DOI 10.17223/20710410/7/4
УДК 004.432.2

ВНЕДРЕНИЕ ПОЛИТИК БЕЗОПАСНОСТИ В КОМПЬЮТЕРНЫЕ СИСТЕМЫ МЕТОДОМ АОП НА ПРИМЕРЕ FTP-СЕРВЕРА APACHE

Д. А. Стефанцов, А. Е. Филимонов

Томский государственный университет, г. Томск, Россия

E-mail: dastephantsov@mail.tsu.ru, filimonov1987@gmail.com

Аспектно-ориентированное программирование (АОП) является одной из важнейших парадигм программирования в области написания политик безопасности для компьютерных систем (КС). С его помощью удаётся модифицировать и встраивать политики безопасности в защищаемую КС без модификации её исходного кода. В данной статье авторы на основе собственного опыта работы в этом направлении формулируют основные требования и некоторые рекомендации к реализации и успешному внедрению политик безопасности в защищаемые КС методом АОП. По этим рекомендациям осуществлено внедрение базовой политики ролевого разграничения доступа в FTP-сервер Apache. Технология этого внедрения подробно изложена в статье.

Ключевые слова: *компьютерные системы, политика безопасности, аспектно-ориентированное программирование, Apache Ftp Server.*

Введение

Реализации политик безопасности (ПБ) можно встретить во многих компьютерных системах — от операционных систем общего назначения [1] до узкоспециализированных Web-систем хранения данных.

Несмотря на годы практики, большинство методов реализации ПБ являются грубым нарушением принципа «разделения аспектов» изучаемой области [2]. Процесс реализации защищённого приложения традиционными методами предполагает как совместное проектирование ПБ и основной части приложения, так и их совместную реализацию. Если впоследствии появится необходимость в изменении политики безопасности, то внедрение этих изменений будет осложнено необходимостью анализа, повторного проектирования и реализации всех точек пересечения модулей ПБ и главной части приложения.

Отметим, что изменение ПБ зачастую является единственной адекватной мерой противодействия новым видам атак и/или угроз, и, несмотря на это, политики безопасности современных компьютерных систем остаются неизменными. Вероятно, причиной тому служит излишняя сложность внесения изменений.

Некоторые исследователи способов проектирования программного обеспечения (ПО) сходятся во мнении, что наличие данных препятствий к модификации программы является следствием ограничений, накладываемых современными средствами реализации, в частности существующими парадигмами программирования [3, 4]. В этих же источниках указывается один из возможных путей решения этой проблемы — аспектно-ориентированное программирование (АОП).

Согласно принципу модульности АОП, вся функциональность политики безопасности должна располагаться в отдельных модулях — аспектах. Важной особенностью

данного принципа является возможность разработки ПБ отдельно от основной функциональности программы, без изменения исходных кодов последней. Один из путей осуществления такой возможности описан в [5]. На этом пути для успешного присоединения аспекта политики безопасности к уже существующей КС необходимо отдельно от КС реализовать в виде модуля основную функциональность ПБ и соединительный модуль, который бы непосредственно связывал между собой модуль КС и модуль ПБ. Именно присоединение ПБ к КС при помощи соединительного модуля без модификации исходного кода КС и понимается здесь под внедрением ПБ в защищаемую КС.

В настоящей статье авторы на основе собственного опыта работы в этом направлении формулируют в разд. 2 и 4 некоторые рекомендации к реализации и успешному присоединению ПБ к защищаемым КС методом АОП. По этим рекомендациям осуществлено внедрение базовой политики ролевого разграничения доступа (РРД) [6] в FTP-сервер Apache. Технология этого внедрения продемонстрирована в разд. 6 и 7, где описываются соответственно реализация основной функциональности РРД — пакет RBAC, содержащий реализации на языке Java основных объектов политики РРД, таких, как Пользователь (User.java), Роль (Role.java) и т. д., и реализация пакета RBAC-AJ — соединительного модуля между сервером и пакетом RBAC и инициализация исходных данных политики РРД сервера — файл MainConf.xml. Необходимые понятия из АОП и сведения о FTP-сервере Apache приводятся в разд. 3 и 5 соответственно. Изложение начинается с краткой характеристики компьютерных систем, защита которых возможна подобным образом.

1. Особенности защищаемых компьютерных систем

АОП является удобным и перспективным инструментом для разработки и внедрения политик безопасности в КС. Однако его использование накладывает ряд ограничений на защищаемую компьютерную систему, а также на модуль ПБ. Первое и самое важное ограничение состоит в доступности исходных кодов защищаемой КС, что позволяет детально изучить заложенную в ней функциональность и определить необходимые срезы точек выполнения, которые будут использоваться для соединения компьютерной системы с политикой безопасности. В эпоху развития свободного программного обеспечения (СПО) данное ограничение не является препятствием для внедрения политики безопасности и позволяет использовать все преимущества СПО.

Поскольку методом внедрения политики выбрано аспектно-ориентированное программирование, то данный способ применим только к тем программным продуктам, аспектные расширения языка которых имеют законченный вид и библиотеки функций в которых полностью отлажены. Например, аспектное расширение языка Java [7] — AspectJ является родоначальником АОП, его библиотеки хорошо отлажены и находятся в свободном доступе [8]. Однако этого нельзя сказать про PHPAspect [9].

2. Рекомендации по разработке и реализации политики безопасности

В настоящее время трудно представить себе приложение, которое не использовало бы механизм передачи данных по сети. Следовательно, необходимо протестировать приложение на возможные ошибки при многопоточном использовании функционала политики безопасности. Некоторые языки программрования предоставляют встроенную возможность синхронизации доступа к ячейкам памяти и критическим участкам кода [1].

Одной из рекомендаций к реализации политики безопасности является наличие двух уровней реализации. Первый — это абстрактная модель, которая представляет

собой реализацию ПБ со всем ее содержимым в соответствии с необходимой для этого теорией и ассоциациями между элементами политики безопасности. Например, ассоциация между элементами Пользователь и Роль базовой политики РРД [6] имеет следующий вид: пользователь может быть авторизован на одну или большее число ролей, и на некоторую роль может быть авторизовано как несколько пользователей, так и ни одного. Для удобства программирования можно построить UML-диаграммы [10] в соответствии с теорией.

На практике для этого требования подходят механизм наследования и наличие абстрактных классов объектно-ориентированного программирования (ООП).

Вторым уровнем должно стать доопределение данной модели в соответствии с защищаемой компьютерной системой. Иначе говоря, необходимо унаследовать классы, описывающие объекты защищаемой компьютерной системы, от соответствующих классов абстрактной модели. Тем самым достигается свойство универсальности реализации политики безопасности. В случае написания политики для другой компьютерной системы абстрактная модель сохранит свою функциональность, и отпадет необходимость переписывать весь код политики, что является ценным качеством программного продукта в связи со всё ужесточающейся борьбой между нападением на компьютерные системы и ее защитой.

Следующим пожеланием к реализации ПБ является возможность гибкого конфигурирования данной политики. Наилучшим для этого способом является вынесение всех важных константных значений в некоторый конфигурационный файл. На данное время общепризнанным стандартом формирования таких файлов является формат XML [11]. Для многих языков существуют библиотеки, представляющие собой XML-парсеры, то есть синтаксические анализаторы конфигурационных файлов. Они позволяют представить содержащуюся в файле информацию в виде, понятном обработчикам этой информации. Следовательно, необходимо таким образом определить расположение информации в тегах XML, чтобы оно содержало всю необходимую конфигурационную информацию и удовлетворяло спецификациям формата XML и чтобы существовала возможность удобного добавления/удаления записей из файла.

Существуют две принципиально различные модели разбора XML-документов — SAX (Simple Api for Xml) и DOM (Document Object Model). Кратко отметим основные особенности этих двух подходов.

DOM позволяет представить данные, заключенные в XML-файле, в виде иерархической (древовидной) объектной структуры. DOM создает дерево с узлами на основе информации, заключенной в файле. В дальнейшем для доступа к информации необходимо производить операции с этим деревом. Очевидно, что данная модель разбора XML-файла ресурсоемкая в смысле расхода памяти для хранения данного дерева.

SAX позволяет представить процесс разбора XML-файла в виде череды событий, генерируемых функцией разбора, использующей SAX. Эти события генерируются в процессе обхода данного файла разборщиком — например, при обнаружении открытия тега, его закрытия, а также некоторой информации, заключенной между этими тегами и т. п. В дальнейшем необходимо написать программы-обработчики данных событий.

Для разбора XML-файлов при реализации политики безопасности рекомендуется использовать SAX-подход, так как содержимое файла конфигурации, который будет подвергаться разбору, обычно хорошо укладывается в объектную модель (например, Конфигурация — Объект конфигурации).

3. Необходимые понятия из АОП

Точка выполнения программы (join point) — любая команда в любом алгоритме объектной системы.

Описание точек выполнения программы (pointcut) — конструкция языка программирования, задающая произвольное число точек выполнения программы. С помощью логических связок «и», «или», «не» можно составить композицию описаний точек выполнения программы, в свою очередь являющуюся описанием точек выполнения программы. Описание точек выполнения программы называется также *срезом точек выполнения программы.*

След описания точек выполнения программы (join point shadow) — это все точки выполнения программы, в данный момент подходящие под описание точек выполнения программы.

Предписание (advice) — алгоритм, который сопоставляется описанию точек выполнения программы и который выполняется всякий раз, когда выполнение программы достигает любой точки из следа описания.

Аспект (aspect) — модуль или совокупность модулей программирования, которые реализуют тот или иной аспект приложения.

Для взаимодействия политики безопасности и КС могут быть использованы два очень полезных свойства АОП — рефлексия и интроспекция.

Рефлексия — это способность программы менять собственную структуру или поведение.

Интроспекция — это способность программы определять свойства объектов во время её выполнения.

Метод интроспекции является весьма полезным способом реализации АОП, так как с его с помощью можно отличать выполнение аспекта от объекта к объекту, узнавать свойства текущего объекта.

4. Рекомендации по разработке и реализации соединительного модуля

На данном этапе подразумевается наличие готовой политики безопасности, существующей отдельно от главного процесса защищаемой КС. Для её присоединения к КС требуется:

1) определить моменты логики работы защищаемой КС, при которых необходимо использование функционала политики безопасности. В данном пункте предстоит серьезная работа по изучению исходного кода компьютерной системы и построение UML-диаграмм полезных классов;
2) распределить указанные моменты по группам для последующего совместного описания на языке АОП;
3) выделить точки выполнения защищаемой компьютерной системы, соответствующие п. 1 и 2;
4) описать срезы точек выполнения на аспектном расширении того языка, на котором написана защищаемая компьютерная система;
5) запрограммировать предписания на описанные срезы точек выполнения.

Процесс функционирования политики безопасности предусматривает непосредственный обмен информацией между КС и политикой, поэтому требуется передача данных от пользователя к модулю политики в обход основной функциональности программы, так как модификация исходных кодов не предполагается в силу принятия принципа разделения аспектов. Так, например, при внедрении ПБ в некоторый сервер,

который имеет возможность пользовательского входа, можно перехватывать данные пользователя при аутентификации в КС этим методом.

Для этого существует один из видов предписаний — around. Он позволяет своему программному тексту быть исполненным вместо некоторого среза точек выполнения. Поэтому очень важно правильно определить точки выполнения и сгруппировать их.

Еще одним полезным свойством данного предписания является его использование в моментах связи программной реализации защищаемой КС и политики безопасности. В случае успешного прохождения проверок политики безопасности ход выполнения может перейти дальше на прерванную команду. В случае же отрицательного ответа от политики возможно прерывание работы компьютерной системы и выброс исключения.

В качестве иллюстрации процесса внедрения политики безопасности выберем в качестве защищаемой КС файловый сервер Apache Ftp Server.

5. Необходимые сведения об Apache Ftp Server

Данный FTP-сервер разработан компанией Apache Software Foundation для свободного использования [12]. Компания Apache позиционирует свою разработку как приложение, обладающее, помимо прочих, следующими особенностями, важными для выполняемой работы:

— сервер полностью разработан на Java;
— виртуальная домашняя директория для каждого пользователя, запрос на возможность записи в директорию;
— возможность анонимного доступа;
— данные о пользователе могут храниться как в файле, так и в базе данных;
— возможность запретить соединения с определенных IP-адресов.

В результате анализа исходных кодов сервера выяснилось, что в процессе его работы возможна потеря и утечка информации вследствие неправильного вынесения решения о предоставлении пользователю доступа к файлам. Для предотвращения этого предлагается внедрить в сервер базовую политику РРД [6] методом АОП. Ниже показывается, как это делается в соответствии с высказанными выше рекомендациями.

6. Реализация ролевого разграничения доступа в FTP-сервере Apache методом аспектно-ориентированного программирования

Реализация РРД в FTP-сервере Apache методом АОП сводится к выполнению следующих мероприятий, рассматриваемых далее подробно:

1) реализовать основную функциональность политики РРД на языке Java, на котором написан FTP-сервер, — модули User.java, Role.java, Session.java, Privilege.java, Query.java, Action.java, TargetDescriptor.java, TargetsDescriptor.java — в соответствии с теоретическими результатами в [6] и оформить в виде пакета RBAC;
2) реализовать механизм ограничений, предусмотренный политикой РРД, на аспектно-ориетированном расширении AspectJ языка Java — RoleUser.aj, RoleSession.aj, RolePrivilege.aj, SessionChange.aj, Test.aj и часть функциональности на Java — Fragmentation.java, PrivFragmentation.java, RoleFragmentation.java;
3) дополнить данную реализацию классами, которые носят специфичные черты защищаемой КС FTP-сервер Apache — FtpTargetsDescriptor.java, FtpTargetDescriptor.java, FtpPriv.java ;
4) реализовать механизм инициализации объектов политики РРД, таких, как пользователь, роль, право доступа и т. д. — SaxHandler.java;

5) написать соединительный модуль на языке AspectJ для связи главного процесса сервера и политики безопасности — пакет RBAC-AJ.

6.1. Реализация пакета RBAC

Основными объектами, согласно теории, в базовой модели РРД являются:

1) пользователь (в текущей релизации политики — класс User);
2) роль пользователя (Role);
3) сессия пользователя (Session);
4) право доступа (Privilege).

Для наглядности описываемых классов, связей и ассоциаций между ними можно построить диаграмму классов реализации РРД (рис. 1).

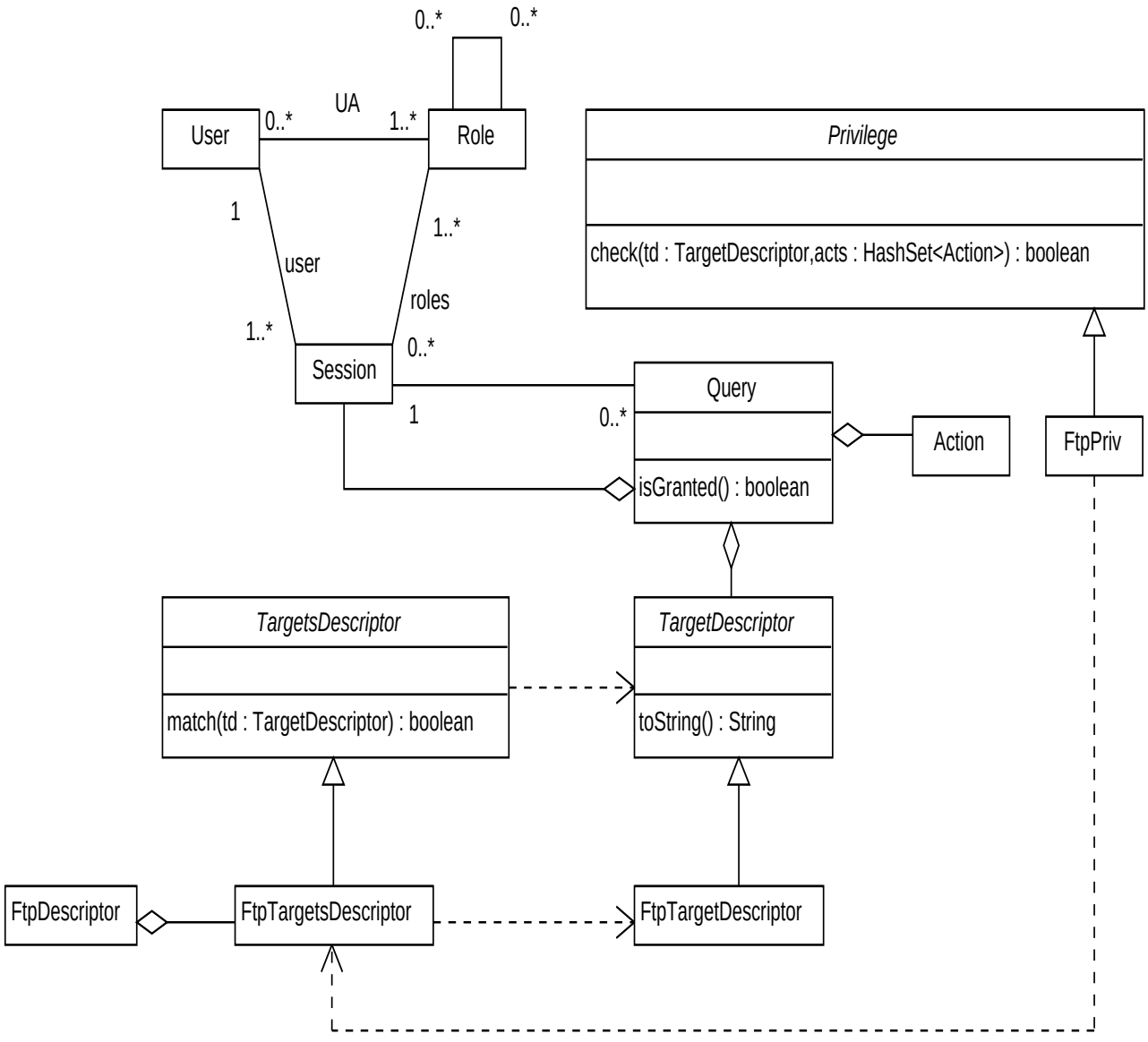

Рис. 1. Диаграмма классов реализации РРД

Для последующей поддержки любой КС, в работу которой может быть введена политика РРД, некоторые классы создаются абстрактными, что предоставляет возможность унаследовать от них класс объектов, имеющих непосредственное отношение к той КС, в работу которой внедряется данная реализация политики безопасности.

Заметим, что класс `Privilege` задуман абстрактным, так как для каждой компьютерной системы, в которую внедряется РРД, понятие «право доступа» определяется по-своему. Важной в данном классе является функция `check`. Эта функция проверяет, соответствует ли право доступа `this` тому, что передано аргументом. В данной реализации от этого класса был унаследован класс `FtpPriv`, который представляет собой право доступа на объект в хранилище FTP-сервера.

Само по себе право доступа реализуется как два объекта — набор из описателей объектов доступа пользователей — `TargetsDescriptor` и набор возможных действий над данными объектами, заданными описателями. Для FTP-сервера эти описатели реализованы в классе `FtpTargetsDescriptor` и представляют собой набор пар:

— регулярное выражение для обобщения некоторого количества имен файлов в хранилище;
— логическое значение, в итоге эмулирующее операцию разности множеств.

Для единичного файла введен класс `FtpTargetDescriptor`, который описывается именем. Для проверки, удовлетворяет ли данное имя конкретного файла регулярному выражению, реализована объявленная абстрактной в классе `TargetsDescriptor` функция `match`. Реализацию можно найти в классе `FtpTargetsDescriptor`.

Заметим, что после входа пользователя в систему для работы с сервером вся его деятельность описывается рядом запросов, которые и реализованы в виде экземпляров класса `Query`. Запрос представляет собой таблицу из двух строк и переменного числа колонок. В каждой колонке находится часть запроса — описатель файла (первая строка), к которому запрашивается некоторое множество действий (вторая строка).

Действие — также отдельный класс `Action`, экземпляры которого характеризуются типом — «read», «write» и т. д.

Согласно диаграмме и ассоциациям между классами, класс `User` представляет собой имя и множество ролей, которые ему разрешены для авторизации.

Согласно общей теории РРД, существует иерархия ролей, что и отражено на диаграмме ассоциацией класса `Role` на себя. В описываемой реализации каждая роль хранит набор потомков в виде экземпляра класса `ArrayList<Role> children`.

Для реализации ассоциации с `Privilege` в экземплярах класса Role хранится множество прав доступа, разрешенных данной роли.

В классе `Session` хранятся имя пользователя и список ролей, на которые он будет авторизован в данной сессии.

Во всех классах реализованы set/get методы.

6.2. Реализация ограничений на языке AspectJ и Java

В любой КС для того, чтобы механизмы политики безопасности функционировали не напрасно, необходимо введение некоторых ограничений на создаваемые и существующие объекты. Например, целесообразно ограничить количество пользователей, обладающих административными правами, до минимума, так как в противном случае о безопасности не может идти речи. Существуют и ограничения, которые необходимо реализовать в соответствии с теоретической моделью РРД:

1) количественное ограничение, связанное с максимальным числом пользователей, которые могут быть авторизованы на некоторую роль;

2) количественное ограничение, связанное с максимальным числом ролей пользователей, которые могут использовать некоторое право доступа;
3) количественное ограничение, связанное с максимальным числом сессий пользователей, которые могут быть авторизованы на некоторую роль;
4) статическое ограничение взаимного исключения ролей;
5) статическое ограничение взаимного исключения прав доступа;
6) динамическое ограничение взаимного исключения ролей;
7) статическое ограничение необходимости обладания ролью;
8) статическое ограничение необходимости обладания правом доступа;
9) динамическое ограничение необходимости обладания ролью.

Реализацию данных ограничений разобьем для удобства на три группы:

— реализация количественных ограничений;
— реализация ограничений взаимного исключения;
— реализация ограничений необходимости обладания.

Реализация количественных ограничений

Общий подход к реализации количественных ограничений одинаков — создать аспект с хранилищем элементов конкретно каждого ограничения, в котором количество элементов будет ограничиваться некоторой константой, полученной из конфигурационного файла.

Проиллюстрируем данный подход на примере первого количественного ограничения, связанного с числом пользователей, которые могут быть авторизованы на некоторую роль (см. листинг 1).

```
 1 import java.util.ArrayList;
 2 import java.util.HashSet;
 3 public aspect RoleUser {
 4  private ArrayList<User> Role.usersPerRole=
 5                            new ArrayList<User>();
 6  public int Role.getNumUsersPerRole()
 7  {  return usersPerRole.size(); }
 8  public boolean Role.isFullOfUsers()
 9  {if(Config.getHandler().getMaxUsersPerRole().get(getName())==
      null) return true;
10   return usersPerRole.size()==Config.getHandler().
11        getMaxUsersPerRole().get(getName());
12  }
13  public ArrayList<User> Role.getUsersPerRole()
14  {  return usersPerRole;   }
15  public void Role.addUser(User u)
16  {  usersPerRole.add(u); }
17  public boolean Role.hasNoUsers()
18  {  return usersPerRole.size()==0; }
19  public void Role.delUser(User u)
20  {  usersPerRole.remove(u); }
21 }
```

Листинг 1. Файл RoleUser.aj

В данном примере используется возможность AspectJ модифицировать статические свойства класса.

В строке 11 листинга 1 создается требуемая ограничением константа, считываемая из файла конфигурации. Далее в строке 5 создается хранилище — список пользователей, которые уже авторизованы на роль. Далее определяются функции для работы с хранилищем, а также проверки, заполнено хранилище или нет, и т. д.

Следующим этапом является написание предписания для применения данного ограничения. Для начала определяется срез точек выполнения, в которых необходимо выполнение предписания. В данном случае это будут вызовы функции из класса `User` — addRole(Role r). Необходимо заметить, что вызов этой функции может происходить как из контекста пользователя (когда объектом this является экземпляр класса User), так и в случае, когда вызывается эта функция явным указанием экземпляра класса User с использованием символа точки.

На языке AspectJ данные два среза будут выглядеть следующим образом:

```
1    public pointcut addingRole(User u,Role r):
2      (call(void addRole(Role)) && this(u)&& args(r));
3    public pointcut addingRole1(User u,Role r):
4      (call(void addRole(Role)) && target(u)&& args(r));
```

Листинг 2. Фрагмент файла Test.aj

Строка 2 в листинге 2 соответствует первому случаю, а строка 4 — второму.

Теперь объединим данные строчки в одну

«`public pointcut addingUserRole(Role r):(call(void addRole(Role)))&& args(r);`»

для того, чтобы не делать лишних предписаний. В дальнейшем будем разграничивать эти два случая при помощи свойства рефлексии — будем получать this и target прямо в момент точки выполнения следующим образом — см. строки 2–8 листинга 3.

Предписание будет выглядеть следующим образом:

```
1  before(Role r):addingUserRole(r)
2  { Object o=thisJoinPoint.getThis();
3   Object o1=thisJoinPoint.getTarget();
4   if((o instanceof User) || (o1 instanceof User)) {
5     User u;
6     RoleFragmentation f=Config.getHandler().
      getRoleFragmentation();
7     if (o instanceof User)
8        u=(User)o; else u=(User)o1;
9     try
10    { if (r.isFullOfUsers()||u.getRoles().contains(r))
11      throw new NUsersPerRoleConstraintCheckException();
12    }
13    catch (NUsersPerRoleConstraintCheckException ex)
14    { System.out.println(ex.toString()); }
15  }
16 }
```

Листинг 3. Фрагмент файла Test.aj

Для наглядности в случае неудовлетворения требованиям ограничения выбрасывается исключение — экземпляр собственного класса исключений, порожденный от стандартного класса `Exception`. Возможно в дальнейшем выдавать от сервера некоторый ответ со своим кодом.

Заполнить хранилище необходимо после того, как выполнены все ограничения. Для этого используется ключевое слово after (листинг 4):

```
1  after(Role r):addingUserRole(r)
2   {Object o=thisJoinPoint.getThis();
3    Object o1=thisJoinPoint.getTarget();
4     if ((o instanceof User)||(o1 instanceof User))
5     { User u;
6       if (o instanceof User) u=(User)o; else u=(User)o1;
7       r.addUser(u);
8     }
9   }
```

Листинг 4. Фрагмент файла Test.aj

Остальные количественные ограничения релизуются аналогично.

Реализация ограничений взаимного исключения

Общий подход к реализации ограничений взаимного исключения — создать класс, реализующий разбиение объектов на классы, и в этом классе написать функцию проверки того, можно ли добавить очередной объект в хранилище согласно разбиению.

Затем аналогично предыдущему — описание среза точек выполнения и написание предписания.

Рассмотрим случай статического взаимного исключения ролей. Для этого случая был написан абстрактный класс `Fragmentation<T>`. Его реализация показана на листинге 5. Он представляет собой реализацию разбиения множества экземпляров класса `T` на классы.

В основе — список в строке 4, элементы которого — классы разбиения. Далее реализованы основные операции с экземпляром данного класса. Основная работа возлагается на реализацию в потомках абстрактной функции проверки, какому классу принадлежит некоторый элемент.

```
1  import java.util.ArrayList;
2  import java.util.HashSet;
3  public abstract class Fragmentation<T> {
4    private ArrayList<HashSet<T>> frag;
5    public Fragmentation()
6    {   frag=new ArrayList<HashSet<T>>(); }
7    public Fragmentation(HashSet<T> r)
8    { frag=new ArrayList<HashSet<T>>();
9      frag.add(r);
10   }
11   public Fragmentation(ArrayList<HashSet<T>> rs)
12   {   frag=rs;  }
13   public ArrayList<HashSet<T>> getFrag()
14   {   return frag;  }
15   public void setFrag(ArrayList<HashSet<T>> rf)
16   {   frag=rf;  }
17   public HashSet<T> getBlock(Integer n)
18   { System.out.println("n="+n.toString());
19     return frag.get(n);
20   }
```

```
21    ...
22    public abstract Integer belongsTo(T element);
23  }
```

Листинг 5. Фрагмент файла Fragmentation.java

От класса `Fragmentation` наследуются два класса, необходимые для реализации ограничений `RoleFragmentation` и `PrivFragmentation`, которые отвечают за разбиение ролей и прав доступа на классы.

Рассмотрим `RoleFragmentation` (листинг 6). Реализация другого класса аналогична.

```
1  import java.util.ArrayList;
2  import java.util.HashSet;
3  public class RoleFragmentation extends Fragmentation<Role>{
4  public Integer belongsTo(Role r)
5    { for (Integer i=0;i<getFrag().size();i++)
6      { for (Role s:getFrag().get(i))
7          { if (s.getName().equals(r.getName()))
8            return i;
9          }
10     }
11     return -1;
12   }
13   public boolean checkAbility(User u,Role r)
14   { ArrayList<Role> userRoles=u.getRoles();
15     HashSet<Integer> blocks=new HashSet<Integer>();
16     Integer nB;
17     for (Role role:userRoles)
18     { nB=belongsTo(role);
19       if (nB!=-1)blocks.add(nB);
20     }
21     return (!blocks.contains(belongsTo(r)));
22   }
23   public boolean checkAbilityS(Session s,Role r)
24   { ArrayList<Role> sessionRoles=s.getAuthRoles();
25     HashSet<Integer> blocks=new HashSet<Integer>();
26     Integer nB;
27     for (Role role:sessionRoles)
28     { nB=belongsTo(role);
29       if (nB!=-1)blocks.add(nB);
30     }
31     return (!blocks.contains(belongsTo(r)));
32   }
33   public String toString()
34   { return getFrag().toString();  }
35   public void printBlockRoles(Integer n)
36   { if (getFrag().get(n)==null) System.out.println("No Such
      Block");
37     for (Role t:getFrag().get(n))
38     { System.out.println(t.getName()); }
39   }
40   public void print()
```

```
41    { for (Integer i=0;i<getFrag().size();i++)
42      { for (Role r:getFrag().get(i))
43        { System.out.println(r.getName());  }
44          System.out.println("");
45      }
46    }
47  }
```

Листинг 6. Файл RoleFragmentation.java

В соответствии с правилами наследования необходимо реализовать абстрактую функцию `belongsTo`, что и сделано в листинге 6 в строках 4–12. Чтобы проверить, действительно ли пользователь может быть авторизован на роли `(HashSet<Role> req)`, необходимо иметь функцию для проверки этого условия. Выглядит она следующим образом (листинг 7):

```
1  public static boolean fitFrag(HashSet<Role> req)
2  { HashSet<Role> ch;
3    RoleFragmentation rf=Config.getHandler().
      getRoleFragmentation();
4    HashSet<Integer> elements=new HashSet<Integer>();
5    Integer curblock;
6    for(Role r:req)
7    { ch=r.getAllChildren();
8      for (Role s:ch)
9      {curblock=rf.belongsTo(s);
10      System.out.println(curblock);
11      if ((curblock==-1)||(elements.contains(curblock)))
12       { return false; }
13       else
14       { elements.add(curblock); }
15     }
16       elements.clear();
17   }
18     return true;
19 }
```

Листинг 7. Файл RoleFragmentation.java

Поскольку ограничения статического и динамического взаимных исключений ролей практически идентичны для реализации, то пишутся функции проверки, можно ли пользователю `u` в список разрешенных ролей для авторизации (статическое ограничение) добавить роль `r` — строки 13–22 листинга 6, а также функция проверки, можно ли добавить сессии `s` в список ролей, на которые она авторизована, роль `r` (динамическое ограничение), что показано в строках 23–32 листинга 6. Так как статическое ограничение взаимного исключения ролей означает, что все роли, на которые может быть авторизован пользователь, находятся в разных блоках разбиения, то динамическое ограничение, в случае задания и статического, будет означать, что все запрашиваемые пользователем роли должны содержаться среди иерархически предшествующих разрешенным ролям. В случае отсутствия статического ограничения необходимо использовать функцию `CheckAbilityS`. Теперь необходимо выделить срез точек выполнения.

Во-первых, это набор `addingUserRole` из реализации количественных ограничений. Во-вторых, аналогичный набор для динамического ограничения (листинг 8):

```
1 public pointcut addingSessionRole(Session s, Role r):
2        call(void addAuthRole(Role)) && args(r);
```

Листинг 8. Фрагмент файла Test.aj

Предписания аналогичны предыдущим, с той лишь разницей, что в этот раз используются написанные функции проверок `checkAbility(User u, Role r)` и `checkAbilityS(Session s, Role r)`.

Реализация ограничений необходимости обладания

Основная идея реализации статических ограничений необходимости обладания состоит в том, чтобы создать для каждого объекта множество объектов, обладание которыми необходимо для обладания самим объектом.

Рассмотрим реализацию на примере ограничения необходимости обладания ролью.

Класс `Config` реализует шаблон *синглтон*, что означает, что экземпляр данного класса единственный. В процессе разбора файла конфигурации заполняется данное хранилище. В дальнейшем оно запрашивается у этого единственного экземпляра. Но так как это разбиение запрашивается еще в процессе разбора конфигурации, то запрос будем производить у самого обработчика. Для этого используется статическая функция `getHandler()` в классе `Config`.

Для эффективного хранения информации из конфигурационного файла используется словарь с эффективным поиском по ключу — `HashMap`. Например, в листинге 9 в строке 1 хранится (имя роли; максимальное количество пользователей на роль):

```
1 private HashMap<String,Integer> MaxUsersPerRoleContainer;
2 private HashMap<String,Integer> MaxRolesPerPrivilegeContainer;
3 private HashMap<String,Integer> MaxSessionsPerRoleContainer;
4 private HashMap<String,ArrayList<Role>>
     NeededRolesPerRoleContainer;
5 private HashMap<String,HashSet<Privilege>>
     NeededPrivsPerPrivContainer;
```

Листинг 9. Фрагмент файла Test.aj

Проверки реализованы функциями в аспекте (листинг 10):

```
1  public boolean checkAddingPrivilegeAbilityN(Role r,
2                                              Privilege p)
3  {System.out.println("checking privilege "+p.getName());
4   System.out.println(Config.getHandler().
      getNeededPrivsPerPrivilege());
5   if (Config.getHandler().getNeededPrivsPerPrivilege().
6      get(p.getName())==null)
7          return true;
8   return r.getPrivs().containsAll(Config.getHandler().
9      getNeededPrivsPerPrivilege().get(p.getName()));
10 }
11 public boolean checkAddingRoleAbilityN(User u, Role r)
12 {System.out.println("Checking role "+r.getName());
13  System.out.println(Config.getHandler().
```

```
14                getNeededRolesPerRole());
15   if (Config.getHandler().getNeededRolesPerRole().
16                get(r.getName())==null)
17      return true;
18   return u.getRoles().containsAll(Config.getHandler().
19             getNeededRolesPerRole().get(r.getName()));
20 }
21 public boolean checkAddingRoleToSessionAbilityN(Session s,
22                                                  Role r)
23 {if (s.getNeededRoles(r)==null) return true;
24   return s.getAuthRoles().containsAll(s.getNeededRoles(r));
25 }
```

Листинг 10. Фрагмент файла Test.aj

Срез точек выполнения для данного ограничения — `addingUserRole`. Предписание выглядит вызовом одной функции из листинга 10. Например (листинг 11):

```
1 try
2 {if (checkAddingRoleAbilityN(u, r)==false) throw new
     NeededRolesConstraintCheckException();
3 }
4 catch (NeededRolesConstraintCheckException ex)
5 { ex.printStackTrace();
6   System.out.println(ex.toString());
7 }
```

Листинг 11. Фрагмент файла Test.aj из before(Role r):addingUserRole(r)

7. Внедрение политики РРД в Apache Ftp Server

Для внедрения политики РРД в работу Ftp-сервера создадим пакет `RBAC-AJ`, который помещается в проект `ftpserver-core-1.0.x` и связывает его и пакет `RBAC`.

Для иллюстрации возможных ситуаций в работе сервера будем полагать, что у всех пользователей одна и та же виртуальная домашняя директория — '/'.

В главном файле `rbac.aj` располагаются, в том числе, предписания на выполнение в точках перехвата введённых пользователем команд проверки введённых данных на соответствие политике безопасности РРД.

В частности, необходим перехват команд `USER, PASS, CWD, MKD, RMD, STOU, RNFR, LIST`, а также всех команд, описанных в [13]. В дальнейшем предлагается сопоставить команды сервера и действия над объектами хранилища для определения необходимых прав доступа пользователя к данным объектам.

Создадим вспомогательный класс `SessionManager`, который будет хранить сопоставления уникального идентификатора сессии UUID и объекта сессии (строка 8 листинга 12), а также UUID и ролей, которые запрашивает пользователь (строка 10 листинга 12). Данный класс реализует шаблон *синглтон* — строки 11–13 листинга 12.

```
1 package rbac;
2 import java.util.HashMap;
3 import java.util.HashSet;
4 import java.util.UUID;
5 import org.apache.ftpserver.impl.FtpIoSession;
6 public class SessionManager {
```

```
 7 private static HashMap<UUID,my.Session> sessionsContainer=
 8  new HashMap<UUID,my.Session>();
 9 private static HashMap<UUID,HashSet<my.Role>>
10  sessionRolesContainer=new HashMap<UUID,HashSet<my.Role>>();
11 public static my.Session getInstance(FtpIoSession fs){
12  return sessionsContainer.get(fs.getSessionId());
13  }
14 ...
15 }
```

Листинг 12. Из проекта ftpserver-core-1.0.x (пакет src/java/main/rbac, файл SessionManager.java)

7.1. Инициализация РРД

Вследствие предоставления языком AspectJ возможностей по написанию предписаний типа `around`, имя пользователя и список ролей, на которые претендует пользователь, запрашиваются вместе с командой `USER` сервера, а затем в предписании извлекается список ролей, а серверу передаётся корректный запрос команды `USER`.

Формат команды `USER`: `USER имя_пользователя роль1[роль2 ... рольN]`.

Таким образом достигается совместимость с любыми ftp-клиентами, поскольку не вводятся дополнительные команды для протокола FTP.

Формировать сессию РРД будем в случае успешной аутентификации пользователя в рамках ftp-сервера, что проиллюстрировано в строках 4–10 листинга 13.

```
 1 void around(Command c,FtpIoSession s,FtpServerContext ctx,
     FtpRequest r):usercommand(c,s,ctx,r) && if(c instanceof
     PASS)
 2 { proceed(c,s,ctx,r);
 3   if (s.getUser()!=null){
 4   my.Session currentSession=new Session();
 5   currentSession.setUser(Config.getDefaultInstance()
 6   .getUserContainer().get(s.getUser().getName()));
 7   currentSession.addAuthRoles(SessionManager.getRoles(s));
 8   SessionManager.addSession(s, currentSession);
 9   SessionManager.delRoles(s);
10  }
11 }
```

Листинг 13. Перехват PASS

Перед стартом сервера необходимо разобрать файл конфигурации, что производится построением экземпляра класса `Config`: `Config.getDefaultInstance();`.

В соответствии с теоретической моделью РРД, так как отношение иерархии ролей не обязательно является деревом (и даже полурешёткой), то и в данной реализации у каждой роли могут быть одна или несколько родительских ролей, что влечет за собой несколько тегов `<parent>` у роли.

7.2. Формат файла конфигурации

В качестве приемлемого способа инициализации параметров политики РРД и гибкого конфигурирования выберем хранение конфигурации в файле формата XML [11]. В соответствии со стандартом формата XML все содержимое конфигурационного файла заключено в тег `<configuration>` и соответствующий закрывающий тег

</configuration>. У данного тега есть три атрибута, олицетворяющие собой настройки по умолчанию для количественных ограничений.

Структура файла конфигурации разбита на следующие блоки:

1) определение прав доступа;
2) разбиение прав доступа на блоки;
3) определение ролей;
4) разбиение множества ролей на блоки;
5) определение пользователей.

Определение права доступа осуществляется между открывающим и закрывающим тегами <privilegedef>; атрибут name задаёт имя права доступа, которое в дальнейшем может быть использовано для ссылки на него.

В соответствии с методикой реализации РРД у права доступа есть список регулярных выражений (описывается с помощью тега <targetsd>), описывающих набор объектов, к которому оно применяется, флаг инвертирования (атрибут inverted тега <targetsd>) и набор действий — список открывающих и закрывающих тегов <action>.

При определении права доступа можно установить количественное ограничение числа ролей на него — <maxRolesPerPrivilege>.

Для каждого права доступа могут быть заданы другие права доступа, на которые роль также должна быть авторизована. Они должны быть перечислены по одному в тегах <neededPriv> и </neededPriv>.

Разбиение множества прав доступа заключено между открывающим и закрывающим тегами privfragmentation, в которых каждый отдельный класс разбиения заключен между тегами <privblock> и </privblock>. Каждое право в блоке идентифицируется по его имени, заданном при определении, и заключается в теги <privpart> и </privpart>.

Определение роли осуществляется между открывающим и закрывающим тегами roledef. В это определение могут входить список прав доступа (в тегах <priv> и </priv>), а также родительские роли в тегах parent по одной, количественные ограничения внутри тегов maxUsersPerRole и maxSessionsPerRole и необходимости обладания ролью — neededRole.

Разбиение множества ролей на классы аналогично разбиению множества прав доступа, за исключением замены в именах тегов priv на role.

Определение пользователей происходит внутри тегов <userdef> и </userdef> и имеет атрибут name, который и вводится пользователем в команде USER и есть в файле пользователей сервера. Далее можно определить роли, на которые может быть авторизован пользователь — <role>rolename</role>.

7.3. Функционирование политики безопасности

Рассмотрим общую схему функционирования политики безопасности.

Пользователь обращается на FTP-сервер с запросом предоставить ему доступ к хранилищу файлов под именем — первым аргументом команды USER — с ролями, список которых он указывает сразу после имени пользователя в команде USER.

В момент поступления команды от пользователя задействуется функциональность политики безопасности путём передачи в специальном виде информации о запросе пользователя, и затем анализируется ответ от модуля политики. Если ответ на запрос пользователя положительный, то запрашиваемый доступ предоставляется, иначе — запрещается.

Для успешного функционирования вышеописанной процедуры проверки на возможность предоставления доступа создаётся экземпляр класса `Query` и вызвается метод `isGranted()`. Эта последовательность действий производится в предписаниях-перехватчиках команд пользователя, реализация которых похожа друг на друга структурой:

— сохранить строку — аргумент команды, если таковой имеется;
— преобразовать полученную строку к полному пути относительно виртуальной директории сервера;
— интерпретировать название команды, полученной от пользователя, в последовательность требуемых для ее успешного выполнения прав доступа, которым должна обладать хотя бы одна роль, на которую авторизован данный пользователь;
— проверить наличие у пользователя необходимых привилегий и вынести решение — продолжать выполнение команды сервером или сгенерировать соответствующее исключение.

Для начала необходимо определить срез точек выполнения — моменты перехвата вводимых команд (листинг 14). Заметим, что вызовы команд однотипны — в интерфейсе под именем названия команды существует единственная функция — `void execute(...)`.

```
1  pointcut usercommand(Command c,FtpIoSession s,
     FtpServerContext ctx,FtpRequest r):call(void execute
2    (FtpIoSession,FtpServerContext,FtpRequest))
3     && target(c) && !adviceexecution() && args(s,ctx,r);
```

Листинг 14. Фрагмент файла Rbac.aj — срез моментов перехвата команд

Рассмотрим по порядку возможную реализацию перехватчиков.

Работа с аргументом команды

У интерфейса `FtpRequest` существуют функции по обработке аргументов запроса пользователя. Основные из них перечислены в листинге 15.

```
1  /* Get the client request string.
2  * @return The full request line, e.g. "MKDIR newdir"
3  */
4  String getRequestLine();
5  /* Returns the ftp request command.
6  * @return The command part of the request line, e.g. "MKDIR"
7  */
8  String getCommand();
9  /* Get the ftp request argument.
10 * @return The argument part of the request line, e.g. "newdir"
11 */
12 String getArgument();
13 /* Check if request contains an argument
14 * @return true if an argument is available
15 */
16 boolean hasArgument();
```

Листинг 15. Из пакета org.apache.ftpserver.ftplet — FtpRequest.java

С их помощью требуемая работа очевидна.

Преобразование полученной строки к полному пути относительно виртуальной директории сервера

Для решения этой задачи необходимо воспользоваться строчкой 6 листинга 16, которая позволяет получить полный путь текущей рабочей директории в сессии пользователя. Сделав некоторые поправки на разницу между именем файла и именем директории, получим искомый результат на листинге 16.

```
1  public String getFullName(FtpIoSession s,String name)
2  { String nD=name;
3    if ((nD.length()>0)&&(nD.charAt(0)!='/'))
4    { String wd="";
5      try{wd=s.getFileSystemView().getWorkingDirectory().
6          getAbsolutePath();
7        }
8      catch(FtpException ex)
9      { ex.printStackTrace(); }
10     if (wd.charAt(wd.length()-1)=='/')
11     { nD=wd+name;   }
12     else
13     { nD=wd+"/"+name; }
14   }
15   return nD;
16 }
```

Листинг 16. Фрагмент файла Rbac.aj- getFullName

В последующем потребуется функция определения родительской директории объекта (листинг 17).

```
1  public static String upper(String name)
2    { Integer index;
3      index=name.lastIndexOf("/");
4      if ((index==-1)||(index==0)) return "/";
5      return name.substring(0,index);
6    }
```

Листинг 17. Фрагмент файла Rbac.aj-upper

Интерпретировать название команды FTP-сервера в последовательность требуемых прав доступа

Для решения данной задачи необходимы сведения о выполняемых командой действиях. Соответственно этим сведениям определяются следующие наборы привилегий, где объект — это аргумент команды:

- APPE — в случае существования объекта право записи в объект, иначе — право записи в родительскую директорию;
- CWD — в случае присутствия аргумента команды право чтения объекта, иначе — право чтения корневой виртуальной директории;
- DELE — право записи в родительскую директорию;
- LIST — право чтения объекта;
- MKD — право записи в родительскую директорию объекта;
- RETR — право чтения объекта;
- RMD — право записи в родительскую директорию объекта;

— RNFR — право записи в родительскую директорию объекта;
— RNTO — право записи в родительскую директорию объекта;
— STOR — в случае существования объекта право записи в объект, иначе — право записи в родительскую директорию;
— STOU — право записи в родительскую директорию объекта.

Проверка требуемых прав доступа и вынесение решения

Проиллюстрируем все шаги на примере команды RETR (листинг 18).

```
1  void around(Command c,FtpIoSession s,FtpServerContext ctx,
       FtpRequest r):
2            usercommand(c,s,ctx,r) && if(c instanceof RETR)
3  { String nD = r.getArgument();
4    nD=getFullName(s, nD);
5    Query q=new Query();
6    FtpTargetDescriptor ftd=new FtpTargetDescriptor(nD);
7    HashSet<Action> hs=new HashSet<Action>();
8    q.addColumn(ftd, hs);
9    q.setSession(SessionManager.getInstance(s));
10   if (q.isGranted())
11           proceed(c,s,ctx,r);
12   else
13           s.write(LocalizedFtpReply.translate(s, r, ctx,
14           FtpReply.REPLY_550_REQUESTED_ACTION_NOT_TAKEN, "RETR",
15           nD));
16 }
```

Листинг 18. Фрагмент файла Rbac.aj- around RETR

В листинге 18 используется написанная ранее в классе `Query` функция `isGranted`, которая и принимает требумое решение. В строках 6 и 9 происходит конструирование запроса с заполнением объекта, к которому требуется доступ и действие чтения, что соответствует команде RETR.

В дальнейшем, если решение положительное (строка 11), то выполнение команды продолжится, иначе (строка 13) — генерируется ответ сервера с кодом 550. При использовании графических ftp-клиентов пользователю будет выдано развернутое сообщение об ошибке.

Заключение

Аспектно-ориентированное программирование — одна из важнейших парадигм программрования, которую можно успешно использовать для написания и внедрения в компьютерные системы политик безопасности. Она вводит требуемый для этого уровень модульности и гибкости внедрения. При помощи заключенных в ней механизмов существует возможность взаимодействовать с пользователем, встраивать политики безопасности и другие модули без модификации исходных кодов компьютерной системы.

В статье сформулированы основные требования и рекомендации к реализации и внедрению политик безопасности в защищаемые КС методом АОП. По этим рекомендациям осуществлено внедрение базовой политики ролевого разграничения доступа в FTP-сервер Apache, не обладающий механизмом разграничения доступа пользователей к файлам друг друга. Технология этого внедрения продемонстрирована в разд. 6

и 7, где представлены соответственно реализация основной функциональности РРД — пакет RBAC, содержащий реализации на языке Java основных объектов политики РРД, таких, как Пользователь (User.java), Роль (Role.java) и т. д., и реализация пакета RBAC-AJ — соединительного модуля между сервером и пакетом RBAC.

ЛИТЕРАТУРА

1. *Таненбаум Э.* Современные операционные системы. 2-е изд. СПб.: Питер, 2009. 314 с.
2. *Dijkstra E. W.* Selected Writings on Computing: A Personal Perspective. London; New York: Springer Verlag, 1982. С. 60–66.
3. *Elrad T., Aksit M. M., Kiczales G., et al.* Discussing Aspects of AOP // Comm. ACM. 2001. V. 44. No. 10. P. 33–38.
4. `www.ddj.com/architect/184414752` — Through the Looking Glass. 2001.
5. *Стефанцов Д. А.* Реализация политик безопасности в компьютерных системах с помощью аспектно-ориентированного программирования // Прикладная дискретная математика. 2008. № 1. С. 94–100.
6. *Девянин П. Н.* Модели безопасности компьютерных систем. М.: Академия, 2005. 144 с.
7. `http://java.sun.com/docs/books/jls/` — The Java books. 2010.
8. `http://eclipse.org/aspectj` — AspectJ Home Page. 2010.
9. `http://aophp.net` — Aspect Oriented PHP. 2010.
10. `http://www.uml.org` — Unified Modelling Language. 2010.
11. `http://tools.ietf.org/html/rfc3076` — RFC 3076 - Canonical Xml Version 1.0. 2010.
12. `http://mina.apache.org/ftpserver` — Apache Ftp Server. 2010.
13. `http://tools.ietf.org/html/rfc959` — RFC 959 - Ftp. 2010.