом, могут не совпадать с данными, полученными и использованными для проверки подлинности. Необходимо обеспечить защиту от пассивного и активного прослушивания сети, то есть от ***перехвата, изменения*** и/или ***воспроизведения*** данных. Передача паролей в открытом виде, очевидно, неудовлетворительна; не спасает положение и шифрование паролей, так как оно не защищает от *воспроизведения*. Нужны более сложные протоколы *аутентификации*.

Надежная *идентификация* затруднена не только из-за сетевых угроз, но и по целому ряду причин. Во-первых, почти все *аутентификационные* сущности можно узнать, украсть или подделать. Во-вторых, имеется противоречие между надежностью *аутентификации*, с одной стороны, и удобствами пользователя и системного администратора с другой. Так, из соображений безопасности необходимо с определенной частотой просить пользователя повторно вводить *аутентификационную* информацию (ведь на его место мог сесть другой человек), а это не только хлопотно, но и повышает

вероятность того, что кто-то может подсмотреть за вводом данных. В-третьих, чем надежнее средство защиты, тем оно дороже.

Современные средства *идентификации/аутентификации* должны поддерживать концепцию *единого входа в сеть*. *Единый вход в сеть* – это, в первую очередь, требование удобства для пользователей. Если в корпоративной сети много информационных сервисов, допускающих независимое обращение, то многократная *идентификация/аутентификация* становится слишком обременительной. К сожалению, пока нельзя сказать, что *единый вход в сеть* стал нормой, доминирующие решения пока не сформировались.

Таким образом, необходимо искать компромисс между надежностью, доступностью по цене и удобством использования и администрирования средств *идентификации* и *аутентификации*.

Любопытно отметить, что сервис *идентификации / аутентификации* может стать объектом атак на доступность. Если система сконфигурирована так, что после определенного числа неудачных попыток устройство ввода идентификационной информации (такое, например, как терминал) блокируется, то злоумышленник может остановить работу легального пользователя буквально несколькими нажатиями клавиш.

Парольная аутентификация

Главное достоинство ***парольной аутентификации*** – простота и привычность. Пароли давно встроены в операционные системы и иные сервисы. При правильном использовании пароли могут обеспечить приемлемый для многих организаций уровень безопасности. Тем не менее, по совокупности характеристик их следует признать самым слабым средством проверки подлинности.

Чтобы пароль был запоминающимся, его зачастую делают простым (имя подруги, название спортивной команды и т.п.). Однако простой пароль нетрудно угадать, особенно если знать пристрастия данного пользователя. Известна классическая история про советского разведчика Рихарда Зорге, объект внимания которого через слово говорил "карамба"; разумеется, этим же словом открывался сверхсекретный сейф.

Иногда пароли с самого начала не хранятся в тайне, так как имеют стандартные значения, указанные в документации, и далеко не всегда после установки системы производится их смена.

Ввод пароля можно подсмотреть. Иногда для подглядывания используются даже оптические приборы.

Пароли нередко сообщают коллегам, чтобы те могли, например, подменить на некоторое время владельца пароля. Теоретически в подобных случаях более правильно задействовать средства управления доступом, но на практике так никто не поступает; а тайна, которую знают двое, это уже не тайна.

Пароль можно угадать "методом грубой силы", используя, скажем, словарь. Если файл паролей зашифрован, но доступен для чтения, его можно скачать к себе на компьютер и попытаться подобрать пароль, запрограммировав полный перебор (предполагается, что алгоритм шифрования известен).

Тем не менее, следующие меры позволяют значительно повысить надежность парольной защиты:

- ***наложение технических ограничений*** (пароль должен быть не слишком коротким, он должен содержать буквы, цифры, знаки пунктуации и т.п.);
- ***управление сроком действия паролей***, их периодическая смена;
- ограничение доступа к файлу паролей;
- ограничение числа неудачных попыток входа в систему (это затруднит применение "метода грубой силы");
- обучение пользователей;

- использование программных ***генераторов паролей*** (такая программа, основываясь на несложных правилах, может порождать только благозвучные и, следовательно, запоминающиеся пароли).
- Перечисленные меры целесообразно применять всегда, даже если наряду с паролями используются другие методы аутентификации.

Одноразовые пароли

Рассмотренные выше **пароли можно назвать многоразовыми**; их раскрытие позволяет злоумышленнику действовать от имени легального пользователя. Гораздо более сильным средством, устойчивым к пассивному прослушиванию сети, являются **одноразовые пароли**.

Наиболее известным программным *генератором одноразовых паролей* является система **S/KEY** компании Bellcore. Идея этой системы состоит в следующем. Пусть имеется **односторонняя функция** f (то есть функция, вычислить обратную которой за приемлемое время не представляется возможным). Эта функция известна и пользователю, и **серверу аутентификации**. Пусть, далее, имеется **секретный ключ** K, известный только пользователю.

На этапе начального администрирования пользователя функция f применяется к ключу K n раз, после чего результат сохраняется на сервере. После этого процедура проверки подлинности пользователя выглядит следующим образом:

- сервер присылает на пользовательскую систему число (n-1);
- пользователь применяет функцию f к секретному ключу K (n-1) раз и отправляет результат по сети на сервер аутентификации;
- сервер применяет функцию f к полученному от пользователя значению и сравнивает результат с ранее сохраненной величиной. В случае совпадения подлинность пользователя считается установленной, сервер запоминает новое значение (присланное пользователем) и уменьшает на единицу счетчик (n).

На самом деле реализация устроена чуть сложнее (кроме счетчика, сервер посылает затравочное значение, используемое функцией f), но для нас сейчас это не важно. Поскольку функция f необратима, *перехват* пароля, равно как и получение доступа к серверу *аутентификации*, не позволяют узнать секретный ключ K и предсказать следующий одноразовый пароль.

Система S/KEY имеет статус Internet-стандарта (RFC 1938).

Другой подход к надежной *аутентификации* состоит в *генерации нового пароля* через небольшой промежуток времени (например, каждые 60 секунд), для чего могут использоваться программы или специальные интеллектуальные карты (с практической точки зрения такие пароли можно считать одноразовыми). Серверу *аутентификации* должен быть известен алгоритм *генерации паролей* и ассоциированные с ним параметры; кроме того, часы клиента и сервера должны быть синхронизированы.

Сервер аутентификации Kerberos

Kerberos – это программный продукт, разработанный в середине 1980-х годов в Массачусетском технологическом институте и претерпевший с тех пор ряд принципиальных изменений. Клиентские компоненты Kerberos присутствуют в большинстве современных операционных систем.

Kerberos предназначен для решения следующей задачи. Имеется открытая (незащищенная) сеть, в узлах которой сосредоточены **субъекты** – пользователи, а также клиентские и серверные программные системы. Каждый субъект обладает секретным ключом. Чтобы субъект C мог доказать свою подлинность субъекту S (без этого S не станет обслуживать C), он должен не только назвать себя, но и продемонстрировать знание секретного ключа. C не может просто послать S свой секретный ключ, во-первых, потому, что сеть открыта (доступна для пассивного и активного прослушивания), а, во-

вторых, потому, что S не знает (и не должен знать) секретный ключ C. Требуется менее прямолинейный способ демонстрации знания секретного ключа.

Система Kerberos представляет собой **доверенную третью сторону** (то есть сторону, которой доверяют все), владеющую секретными ключами обслуживаемых субъектов и помогающую им в попарной проверке подлинности.

Чтобы с помощью Kerberos получить доступ к S (обычно это сервер), C (как правило – клиент) посылает Kerberos запрос, содержащий сведения о нем (клиенте) и о запрашиваемой услуге. В ответ Kerberos возвращает так называемый **билет**, зашифрованный секретным ключом сервера, и копию части информации из билета, зашифрованную секретным ключом клиента. Клиент должен расшифровать вторую порцию данных и переслать ее вместе с билетом серверу. Сервер, расшифровав билет, может сравнить его содержимое с дополнительной информацией, присланной клиентом. Совпадение свидетельствует о том, что клиент смог расшифровать предназначенные ему данные (ведь содержимое билета никому, кроме сервера и Kerberos, недоступно), то есть продемонстрировал знание секретного ключа. Значит, клиент – именно тот, за кого себя выдает. Подчеркнем, что секретные ключи в процессе проверки подлинности не передавались по сети (даже в зашифрованном виде) – они только использовались для шифрования. Как организован первоначальный обмен ключами между Kerberos и субъектами и как субъекты хранят свои секретные ключи – вопрос отдельный.

Проиллюстрируем описанную процедуру.

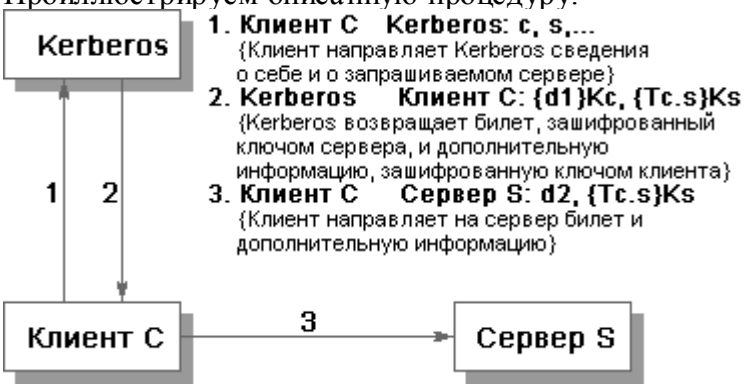

Рис. 10.1. Проверка сервером S подлинности клиента C.

Здесь c и s – сведения (например, имя), соответственно, о клиенте и сервере, d1 и d2 – дополнительная (по отношению к билету) информация, Tc.s – билет для клиента C на обслуживание у сервера S, Kc и Ks – секретные ключи клиента и сервера, {info}K – информация info, зашифрованная ключом K.

Приведенная схема – крайне упрощенная версия реальной процедуры проверки подлинности. Более подробное рассмотрение системы Kerberos можно найти, например, в статье В. Галатенко "Сервер аутентификации Kerberos (Jet Info, 1996, 12-13). Нам же важно отметить, что Kerberos не только устойчив к сетевым угрозам, но и поддерживает концепцию *единого входа в сеть*.

Идентификация/аутентификация с помощью биометрических данных

Биометрия представляет собой совокупность автоматизированных методов *идентификации* и/или *аутентификации* людей на основе их физиологических и поведенческих характеристик. К числу физиологических характеристик принадлежат особенности **отпечатков пальцев**, **сетчатки** и **роговицы** глаз, **геометрия руки и лица** и т.п. К поведенческим характеристикам относятся **динамика подписи** (ручной), стиль **работы с клавиатурой**. На стыке физиологии и поведения находятся анализ особенностей **голоса** и **распознавание речи**.

Биометрией во всем мире занимаются очень давно, однако долгое время все, что было связано с ней, отличалось сложностью и дороговизной. В последнее время спрос на биометрические продукты, в первую очередь в связи с развитием электронной коммерции,

постоянно и весьма интенсивно растет. Это понятно, поскольку с точки зрения пользователя гораздо удобнее предъявить себя самого, чем что-то запоминать. Спрос рождает предложение, и на рынке появились относительно недорогие аппаратно-программные продукты, ориентированные в основном на распознавание отпечатков пальцев.

В общем виде работа с биометрическими данными организована следующим образом. Сначала создается и поддерживается база данных характеристик потенциальных пользователей. Для этого биометрические характеристики пользователя снимаются, обрабатываются, и результат обработки (называемый **биометрическим шаблоном**) заносится в базу данных (исходные данные, такие как результат сканирования пальца или роговицы, обычно не хранятся).

В дальнейшем для *идентификации* (и одновременно *аутентификации*) пользователя процесс снятия и обработки повторяется, после чего производится поиск в базе данных шаблонов. В случае успешного поиска личность пользователя и ее подлинность считаются установленными. Для аутентификации достаточно произвести сравнение с одним биометрическим шаблоном, выбранным на основе предварительно введенных данных.

Обычно биометрию применяют вместе с другими *аутентификаторами*, такими, например, как интеллектуальные карты. Иногда биометрическая аутентификация является лишь первым рубежом защиты и служит для активизации интеллектуальных карт, хранящих криптографические секреты; в таком случае биометрический шаблон хранится на той же карте.

Активность в области биометрии очень велика. Организован соответствующий консорциум (см. http://www.biometrics.org/), активно ведутся работы по стандартизации разных аспектов технологии (формата обмена данными, прикладного программного интерфейса и т.п.), публикуется масса рекламных статей, в которых биометрия преподносится как средство обеспечения сверхбезопасности, ставшее доступным широким массам.

На наш взгляд, к биометрии следует относиться весьма осторожно. Необходимо учитывать, что она подвержена тем же угрозам, что и другие методы аутентификации. Во-первых, биометрический шаблон сравнивается не с результатом первоначальной обработки характеристик пользователя, а с тем, что пришло к месту сравнения. А, как известно, за время пути... много чего может произойти. Во-вторых, биометрические методы не более надежны, чем база данных шаблонов. В-третьих, следует учитывать разницу между применением биометрии на контролируемой территории, под бдительным оком охраны, и в "полевых" условиях, когда, например к устройству сканирования роговицы могут поднести муляж и т.п. В-четвертых, биометрические данные человека меняются, так что база шаблонов нуждается в сопровождении, что создает определенные проблемы и для пользователей, и для администраторов.

Но главная опасность состоит в том, что любая "пробоина" для биометрии оказывается фатальной. Пароли, при всей их ненадежности, в крайнем случае можно сменить. Утерянную аутентификационную карту можно аннулировать и завести новую. Палец же, глаз или голос сменить нельзя. Если биометрические данные окажутся скомпрометированы, придется как минимум производить существенную модернизацию всей системы.

Управление доступом

Основные понятия

С традиционной точки зрения средства управления доступом позволяют специфицировать и контролировать действия, которые субъекты (пользователи и процессы) могут выполнять над **объектами** (информацией и другими компьютерными ресурсами). В данном разделе речь идет о логическом управлении доступом, которое, в отличие от физического, реализуется программными средствами. Логическое управление

доступом – это основной механизм многопользовательских систем, призванный обеспечить конфиденциальность и целостность объектов и, до некоторой степени, их доступность (путем запрещения обслуживания неавторизованных пользователей).

Рассмотрим формальную постановку задачи в традиционной трактовке. Имеется совокупность субъектов и набор объектов. Задача логического управления доступом состоит в том, чтобы для каждой пары "субъект-объект" определить множество допустимых операций (зависящее, быть может, от некоторых дополнительных условий) и контролировать выполнение установленного порядка.

Отношение "субъекты-объекты" можно представить в виде **матрицы доступа**, в строках которой перечислены субъекты, в столбцах – объекты, а в клетках, расположенных на пересечении строк и столбцов, записаны дополнительные условия (например, время и место действия) и разрешенные виды доступа. Фрагмент матрицы может выглядеть, например, так:

Таблица 10.1. Фрагмент матрицы доступа

	Файл	**Программа**	**Линия связи**	**Реляционная таблица**
Пользователь 1	orw с системной консоли	e	rw с 8:00 до 18:00	
Пользователь 2				a

"o" – обозначает разрешение на передачу **прав доступа** другим пользователям,
"r" – чтение,
"w" – запись,
"e" – выполнение,
"a" – добавление информации

Тема логического управления доступом – одна из сложнейших в области информационной безопасности. Дело в том, что само понятие объекта (а тем более видов доступа) меняется от сервиса к сервису. Для операционной системы к объектам относятся файлы, устройства и процессы. Применительно к файлам и устройствам обычно рассматриваются права на чтение, запись, выполнение (для программных файлов), иногда на удаление и добавление. Отдельным правом может быть возможность передачи полномочий доступа другим субъектам (так называемое право владения). Процессы можно создавать и уничтожать. Современные операционные системы могут поддерживать и другие объекты.

Для систем управления реляционными базами данных объект – это база данных, таблица, представление, хранимая процедура. К таблицам применимы операции поиска, добавления, модификации и удаления данных, у других объектов иные виды доступа.

Разнообразие объектов и применимых к ним операций приводит к принципиальной децентрализации логического управления доступом. Каждый сервис должен сам решать, позволить ли конкретному субъекту ту или иную операцию. Теоретически это согласуется с современным объектно-ориентированным подходом, на практике же приводит к значительным трудностям. Главная проблема в том, что ко многим объектам можно получить доступ с помощью разных сервисов (возможно, при этом придется преодолеть некоторые технические трудности). Так, до реляционных таблиц можно добраться не только средствами СУБД, но и путем непосредственного чтения файлов или дисковых разделов, поддерживаемых операционной системой (разобравшись предварительно в структуре хранения объектов базы данных). В результате при задании матрицы доступа нужно принимать во внимание не только принцип распределения привилегий для каждого сервиса, но и существующие связи между сервисами (приходится заботиться о согласованности разных частей матрицы). Аналогичная трудность возникает при экспорте/импорте данных, когда информация о правах доступа, как правило, теряется (поскольку на новом сервисе она не имеет смысла). Следовательно, обмен данными между различными сервисами представляет особую опасность с точки зрения управления

доступом, а при проектировании и реализации разнородной конфигурации необходимо позаботиться о согласованном распределении прав доступа субъектов к объектам и о минимизации числа способов экспорта/импорта данных.

При принятии решения о предоставлении доступа обычно анализируется следующая информация:

- идентификатор субъекта (идентификатор пользователя, сетевой адрес компьютера и т.п.). Подобные идентификаторы являются основой **произвольного (или дискреционного) управления доступом**;
- атрибуты субъекта (метка безопасности, группа пользователя и т.п.). Метки безопасности – основа **принудительного (мандатного) управления доступом.**

Матрицу доступа, ввиду ее разреженности (большинство клеток – пустые), неразумно хранить в виде двухмерного массива. Обычно ее хранят по столбцам, то есть для каждого объекта поддерживается список "допущенных" субъектов вместе с их правами. Элементами списков могут быть имена групп и шаблоны субъектов, что служит большим подспорьем администратору. Некоторые проблемы возникают только при удалении субъекта, когда приходится удалять его имя из всех списков доступа; впрочем, эта операция производится не часто.

Списки доступа – исключительно гибкое средство. С их помощью легко выполнить требование о гранулярности прав с точностью до пользователя. Посредством списков несложно добавить права или явным образом запретить доступ (например, чтобы наказать нескольких членов группы пользователей). Безусловно, списки являются лучшим средством произвольного управления доступом.

Подавляющее большинство операционных систем и систем управления базами данных реализуют именно произвольное управление доступом. Основное достоинство произвольного управления – гибкость. Вообще говоря, для каждой пары "субъект-объект" можно независимо задавать права доступа (особенно легко это делать, если используются **списки управления доступом**). К сожалению, у "произвольного" подхода есть ряд недостатков. Рассредоточенность управления доступом ведет к тому, что доверенными должны быть многие пользователи, а не только системные операторы или администраторы. Из-за рассеянности или некомпетентности сотрудника, владеющего секретной информацией, эту информацию могут узнать и все остальные пользователи. Следовательно, произвольность управления должна быть дополнена жестким контролем за реализацией избранной политики безопасности.

Второй недостаток, который представляется основным, состоит в том, что права доступа существуют отдельно от данных. Ничто не мешает пользователю, имеющему доступ к секретной информации, записать ее в доступный всем файл или заменить полезную утилиту ее "троянским" аналогом. Подобная "разделенность" прав и данных существенно осложняет проведение несколькими системами согласованной политики безопасности и, главное, делает практически невозможным эффективный контроль согласованности.

Возвращаясь к вопросу представления матрицы доступа, укажем, что для этого можно использовать также функциональный способ, когда матрицу не хранят в явном виде, а каждый раз вычисляют содержимое соответствующих клеток. Например, при принудительном управлении доступом применяется сравнение меток безопасности субъекта и объекта.

Удобной надстройкой над средствами логического управления доступом является **ограничивающий интерфейс**, когда пользователя лишают самой возможности попытаться совершить несанкционированные действия, включив в число видимых ему объектов только те, к которым он имеет доступ. Подобный подход обычно реализуют в рамках системы меню (пользователю показывают лишь допустимые варианты выбора) или посредством ограничивающих оболочек, таких как restricted shell в ОС Unix.

В заключение подчеркнем важность управления доступом не только на уровне операционной системы, но и в рамках других сервисов, входящих в состав современных приложений, а также, насколько это возможно, на "стыках" между сервисами. Здесь на первый план выходит существование единой политики безопасности организации, а также квалифицированное и согласованное системное администрирование.

Ролевое управление доступом

При большом количестве пользователей традиционные подсистемы управления доступом становятся крайне сложными для администрирования. Число связей в них пропорционально произведению количества пользователей на количество объектов. Необходимы решения в объектно-ориентированном стиле, способные эту сложность понизить.

Таким решением является **ролевое управление доступом** (РУД). Суть его в том, что между пользователями и их привилегиями появляются промежуточные сущности – роли. Для каждого пользователя одновременно могут быть активными несколько ролей, каждая из которых дает ему определенные права (см. рис. 10.2).

Рис. 10.2. Пользователи, объекты и роли.

Ролевой доступ нейтрален по отношению к конкретным видам прав и способам их проверки; его можно рассматривать как объектно-ориентированный каркас, облегчающий администрирование, поскольку он позволяет сделать подсистему разграничения доступа управляемой при сколь угодно большом числе пользователей, прежде всего за счет установления между ролями связей, аналогичных наследованию в объектно-ориентированных системах. Кроме того, ролей должно быть значительно меньше, чем пользователей. В результате число администрируемых связей становится пропорциональным сумме (а не произведению) количества пользователей и объектов, что по порядку величины уменьшить уже невозможно.

Ролевой доступ развивается более 10 лет (сама идея ролей, разумеется, значительно старше) как на уровне операционных систем, так и в рамках СУБД и других информационных сервисов. В частности, существуют реализации ролевого доступа для Web-серверов.

В 2001 году Национальный институт стандартов и технологий США предложил проект стандарта ролевого управления доступом (см. http://csrc.nist.gov/rbac/), основные положения которого мы и рассмотрим.

Ролевое управление доступом оперирует следующими основными понятиями:

- **пользователь** (человек, интеллектуальный автономный агент и т.п.);
- **сеанс работы пользователя**;
- **роль** (обычно определяется в соответствии с организационной структурой);
- **объект** (сущность, доступ к которой разграничивается; например, файл ОС или таблица СУБД);
- **операция** (зависит от объекта; для файлов ОС – чтение, запись, выполнение и т.п.; для таблиц СУБД – вставка, удаление и т.п., для прикладных объектов операции могут быть более сложными);

- право доступа (разрешение выполнять определенные операции над определенными объектами).

Ролям приписываются пользователи и права доступа; можно считать, что они (роли) именуют отношения "многие ко многим" между пользователями и правами. Роли могут быть приписаны многим пользователям; один пользователь может быть приписан нескольким ролям. Во время сеанса работы пользователя активизируется подмножество ролей, которым он приписан, в результате чего он становится обладателем объединения прав, приписанных активным ролям. Одновременно пользователь может открыть несколько сеансов.

Между ролями может быть определено отношение частичного порядка, называемое наследованием. Если роль r2 является наследницей r1, то все права r1 приписываются r2, а все пользователи r2 приписываются r1. Очевидно, что **наследование ролей** соответствует наследованию классов в объектно-ориентированном программировании, только правам доступа соответствуют методы классов, а пользователям – объекты (экземпляры) классов.

Отношение наследования является иерархическим, причем права доступа и пользователи распространяются по уровням иерархии навстречу друг другу. В общем случае наследование является множественным, то есть у одной роли может быть несколько предшественниц (и, естественно, несколько наследниц, которых мы будем называть также преемницами).

Можно представить себе формирование **иерархии ролей**, начиная с минимума прав (и максимума пользователей), приписываемых роли "сотрудник", с постепенным уточнением состава пользователей и добавлением прав (роли "системный администратор", "бухгалтер" и т.п.), вплоть до роли "руководитель" (что, впрочем, не значит, что руководителю предоставляются неограниченные права; как и другим ролям, в соответствии с принципом **минимизации привилегий**, этой роли целесообразно разрешить только то, что необходимо для выполнения служебных обязанностей). Фрагмент подобной иерархии ролей показан на рис. 10.3.

Рис. 10.3. Фрагмент иерархии ролей.

Для реализации еще одного упоминавшегося ранее важного принципа информационной безопасности вводится понятие **разделения обязанностей**, причем в двух видах: статическом и динамическом.

Статическое разделение обязанностей налагает ограничения на **приписывание пользователей ролям**. В простейшем случае членство в некоторой роли запрещает приписывание пользователя определенному множеству других ролей. В общем случае данное ограничение задается как пара "множество ролей – число" (где множество состоит, по крайней мере, из двух ролей, а число должно быть больше 1), так что никакой пользователь не может быть приписан указанному (или большему) числу ролей из заданного множества. Например, может существовать пять бухгалтерских ролей, но политика безопасности допускает членство не более чем в двух таких ролях (здесь число=3).

При наличии наследования ролей ограничение приобретает несколько более сложный вид, но суть остается простой: при проверке членства в ролях нужно учитывать приписывание пользователей ролям-наследницам.

Динамическое разделение обязанностей отличается от статического только тем, что рассматриваются роли, одновременно активные (быть может, в разных сеансах) для данного пользователя (а не те, которым пользователь статически приписан). Например, один пользователь может играть роль и кассира, и контролера, но не одновременно; чтобы стать контролером, он должен сначала закрыть кассу. Тем самым реализуется так называемое **временное ограничение доверия**, являющееся аспектом минимизации привилегий.

Рассматриваемый проект стандарта содержит спецификации трех категорий функций, необходимых для администрирования РУД:

- **Административные функции** (создание и сопровождение ролей и других атрибутов ролевого доступа): создать/удалить роль/пользователя, приписать пользователя/право роли или ликвидировать существующую ассоциацию, создать/удалить отношение наследования между существующими ролями, создать новую роль и сделать ее наследницей/предшественницей существующей роли, создать/удалить ограничения для статического/динамического разделения обязанностей.
- **Вспомогательные функции** (обслуживание сеансов работы пользователей): открыть сеанс работы пользователя с активацией подразумеваемого набора ролей; активировать новую роль, деактивировать роль; проверить правомерность доступа.
- **Информационные функции** (получение сведений о текущей конфигурации с учетом отношения наследования). Здесь проводится разделение на обязательные и необязательные функции. К числу первых принадлежат получение списка пользователей, приписанных роли, и списка ролей, которым приписан пользователь.

Все остальные функции отнесены к разряду необязательных. Это получение информации о правах, приписанных роли, о правах заданного пользователя (которыми он обладает как член множества ролей), об активных в данный момент сеанса ролях и правах, об операциях, которые роль/пользователь правомочны совершить над заданным объектом, о статическом/динамическом разделении обязанностей.

Можно надеяться, что предлагаемый стандарт поможет сформировать единую терминологию и, что более важно, позволит оценивать РУД-продукты с единых позиций, по единой шкале.

Управление доступом в Java-среде

Java – это объектно-ориентированная система программирования, поэтому и управление доступом в ней спроектировано и реализовано в объектном стиле. По этой причине рассмотреть Java-среду для нас очень важно. Подробно о Java-технологии и безопасности Java-среды рассказано в статье А. Таранова и В. Цишевского "Java в три года" (Jet Info, 1998, 11-12). С разрешения авторов далее используются ее фрагменты.

Прежде всего, остановимся на эволюции модели безопасности Java. В JDK 1.0 была предложена концепция **"песочницы"** (sandbox) – замкнутой среды, в которой выполняются потенциально ненадежные программы (**апплеты**, поступившие по сети). Программы, располагающиеся на локальном компьютере, считались абсолютно надежными, и им было доступно все, что доступно виртуальной Java-машине.

В число ограничений, налагаемых "песочницей", входит запрет на доступ к локальной файловой системе, на сетевое взаимодействие со всеми хостами, кроме источника апплета, и т.п. Независимо от уровня достигаемой при этом безопасности (а проблемы возникали и с разделением свой/чужой, и с определением источника апплета), наложенные ограничения следует признать слишком обременительными: возможности для содержательных действий у апплетов почти не остается.

Чтобы справиться с этой проблемой, в JDK 1.1 ввели деление источников (точнее, распространителей) апплетов на надежные и ненадежные (источник определялся по

электронной подписи). Надежные апплеты приравнивались в правах к "родному" коду. Сделанное послабление решило проблемы тех, кому прав не хватало, но защита осталась неэшелонированной и, следовательно, неполной.

В JDK 1.2 сформировалась модель безопасности, используемая и в Java 2. От модели "песочницы" отказались. Оформились три основных понятия:

- **источник программы**;
- право и **множество прав**;
- **политика безопасности**.

Источник программы определяется парой (URL, распространители программы). Последние задаются набором цифровых сертификатов.

Право – это абстрактное понятие, за которым, как и положено в **объектной среде**, стоят классы и объекты. В большинстве случаев право определяется двумя цепочками символов – именем ресурса и действием. Например, в качестве ресурса может выступать файл, а в качестве действия – чтение. Важнейшим **методом** "правовых" объектов является implies(). Он проверяет, следует ли одно право (запрашиваемое) из другого (имеющегося).

Политика безопасности задает соответствие между источником и правами поступивших из него программ (формально можно считать, что каждому источнику соответствует своя "песочница"). В JDK 1.2 "родные" программы не имеют каких-либо привилегий в плане безопасности, и политика по отношению к ним может быть любой. В результате получился традиционный для современных ОС и СУБД механизм прав доступа со следующими особенностями:

- Java-программы выступают не от имени пользователя, их запустившего, а от имени источника программы. (Это весьма глубокая и прогрессивная трактовка, если ее правильно развить, см. следующий раздел);
- нет понятия владельца ресурсов, который мог бы менять права; последние задаются исключительно политикой безопасности (формально можно считать, что владельцем всего является тот, кто формирует политику);
- механизмы безопасности снабжены объектной оберткой.

Весьма важным понятием в модели безопасности JDK 1.2 является **контекст выполнения**. Когда виртуальная Java-машина проверяет права доступа объекта к системному ресурсу, она рассматривает не только текущий объект, но и предыдущие элементы стека вызовов. Доступ предоставляется только тогда, когда нужным правом обладают все объекты в стеке. Разработчики Java называют это реализацией принципа минимизации привилегий.

На первый взгляд, учет контекста представляется логичным. Нельзя допускать, чтобы вызов какого-либо метода расширял права доступа хотя бы по той причине, что доступ к системным ресурсам осуществляется не напрямую, а с помощью системных объектов, имеющих все права.

К сожалению, подобные доводы противоречат одному из основных принципов объектного подхода – принципу инкапсуляции. Если объект А обращается к объекту В, он не может и не должен знать, как реализован В и какими ресурсами он пользуется для своих целей. Если А имеет право вызывать какой-либо метод В с некоторыми значениями аргументов, В обязан обслужить вызов. В противном случае при формировании политики безопасности придется учитывать возможный граф вызовов объектов, что, конечно же, нереально.

Разработчики Java осознавали эту проблему. Чтобы справиться с ней, они ввели понятие **привилегированного интервала программы**. При выполнении такого интервала контекст игнорируется. Привилегированная программа отвечает за себя, не интересуясь предысторией. Аналогом привилегированных программ являются файлы с битами переустановки идентификатора пользователя/группы в ОС Unix, что лишний раз подтверждает традиционность подхода, реализованного в JDK 1.2. Известны угрозы

безопасности, которые привносят подобные файлы. Теперь это не лучшее средство ОС Unix перекочевало в Java.

Рассмотрим дисциплину контроля прав доступа более формально.

Класс AccessController (встроенный менеджер безопасности) предоставляет единый метод для проверки заданного права в текущем контексте – checkPermission (Permission). Это лучше (по причине параметризуемости), чем множество методов вида checkXXX, присутствующих в SecurityManager – динамически изменяемом менеджере безопасности из ранних версий JDK.

Пусть текущий контекст выполнения состоит из N стековых фреймов (верхний соответствует методу, вызвавшему checkPermission(p)). Метод checkPermission реализует следующий алгоритм (см. Листинг 10.1).

```
i = N;
while (i > 0) {
   if (метод, породивший i-й фрейм, не имеет проверяемого
   права) {
     throw AccessControlException
     } else if (i-й фрейм помечен как привилегированный) {
       return;
     }
   i = i – 1;
};
// Выясним, есть ли проверяемое право у унаследованного контекста
inheritedContext.checkPermission (p);
```

Листинг 10.1. Алгоритм работы метода checkPermission класса AccessController. (html, txt)

Сначала в стеке ищется фрейм, не обладающий проверяемым правом. Проверка производится до тех пор, пока либо не будет исчерпан стек, либо не встретится "привилегированный" фрейм, созданный в результате обращения к методу doPrivileged(PrivilegedAction) класса AccessController. Если при порождении текущего потока выполнения был сохранен контекст inheritedContext, проверяется и он. При положительном результате проверки метод checkPermission(p) возвращает управление, при отрицательном возникает исключительная ситуация AccessControlException.

Выбранный подход имеет один недостаток – тяжеловесность реализации. В частности, при порождении нового потока управления с ним приходится ассоциировать зафиксированный "родительский" контекст и, соответственно, проверять последний в процессе контроля прав доступа.

Отметим, что этот подход не распространяется на распределенный случай (хотя бы потому, что контекст имеет лишь локальный смысл, как, впрочем, и политика безопасности).

В целом средства управления доступом в JDK 1.2 можно оценить как "наполовину объектные". Реализация оформлена в виде **интерфейсов** и классов, однако по-прежнему разграничивается доступ к необъектным сущностям – ресурсам в традиционном понимании. Не учитывается семантика доступа. Имеют место и другие отмеченные выше концептуальные проблемы.

Возможный подход к управлению доступом в распределенной объектной среде

Представляется, что в настоящее время проблема управления доступом существует в трех почти не связанных между собой проявлениях:

- традиционные модели (дискреционная и мандатная);
- модель "песочница" (предложенная для Java-среды и близкой ей системы Safe-Tcl);
- модель фильтрации (используемая в межсетевых экранах).

На наш взгляд, необходимо объединить существующие подходы на основе их развития и обобщения.

Формальная постановка задачи разграничения доступа может выглядеть следующим образом.

Рассматривается множество объектов (в смысле объектно-ориентированного программирования). Часть объектов может являться **контейнерами**, группирующими объекты-компоненты, задающими для них общий контекст, выполняющими общие функции и реализующими перебор компонентов. Контейнеры либо вложены друг в друга, либо не имеют общих компонентов.

С каждым объектом ассоциирован набор интерфейсов, снабженных **дескрипторами (ДИ)**. К объекту можно обратиться только посредством ДИ. Разные интерфейсы могут предоставлять разные методы и быть доступными для разных объектов.

Каждый контейнер позволяет опросить набор ДИ объектов-компонентов, удовлетворяющих некоторому условию. Возвращаемый результат в общем случае зависит от вызывающего объекта.

Объекты изолированы друг от друга. Единственным видом межобъектного взаимодействия является вызов метода.

Предполагается, что используются надежные средства аутентификации и защиты коммуникаций. В плане разграничения доступа локальные и удаленные вызовы не различаются.

Предполагается также, что разрешение или запрет на доступ не зависят от возможного параллельного выполнения методов (синхронизация представляет отдельную проблему, которая здесь не рассматривается).

Разграничивается доступ к интерфейсам объектов, а также к методам объектов (с учетом значений **фактических параметров** вызова). **Правила разграничения доступа (ПРД)** задаются в виде **предикатов** над объектами.

Рассматривается задача разграничения доступа для выделенного контейнера СС, компонентами которого должны являться вызывающий и/или вызываемый объекты. ДИ этого контейнера полагается общеизвестным. Считается также, что между внешними по отношению к выделенному контейнеру объектами возможны любые вызовы.

Выполнение ПРД контролируется **монитором обращений**.

При вызове метода мы будем разделять действия, производимые вызывающим объектом (инициация вызова) и вызываемым методом (прием и завершение вызова).

При инициации вызова может производиться преобразование ДИ фактических параметров к виду, доступному вызываемому методу (**"трансляция интерфейса"**). Трансляция может иметь место, если вызываемый объект не входит в тот же контейнер, что и вызывающий.

Параметры методов могут быть входными и/или выходными. При приеме вызова возникает информационный поток из входных параметров в вызываемый объект. В момент завершения вызова возникает информационный поток из вызываемого объекта в выходные параметры. Эти потоки могут фигурировать в правилах разграничения доступа.

Структурируем множество всех ПРД, выделив четыре группы правил:

- политика безопасности контейнера;
- ограничения на вызываемый метод;
- ограничения на вызывающий метод;
- **добровольно налагаемые ограничения**.

Правила, общие для всех объектов, входящих в контейнер С, назовем политикой безопасности данного контейнера.

Пусть метод M1 объекта O1 в точке P1 своего выполнения должен вызвать метод M объекта O. Правила, которым должен удовлетворять M, можно разделить на четыре следующие подгруппы:

- правила, описывающие требования к **формальным параметрам** вызова;
- правила, описывающие требования к семантике M;
- реализационные правила, накладывающие ограничения на возможные реализации M;
- правила, накладывающие ограничения на вызываемый объект O.

Метод M объекта O, потенциально доступный для вызова, может предъявлять к вызывающему объекту следующие группы требований:

- правила, описывающие требования к фактическим параметрам вызова;
- правила, накладывающие ограничения на вызывающий объект.

Можно выделить три разновидности предикатов, соответствующих семантике и/или особенностям реализации методов:

- утверждения о фактических параметрах вызова метода M в точке P1;
- предикат, описывающий семантику метода M;
- предикат, описывающий особенности реализации метода M.

Перечисленные ограничения можно назвать добровольными, поскольку они соответствуют реальному поведению объектов и не связаны с какими-либо внешними требованиями.

Предложенная постановка задачи разграничения доступа соответствует современному этапу развития программирования, она позволяет выразить сколь угодно сложную политику безопасности, найти баланс между богатством выразительных возможностей и эффективностью работы монитора обращений.