DevColibri

Главная

Как реализовать Security в Java EE? Часть 1

Первый раз когда я услышал о безопасности в EE приложениях, это вызывало у меня страх. Так как не понятно было вовсе что и как работает. литературы, но все разбросано по просторам интернета. Сегодня, для тех кто испытывает то что и я когда-то, предоставляю урок, где все будет воедино.

Скачать

Скорее всего, если вы уже думали о том, чтобы использовать безопасность в своих приложениях, то вы уже знаете для каких целей она вам ну

В частности — закрыть доступ для каких-то операций, которые может делать только конкретная **группа** людей, в которых есть **право** на выполн Можно конечно не усложнять себе жизнь и не добавлять никакой защиты, но что тогда будет?

Любой человек может зайти на наш сайт и принести ему вред будь то случайно или умеренно.

Представьте что вы разрабатываете банковскую систему. Вокруг вашего приложения крутятся деньги и не малые. Думаю что всегда найдутся те которые захотят эти деньги получить. И если нет системы защиты и надеяться только на честность людей, то проблем не схлопотать.

Шаг 1. Типы реализации

У нас есть несколько вариантов безопасности, которую мы можем использовать. Начнем с того, что объяснит нам всю суть.

Ручная реализация

Суть состоит в том, что мы берем на себя ответственность за реализацию безопасности в нашем приложении, сами придумываем умные алгорит шифрования, на каждой страничке делаем проверку на правомерность пользователя выполнять действия.

И если что-то пойдет не так, мы допустим ошибку в своем алгоритме, то приложение стает под угрозой перед злоумышленниками.

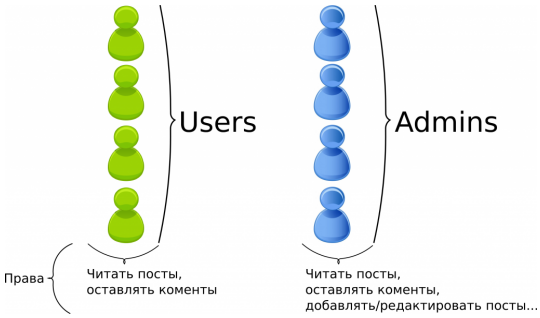
Реализация с помощью сервера приложений

Естественно безопасность разрабатывается очень часто и есть уже конкретные шаблонны, как именно ее организовывать.

Чтобы уйти от написания безопасности вручную, уменьшить вероятность взлома системы и ускорить разработку каждый может воспользоваться реализацией безопасности, которую предлагает нам сервер приложений. Это хорошо продуманная и гибкая система.

Шаг 2. Основные понятия

Мы будем заниматься именно безопасностью которую предоставляет нам сервер приложений. Для начала, стоит познакомиться с некоторыми по



Аутентификация — проверка на существование человека(зарегистрированного) в нашем приложении. Является ли он тем, за кого себя выдает?

Авторизация — проверка прав аутентифицированного пользователя выполнять конкретные действия.

User — пользователь, информация о человеке, которая храниться у нас в **базе** или **файле.**

Group — группа пользователей, которая имеет одинаковые характеристики и права доступа.

Security Realm — специальная область отвечающая за аутентификацию пользователя и хранит ее настройки.

Role — определяет уровень доступа. Определяет какие действия может выполнять пользователь или группа.

Шаг 3. Настройка безопасности

Думаю для начала этих знаний будет достаточно, с остальной частью мы уже познакомимся на практике. **Постановка задачи** Пусть у нас будет г которое имеет несколько страниц, которые будут доступны конкретной группе пользователей, таких как менеджеры или администраторы. Данн будет содержать в себе «секретную» информацию. Начинаем с того что просто создаем проект с основой для творчества.

Мы создали основную структуру. Пока что нету ничего кроме статического **html.** Сейчас все ссылки доступны с приложения и можно дойти к лю файлу. Есть специально создано 2 ресурса: **public, secured.** Первым делом, ограничим доступ ко всему что находиться внутри **secured folder.** Дл создаем внутри **webapp** директорию **WEB-INF,** и прописываем дескриптор развертывания **web.xml.**

Пример файла **web.xml:**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>secured</web-resource-name>
            <url-pattern>/secured/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>MANAGER</role-name>
        </auth-constraint>
    </security-constraint>

    <security-role>
        <role-name>MANAGER</role-name>
    </security-role>

</web-app>
```

security-constraint — блок ограничений безопасности(настраиваем права доступа и метод(GET,POST..)).

web-resource-collection — ресурсы, к которым стоит закрыть доступ, ввести ограничение.

web-resource-name — название ресурса.

url-pattern — запрос, к которому будет применена безопасность. ***** — указывает что все что идет после также попадает под настройки безопасн **/security/*** — будет закрывать доступ для всех адресов вида: **/security/1.html, /security/posts/1/.**

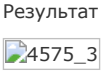
http-method — метод к которому применим фильтр безопасности.

auth-constraint — выставляем роли доступа.

security-role — прописываем роль.

role-name — указываем роль.

Результат работы по рут:



Результат, при доступе к закрытой страничке: **Forbidden**

Теперь у нас есть закрытый доступ к ресурсам, которые находятся по адресу **secured/**, доступ к содержимому имеет только менеджер. Но как ук мы являемся менеджером?

Шаг 4. Аутентификация

Добавим теперь нашему приложению возможность аутентификации, которая есть у нас по умолчанию. Методы аутентификации:

1. BASIC — Здесь используется стандартная форма ввода данных для аутентификации.

```
<login-config>
        <auth-method>BASIC</auth-method>
</login-config>
```

При доступе к закрытым ресурсам вы увидите окно, которое попросит вас ввести свои данные.

2.FORM — Здесь используем свою **html** форму. Делаем настройки в **web.xml:**

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/login.html</form-login-page>
        <form-error-page>/error.html</form-error-page>
    </form-login-config>
</login-config>
```

login-config — конфигурация аутентификации.

auth-method — каким методом проводить аутентификацию. Если выбираем **FORM,** то нужно ввести дополнительную информацию:

form-login-config — дополнительная информация при использовании своей формы

form-login-page — **url** логина

form-error-page — **url** ошибки логина.

Создаем нашу форму:

```
<form action="j_security_check" method="post">
    Input for username:<br>
    <input name="j_username" type="text"><br>
    Input for password:<br>
    <input name="j_password" type="password"><br>
    <input type="submit" value="Авторизироваться">
</form>
```

Соглашения, которых стоит придерживаться при использовании безопасности сервера приложений.

j_security_check — **action** при нажатии на **submit**

j_username — поле имя пользователя

j_password — поле пароля

Форма(без стилей и ничего лишнего):

3.DIGEST — цифровая аутентификация

4.CLIENT-CERT — аутентификация с помощью клиентского сертификата.

Последние два мы не будем рассматривать в нашем примере.

Шаг 5. Настройка сервера приложений

Теперь, осталось настроить наш сервер приложений и связать его с нашим приложением. Для связки, в зависимости от сервера, в приложении используется специальный файл, который имеет название: ***-web.xml.**

JBoss, WildFly: jboss-web.xml

GlassFish: sun-web.xml

Создаем нужный нам файл в директории **WEB-INF.**

В данном случае мы будем использовать **properties** файлы у нас на сервере, что будут хранить данные о пользователях.

Пример файла **jboss-web.xml:**

```
1 <jboss-web>
2  <security-domain>java:/jaas/other</security-domain>
3 </jboss-web>
```

Теперь добавим пользователя, у которого будет право на просмотр скрытой информации. Для этого используем консольное приложение которое находиться в директории **bin/add-user.sh** (Linux), **bin/add-user.bat** (Windows).

После чего, если у Вас был запущен сервер — перезапускаем и пробуем зайти на защищенные странички.

И после корректного логина, мы попадаем на защищенную страницу.

Вот так просто вы добавили безопасность к своему приложению. Такой подход используется не часто, и больше всего распространен подход с использованием БД.

Шаг 6. Использование безопасности в сервлетах

Здесь очень кратко покажу базовые настройки для сервлетов. Для этого, создадим 2 сервлета в нашем проекте.

SecuredServlet1 — если посмотреть на адресс, защищен с помощью настроек в **web.xml**

```
@WebServlet(urlPatterns = "/secured/servlet1")
public class SecuredServlet1 extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                                    throws ServletException, IOException {
        resp
                .getWriter()
                .write("<h1>Secured Servlet1</h1>");
    }
}
```

SecuredServlet2 — имеет свои настройки безопасности и не попадает под настройки дескриптора.

```
@WebServlet(urlPatterns = "/servlet2")
@ServletSecurity(httpMethodConstraints = {
        @HttpMethodConstraint(value = "GET", rolesAllowed = "MANAGER"),
        @HttpMethodConstraint(value = "POST", rolesAllowed = "MANAGER")
})
public class SecuredServlet2 extends HttpServlet{
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                                                    throws ServletException, IOException {
        resp
                .getWriter()
                .write("<h1>Secured Servlet2</h1>");
    }
}
```

@ServletSecurity — определяет настройки безопасности

@HttpMethodConstraint — ограничения для каждого метода доступа

value — http method (GET,POST….)

rolesAllowed — роль, которая может получить доступ.

Есть и остальные параметры, их мы рассмотрим в след. уроках, с практическим примером.

Результат доступа к сервлету 2.

После авторизации вы увидите следующее:

Урок создан: 30 июня 2014 | Просмотров: 29095 | Автор: Олег Криль | Правила перепечатки

Добавить комментарий

Комментарии:

Антон 07 июля

А как сделать кнопку «Выход»?
Ответить

Олег Криль 07 июля

Просто создайте сервлет, который будет чистить данные аутентификации.
Я напишу об этом в след. уроке.
Ответить

theNatd 21 июля

Всегда пугала именно эта часть и как то приходилось её обходить, делал в jax-rs проверку токена в интерцепторе, роли хранил в базе, соответственно в интерцепторе проверял роли у пользователя.

Расскажите лучше этот момент, как лучше сделать если роли хранятся в базе.
Ответить