Обеспечение безопасности веб-приложений.

Обеспечение безопасности приложения заключается в выполнении по крайней мере двух функций: аутентификации и авторизации пользователей приложения. Аутентификация – это проверка того, что пользователь является именно тем, за кого себя выдает. Как правило, пользователь аутентифицируется с помощью имени пользователя и пароля. После успешной аутентификации пользователь должен быть авторизован. Авторизация – это процесс определения прав доступа пользователя к конкретному ресурсу. Как правило, авторизация выполняется с помощью списка контроля доступа, в котором перечислены пользователи и операции, которые они могут выполнять с теми или иными ресурсами. Например, пользователь системы «клиент-банк» успешно прошел аутентификацию, введя имя пользователя и пароль, однако он может работать только со своим банковским счетом, что обеспечивается средствами авторизации.

Для аутентификации пользователей на конкретном сервере приложений существуют учетные записи пользователей (например, управление пользователями сервера приложений SJSAS 9 ведется в консоли администрирования в ветке Configuration / Security / Realms). Пользователи могут объединяться в *группы* – категории пользователей, выделяемые по тем или иным характеристикам, например, по должности или профилю деятельности.

Домен безопасности (realm) представляет собой некоторый контекст, в рамках которого сервер приложений соблюдает установленную политику безопасности. На практике домен безопасности представляет собой базу данных пользователей и групп. По умолчанию все приложения на сервере используют единый домен безопасности, но для каждого приложения можно определить собственный домен безопасности.

Роль (role) является механизмом авторизации и определяет права доступа пользователя, необходимые ему для выполнения круга его задач. Например, в приложении управления кадрами все сотрудники могут иметь доступ к телефонным номерам и адресам электронной почты других сотрудников, но доступ к сведениям о зарплате должен быть только у руководителей. Таким образом, для этого приложения можно выделить две роли: `employee` (сотрудник) и `manager` (руководитель), причем просматривать сведения о зарплате могут только пользователи, авторизованные как руководители (то есть находящиеся в роли `manager`).

Следует отличать роль от группы пользователей. Во-первых, роль соответствует определенной функции приложения, а группа объединяет пользователей по некоторым признакам лишь для удобства администрирования (например, для назначения на роль в приложении сразу всех пользователей группы). Во-вторых, роли определяются в установочном дескрипторе приложения и не зависят от конкретного сервера приложений, тогда как группы пользователей всегда определяются в конкретном сервере приложений, точнее - в конкретном домене безопасности (realm).

На этапе установки веб-приложения необходимо выполнить сопоставление абстрактных ролей, в терминах которых было выполнено распределение прав доступа к компонентам веб-приложения, с конкретными пользователями и группами конкретного сервера приложений.

Механизмы аутентификации

Сервер приложений Java EE поддерживает четыре механизма аутентификации веб-клиентов: HTTP Basic, HTTP Digest, Form-based и Certificate. Определение используемого в веб-приложении механизма аутентификации выполняется с помощью элемента `login-config` установочного дескриптора.

При использовании механизма HTTP Basic на запрос к защищенному ресурсу сервер отвечает сообщением следующего вида:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="домен_безопасности"
```

Получив подобное сообщение, браузер выводит собственное диалоговое окно, в котором пользователь вводит имя и пароль. Введенная информация отправляется браузером к серверу без шифрования, открытым текстом. Сервер выполняет проверку и в случае успеха возвращает пользователю запрошенную информацию (передает запрос на обработку компоненту, который был изначально запрошен, и тот генерирует ответ).

Механизм HTTP Digest аналогичен механизму HTTP Basic, с тем лишь отличием, что введенные пользователем имя и пароль передаются не в открытом виде, а как хэш-значение (т.н. дайджест).

Механизм Form-based (с использованием формы) позволяет разработчику предоставить собственную веб-форму, в которую пользователь вводит имя и пароль. Страница, содержащая описание формы, указывается в установочном дескрипторе. При этом к форме предъявляются следующие требования:

1) форма должна передавать данные методом POST;

2) атрибут `action` формы должен иметь значение `j_security_check`;

3) элемент ввода имени пользователя должен иметь имя `j_username`;

4) элемент ввода пароля должен иметь имя `j_password`.

Недостаток данного механизма такой же, что и у HTTP Basic, то есть имя и пароль передаются без шифрования, открытым текстом.

Механизм Certificate предусматривает взаимную аутентификацию (mutual authentication) сервера и клиента на основе сертификатов стандарта X.509. Этот механизм является частью протокола SSL и обеспечивает также шифрование всего трафика.

Шифрование выполняется на основе алгоритма с открытым ключом. При генерации сертификата он содержит два ключа: открытый и закрытый. Открытый ключ используется для шифрования информации, передаваемой от других пользователей этому пользователю, который для расшифровки использует закрытый ключ. Сертификат, фактически, – это оболочка над парой ключей закрытый/открытый (используется для расшифровки) и электронной подписи, или только над открытым ключом (используется для шифрования).

Сертификаты распространяются в виде файлов различных форматов (`pfx` – для содержащих закрытый ключ; `cer`, `crt` – для содержащих только открытый ключ). Программы, работающие с сертификатами, используют не файлы, а так называемые хранилища сертификатов, которые могут быть самыми разными. Например, MS Internet Explorer и MS Outlook Express работают с хранилищем, содержащимся в реестре Windows. Виртуальные машины Java как физическое хранилище используют файл формата `JKS` (Java Key Store), для работы с которыми предусмотрен инструмент `keytool`. Он входит в состав дистрибутива Java SE, а также сервера приложений фирмы Sun. Этот инструмент позволяет создать новое хранилище сертификатов, сгенерировать самоподписанный сертификат, сгенерировать запрос на получение сертификата к центру сертификации, импортировать сертификат из файла в хранилище, экспортировать сертификат из хранилища в файл, просмотреть содержимое хранилища и отдельного сертификата из хранилища или из файла, удалить сертификат из хранилища.

В большинстве практических случаев пользователи получают сертификаты от центров сертификации (Certificate Authority, CA), которые, как говорят, подписывают пользовательский сертификат своим сертификатом. Такая подпись – основа для доверия пользовательскому сертификату. Сертификаты центров сертификации и сертификаты пользователей хранятся отдельно. В Java первые хранятся в хранилище `cacerts`, а вторые – в хранилище `keystore`. Они расположены в подкаталоге `lib\security` каталога установки JRE и сервера приложений.

Ограничение доступа к компонентам

Платформа Java EE предусматривает два подхода к организации ограничений доступа к компонентам (как веб-, так и EJB-компонентам). Первый подход – декларативный (declarative security) – заключается в описании правил ограничения доступа в установочном дескрипторе. Второй подход – программный (programmatic security) – предусматривает, что проверка прав доступа к компоненту выполняется самим компонентом. Часто данные подходы комбинируются – ограничение обращений к компоненту выполняется декларативно, а функционирование компонента зависит от роли пользователя, которая проверяется программно. Например, для вывода списка сотрудников (см. выше) может использоваться одна и та же JSP-страница, доступ к которой разрешен только сотрудникам организации, но состав отображаемой на ней информации зависит от роли пользователя (так, зарплата сотрудников отображается только для пользователей с ролью `manager`).

Декларативное ограничение доступа к различным компонентам и статическому содержимому веб-приложения выполняется на основе ограничений безопасности (security constraint), задаваемых элементами `security-constraint` установочного дескриптора.

Действие ограничения безопасности распространяется на одну или несколько коллекций ресурсов (web resource collection). Принадлежность компонента или файла веб-приложения к коллекции ресурсов указывается с помощью шаблона URL (например, `*.jsp` для всех JSP-страниц). Также для коллекции устанавливаются методы передачи HTTP-запросов (GET, POST и т.д.), при использовании которых данное ограничение безопасности будет иметь силу.

Набор ролей, которые имеют доступ к заданным ресурсам, указывается в элементе `auth-constraint` ограничения безопасности. Если элемент `auth-constraint` пустой, то доступ к соответствующим ресурсам запрещен. Если же данный элемент отсутствует, то доступ пользователя к указанным ресурсам возможен без аутентификации. Роли берутся из числа определенных для веб-приложения (задаются элементами `security-role` в установочном дескрипторе).

В ограничении безопасности можно установить следующие требования к способу передачи запросов (транспорту): обеспечение целостности данных и обеспечение конфиденциальности.

Таким образом, ограничение безопасности устанавливает, кто (набор ролей), к чему (коллекции ресурсов) и как (транспорт) может получить доступ.

Механизмы декларативного ограничения доступа применяются в случае, когда запрос поступил непосредственно от клиента, и не применяются при перенаправлении запроса методами `forward()` и `include()` интерфейса `javax.sevlet.RequestDispatcher`.

Пример ограничений безопасности:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>JSP pages not available </web-resource-name>
        <url-pattern>*.jsp</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint/>
</security-constraint>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Administrator pages</web-resource-name>
        <url-pattern>/admin/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ADMIN</role-name>
```

```
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Staff pages</web-resource-name>
        <url-pattern>/staff/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>STAFF</role-name>
        <role-name>ADMIN</role-name>
    </auth-constraint>
</security-constraint>
```

Приведенные ограничения безопасности для ИС учебного заведения устанавливают следующие правила доступа:

1) Доступ ко всем JSP-страницам напрямую запрещен – все запросы обрабатываются сервлетами.
2) Обращение к разделу администрирования (ресурсам `/admin/*`) методами GET и POST разрешено только администраторам (роль `ADMIN`) по защищенному соединению.
3) Обращение к разделу с информацией для преподавателей (ресурсам `/staff/*`) методами GET и POST разрешено преподавателям (роль STAFF) и администраторам.

Применяемое ограничение доступа определяется по тем же правилам, по которым выполняется сопоставление запроса обрабатывающему его веб-компоненту. Если в приведенном примере среди ресурсов, заданных шаблонами `/admin/*` и `/staff/*` встретятся JSP-страницы, то они будут обработаны согласно правилам 2 и 3, соответственно.

Для реализации программного ограничения доступа в веб-компонентах в интерфейсе `HttpServletRequest` предусмотрены следующие методы:

1) `getRemoteUser()` – возвращает имя пользователя, под которым клиент, пройдя аутентификацию, обратился к компоненту;
2) `getUserPrincipal()` – возвращает объект типа `java.security.Principal` с информацией о пользователе, обратившемся к компоненту;
3) `isUserInRole(String roleName)` – позволяет проверить, обладает ли обратившийся к компоненту пользователь ролью `roleName`.

Пример программного ограничения доступа в рамках JSP-страницы:

```
<c:if test='<%= request.isUserInRole("ADMIN") %>'>
    <!-- Вывести информацию для администратора -->
</c:if>
```

Конфигурирование домена безопасности

На практике часто нужно подключить веб-сервер к уже существующему механизму аутентификации или базе пользователей. Также часто встречаются случаи, когда веб-приложение самостоятельно осуществляет управление пользователями (регистрация, изменение, удаление), но аутентификацию выполняет средствами веб-сервера.

К сожалению, в спецификации сервлетов не определен интерфейс между веб-сервером и реализацией домена безопасности (базой данной пользователей и групп). Поэтому каждый сервер приложений предоставляет собственные средства для конфигурирования доменов безопасности. Рассмотрим на примере серверов приложений

Tomcat 6 и SJSAS 9 типичные источники аутентификационной информации. (В табл. имена классов, реализующих домен безопасности даны относительно пакетов `org.apache.catalina.realm` для Tomcat 6 и `com.sun.enterprise.security.auth.realm` для SJSAS 9).

Источник	Tomcat 6	SJSAS 9
1. Файл с данными о пользователях	`MemoryRealm` (используется по умолчанию), список пользователей хранится в XML-файле, по умолчанию – `conf/tomcat-users.xml`	`file.FileRealm`, определены домены `file` (используется по умолчанию) и `admin-realm`, список пользователей хранится в текстовом файле
2. Реляционная база данных	`JDBCRealm` (подключение через `DriverManager`), `DataSourceRealm` (подключение через источник данных)	`jdbc.JDBCRealm` (подключение через источник данных)
3. LDAP-каталог	`JNDIRealm`	`ldap.LDAPRealm`
4. Учетные записи пользователей операционной системы	–	`solaris.SolarisRealm`, определен домен `solaris`
5. Хранилище сертификатов	–	`certificate.CertificateRealm`, определен домен `certificate`
6. Модуль JAAS (Java Authentication & Authorization Service)	`JAASRealm`	–

Разработчик имеет возможность подключить собственную реализацию домена безопасности, для чего ему нужно в Tomcat реализовать интерфейс `org.apache.catalina.Realm`, а в SJSAS 9 – расширить класс `com.sun.enterprise.security.auth.realm.Realm`.

Если веб-приложение, самостоятельно управляющее учетными записями пользователей, должно быть переносимым (то есть допускать установку на различные серверы приложений), то источником аутентификационной информации может быть реляционная база данных либо LDAP-каталог. Именно эти источники обеспечивают переносимые механизмы добавления, изменения и удаления учетных записей пользователей (с помощью интерфейсов JDBC и JNDI, соответственно). Рассмотрим подробнее первый вариант – с использованием реляционной базы данных.

В базе данных должны быть таблицы, содержащие сведения о пользователях и группах. В таблице пользователей должно быть не менее двух столбцов: для хранения имени пользователя и пароля, соответственно. Пароль может храниться как в прямом значении, так и в виде своего хэша (digested password). В таблице групп должны быть столбцы для хранения имени пользователя и названия группы. При конфигурировании домена безопасности необходимо указать в свойствах домена названия соответствующих таблиц и столбцов. Пример скрипта для создания базы данных пользователей и групп:

```
create table usertable(
    userid varchar(10) not null,
    password varchar(32) not null,
    primary key(userid)
);
create table grouptable(
    userid varchar(10) not null references usertable(userid),
    groupid varchar(20) not null,
```

```
    primary key(userid, groupid)
);
```

Веб-сервер Tomcat не поддерживает концепцию групп пользователей и отображение пользователей и групп на роли, определенные в веб-приложении. Поэтому база данных пользователей и групп для Tomcat на самом деле содержит таблицы пользователей и их ролей, но структура таблиц не отличается от описанной выше.

Очевидно, что база данных пользователей и групп может состоять и из одной таблицы, при этом каждый пользователь может быть членом лишь одной группы (в Tomcat – выполнять единственную роль).

Работа с пользователями в веб-приложениях

Во многих приложениях требуется не только аутентифицировать и авторизовать пользователей, но также использовать подробную информацию о пользователе при выполнении бизнес-логики или формировании представления (веб-интерфейса). Данная подробная информация, как правило, хранится в базе данных, а в приложении представляется классом(-ами) модели предметной области. Очевидно, что можно совместить учетную запись (имя, пароль) и подробную информацию о пользователе в одной таблице, так как для связи пользователя с учетной записью все равно придется вводить поле `username`.

При многократном использовании объект пользователя можно хранить в контексте сессии, так как сессия представляет сеанс работы конкретного пользователя, и в рамках сессии нельзя сменить связанного с ней пользователя на другого. Для гарантированного помещения объекта пользователя в контекст сессии можно поместить в начало метода `doFilter()` фильтра либо в начало метода обработки запроса в сервлет-контроллере код, аналогичный следующему:

```
String username = request.getRemoteUser();
if (username != null) {
    User user = UserDAO.getUser(username);
    request.getSession().setAttribute(Constants.USER, user);
}
```

Использование фильтра является предпочтительным при наличии нескольких сервлет-контроллеров или при возможности прямого обращения пользователя к защищенным JSP-страницам.

Нередко отдельным ролям пользователей соответствуют различные классы приложения. Например, в ИС медицинского назначения могут быть следующие роли: пациент, врач, администратор. Каждой из ролей соответствует различный объем информации и класс в приложении. Так, для врача важна специализация и квалификация, а для пациента – физиологические характеристики и история болезни. В этом случае можно обойтись единственной таблицей для хранения пользователей и групп (ролей), так как набор ролей является непересекающимся. Возможность совмещения таблицы учетных записей пользователей и таблиц(-ы) с подробной информацией о пользователях зависит от того, находятся ли все классы пользователей в одной иерархии наследования, а также от используемой стратегии отображения наследования классов на структуру таблиц.