基于 J2EE 的 Web 应用系统身份认证技术研究①

Study of J2EE – based Web Application System Identity Authentication Technology

郑秀琴　诸葛毅　诸葛理绣　（衢州学院　浙江衢州 324000）

摘　要： 基于 J2EE 平台的 Web 应用系统常用的验证主体身份的方法是基于主体了解的秘密的，常用的认证技术主要有基于 X.509 证书的认证和基于用户名和口令的认证。本文较系统地阐述了 Servlet 容器和 Web 服务器用户认证的原理与实现，并剖析了应用程序本身运用 Session 和 Cookie 机制进行用户认证的原理。

关键词： 身份认证 WEB J2EE Servlet Session Cookie

1　引言

用户认证是保护网络系统资源的第一道防线，它控制着所有登录并检查访问用户的合法性，其目标是仅让合法用户以合法的权限访问网络系统的资源。在计算机网络中，通常采用三种方法验证主体身份。一是只有该主体了解的秘密，如口令、密钥；二是主体携带的物品，如智能卡；三是只有该主体具有的独一无二的特征或能力，如指纹、声音、视网膜血管分布图或签字等。

在 WEB 应用系统中通常采用第一种方法。基于 JAVA 平台的 Web 应用系统常用的用户认证技术主要有基于证书的认证和基于用户名、口令的认证。本文主要阐述了 Servlet 容器和 Web 服务器用户认证的原理与实现，并剖析了应用程序本身运用 Session 和 Cookie 进行用户认证的原理。

2　Web 服务器上的证书认证和用户名、口令认证

Apache 是目前流行的 Web 服务器，下面我们以 Apache 服务器为例剖析证书认证和用户名、口令认证的原理，并给出实现认证的步骤。

2.1　Apache 服务器的证书认证

2.1.1　X.509 证书认证原理

X.509 证书存储了一个实体的身份，并通过使用该实体的私钥加密某段数据以数字方式签署它。已签署消息的接收者必须使用这个实体的公钥解密签名。只有当签名有效时才能解密成功。只有实体本身知道私钥，如果解密成功则表示签名是有效的，实体得到认证。按照双方交换信息的不同，可以分为一路单向认证、二路双向认证和三路双向认证三种不同的方案。

一路单向认证可表示为：A – >B：A || Sa (Ta || Ra || B || Eb (Ya))

二路双向认证可表示为：A – >B：A || Sa (Ta || Ra || B || Eb (Ya))

B – >A：Sb (Tb || Rb || A || Ra || Ea (Yb))

三路双向认证可表示为：A – >B：A || Sa (Ta || Ra || B || Eb (Ya))

B – >A：Sb (Tb || Rb || A || Ra || Ea (Yb))

A – >B：Sa (Rb || B)

在公式中，A、B 是 A、B 的标识，Sx( ) 是 X 的签名，Tx 是一个时间戳，Rx 是 X 生成的不会重复的数字，Ex ( ) 是使用 X 的公开密钥进行的加密，Yx 是用户 X 的

① 本文为《浙江省教育科学规划重点研究课题》（编号 2007SB37）的研究成果

数据。

目前，Web 浏览器是安全证书的最著名用户，它使用证书来启用安全套接字层（SSL）协议。Web 站点也使用证书来将自己标识为浏览器用户。Java 程序员可以使用 java. security. cert 程序包并从开发商处购买程序包中定义的抽象类的实现，从而在应用程序中嵌入基于证书的认证。

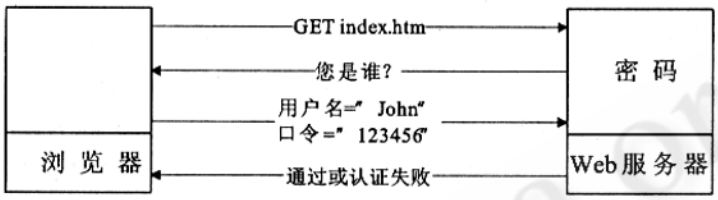

图 1　质询－响应安全机制

2.1.2　Apache 服务器实现证书认证

要在 LINUX + Apache + Tomcat + J2EE + openssl 环境下实现 SSL 的证书认证的步骤如下：

（1）安装 openssl 和 apache

（2）签证

安装 openssl 后，在 openssl 下有一个 CA. sh 文件，我们可以利用此文件来签三张证书：根证书、服务器证书、客户端证书，然后利用这三张证书来布署 SSL 服务。

（3）编辑 ssl. conf，设置如下：

SSLCertificateFile /xxx/ssl. crt/server. crt（指定服务器证书位置）

SSLCertificateKeyFile /xxx/ssl. crt/server. key（指定服务器证书 key 位置）

SSLCACertificatePath /xxx/ssl. crt（证书目录）

SSLCACertificateFile /xxx/ssl. crt/cacert. pem（根证书位置）

SSLVerifyClient require（开启客户端 SSL 请求）

SSLVerifyDepth 1

其中证书的存放位置用户可自行确定。设置完毕后启动 SSL 服务。

（4）安装和使用证书

把前面生成的根证书 ca. crt 和客户证书 client. pfx 下载到客户端，并安装，ca. crt 安装到信任的机构，client. pfx 直接在 Windows 中安装或安装到个人证书位置，然后用域名或 IP 访问 HTTP 和 https 服务器。更详细的步骤可参考 openssl 手册。

2.2　Apache 服务器的用户名、口令认证

2.2.1　用户名和口令认证原理

Web 服务器都提供了某种机制来控制资源的访问。在此我们以 Apache Web 服务器为例。Apache 支持基本认证和摘要认证两种形式，使用质询－响应协商的形式，图－1 说明了这种认证机制的工作原理。

在质询－响应协商中，客户首先请求访问特定的资源。如果对资源的访问受到控制，那么安全机制将截取这个请求并返回一个授权错误。浏览器将显示一个对话框，用户必须输入凭据信息。如果客户提供了足够的凭据，那么安全机制将允许这个请求；否则，安全机制将返回另一个授权错误。如果授权成功，浏览器通常会缓存认证信息，这样客户在会话持续期间就不需要再次提供这些认证信息。

摘要认证将凭据的单向散列添加到消息摘要中。这允许用户输入用户名和密码，以认证自己，但是不以明文形式在网络上发送这些凭据。摘要认证最严重的弱点是没有提供向服务器传递摘要的安全方式，第二个问题是这种模式没有提供任何方法在用户与特定用户的正确的服务器之间建立初始排列。

2.2.2　Apache 服务器上的用户名和口令认证

Apache 用户认证所需要的用户名和密码有两种不同的存贮方式：一种是文本文件；另一种是 MSQL、Oracle、MySQL 等数据库。在此着重阐述采用文本文件的存储方式的实现步骤。

这种认证方式的基本思想是：Apache 启动认证功能后，就可以在需要限制访问的目录下建立一个名为. htaccess 的文件，指定认证的配置命令。当用户第一次访问该目录的文件时，浏览器会显示一个对话框，要求输入用户名和密码，进行用户身份的确认。实现的具体步骤如下：

（1）修改 Apache 的配置文件 httpd. conf，对认证资源所在的目录设定配置命令。如要对 usr/local/apache/htdocs/internal 目录设置用户认证可配置如下：

<Directory /usr/local/apache/htdocs /internal >

Options Indexes FollowSymLinks
allowoverride authconfig
order allow,deny
allow from all
</Directory>

其中,allowoverride authconfig 一行表示允许对/usr/local/apache/htdocs/internal 目录下的文件进行用户认证。

(2) 在限制访问的目录/usr/local/apache/htdocs/internal 下建立一个文件. htaccess,其内容如下:

AuthName "访问该目录需要输入用户名和口令"
AuthType basic
AuthUserFile /usr/local/apache/internal. txt
require valid – user

文件. htaccess 中常用的配置命令请参考 Apache 手册。

(3) 利用 Apache 附带的程序 htpasswd,生成包含用户名和密码的文本文件。不要将此文本文件存放在 Web 文档的目录树中,以免被用户下载。

当用户数量比较少时,这种方法对用户的认证是方便、省事的,维护工作也简单。但是在用户数达到一定量时,会在查找用户上花掉一定时间,从而降低服务器的效率。这种情形下应采用数据库存储方式。

3 Servlet 中的证书认证和用户名、口令认证

Servlet 容器认证只与 Servlet 和 JSP 应用程序有关。Servlet2. 2 规范鼓励和要求 4 种认证方法:基本认证、摘要认证、基于表单的认证和使用 HTTPS 客户认证。

3.1 Servlet 容器中基于用户名、口令的认证

Servlet 容器中基于用户名、口令的认证有三种方式:基本认证、摘要认证、基于表单的认证。

3.1.1 基本认证

Servlet 容器的基本认证与 Web 服务器(如 Apache)中的基本认证完全一样。基本认证使用标准的 HTTP 验证,在此验证中服务器检查 Authorization 头。如果缺少这个头则返回 401 状态代码和一个 WWW – Authenticate 头。这将导致客户机弹出一个用于填写 Authorization 头的对话框。此机制很少或不提供对攻击者的防范,为为用户名和口令是用简单的可逆 base64 编码发送的。所有兼容的服务器都需要支持基本认证。

3.1.2 摘要认证

摘要认证的工作方式和基本认证相似,只是用户名和密码是以加密的形式返回的。对于抵抗非法的网络通信监测,加密后的用户名和密码更安全。摘要认证没有被 Servlet2. 3 规范要求,因为它目前没有被 Web 客户端广泛支持。

3.1.3 基于表单的认证

基于表单的认证允许提供一个自定义的登录页。Servlet 规范指定了在使用基于表单的认证时必须发生的操作。下面是当用户请求一个受保护的 Web 资源时发生的操作:(1)容器返回指定的登录表单并存储请求的 URL;(2)客户填写表单后发送回服务器;(3)容器使用表单域认证用户,如果认证失败则向客户返回指定的错误页面,若认证成功则容器将检查已验证的角色是否可以访问所请求的 URL(Web 资源);如果不能访问该 Web 资源,则 Servlet 容器返回指定的错误页面,如果能够访问,则将客户重定向到请求的 URL。

3.1.4 Tomcat 中实现三种用户名、口令认证

Tomcat 中实现三种用户名、口令认证需要 3 个步骤:

(1) 向% TOMCAT_HOME% \conf 中的 tomcat – users. xml 文件中的 <tomcat – users> 模块添加用户、密码和角色。

(2) 将 <security – constraint> 模块添加到应用程序的 web. xml 文件中,以便在该应用程序环境中配置它。

(3) 将 <login – config> 模块添加到应用程序的 web. xml 文件中,以便在该应用程序环境中配置它。

<login – config> 的子元素 <auth – method> 的取值为 BASIC 时采用基本认证机制,取值为 DIGEST 时采用摘要认证机制,取值为 FORM 时采用表单认证机制。对于基于表单的认证可以在 <login – config> 中添加子元素 <form – login – page> 和 <form – error – page> 来添加登录页和错误页。

3.2 Servlet 容器中基于证书的认证

Servlet 容器中基于证书的认证也就是 HTTPS 客户认证,它是最安全的一种认证形式,因为它要求客户使用数字证书来标识自己。客户认证通常使用 SSL 来实

现，一般都被普通的 Web 浏览器所支持。

3.3 Tomcat 中实现基于证书的认证

在此我们用 JSSE 设置 Tomcat 的 SSL，步骤如下：

(1) 生成密钥

用 Keytool 工具生成一个，如 tomcat. keystore，将该文件复制到 Tomcat 的 conf 目录下。

(2) 配置 Tomcat 的 server. xml

打开 server. xml 文件，找到注释行 <! -- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->，去掉对其下方的 Connector 节点的注释，并修改设置如下。

```
< Connector port = " 8443 " maxHttpHeaderSize
= " 8192 " maxThreads = " 150 " minSpareThreads = "
25 "
    maxSpareThreads = " 75 " enableLookups = "
false " disableUploadTimeout = " true "
    acceptCount = " 100 " scheme = " https " secure
= " true " clientAuth = " false " sslProtocol = " TLS "
    keystoreFile = " conf/tomcat. keystore " keystore-
Pass = " ChangeMe " / >
```

如果配置 Tomcat 验证客户身份，可以设置 clientAuth = " true " 。

(3) 配置 Web 站点的 Web. xml

在 Web 站点根目录 \WEB - INF 目录中找到 web. xml 文件，编辑该文件，在 < web - app > < /web - app > 节点中找到 < security - constraint > ，下面是 < security - constraint > 的配置样例：

```
< security - constraint >
  < description >
    Constrain the user data transport for the
whole application
  < /description >
  < transport - guarantee >
  CONFIDENTIAL
  < /transport - guarantee >
  < display - name > Example Security Constraint
< /display - name >
  < web - resource - collection >
    < web - resource - name > Protected Area
< /web - resource - name >
    < url - pattern > / * < /url - pattern >
  < http - method > DELETE < /http - method >
    < http - method > GET < /http - method >
    < http - method > POST < /http - method >
  < http - method > PUT < /http - method >
  < /web - resource - collection >
< /security - constraint >
```

(4) 启动 Tomcat，验证配置是否成功。

此设置适用于用 JSSE 设置 Tomcat 的 SSL。如果使用 APR，Tomcat 是通过 OpenSSL 来实现 SSL，设置会不同。

4 用户应用程序中运用 Session 和 Cookie 技术进行用户名、口令认证

在应用程序中进行用户名、口令认证很简单，关键是如何保存用户信息。我们知道 HTTP? 协议本身是无状态的，目前保持状态的方法通常采用 Session 和 Cookie。Cookie 机制采用的是在客户端保持状态的方案，而 Session 机制采用的是在服务器端保持状态的方案。

4.1 Cookie 机制分析

Cookie 是服务器发送给浏览器的信息量很小的纯文本文件信息。服务器发送信息是通过扩展 HTTP 协议来实现的，服务器通过在 HTTP 的响应头中加上一行特殊的指示以提示浏览器按照指示生成相应的 Cookie。不过纯粹的客户端脚本如 JavaScript 或者 VBScript 也可以生成 Cookie。Cookie 的内容主要包括：名字，值，过期时间，路径和域。路径与域合在一起就构成了 Cookie 的作用范围。

Cookie 的使用是由浏览器按照一定的原则在后台自动发送给服务器的。浏览器检查所有存储的 Cookie，如果某个 Cookie 所声明的作用范围大于等于将要请求的资源所在的位置，则把该 Cookie 附在请求资源的 HTTP 请求头上发送给服务器。

如果不设置过期时间，则表示这个 Cookie 的生命期为浏览器会话期间，只要关闭浏览器窗口，Cookie 就消失了。这种生命期为浏览器会话期的 Cookie 被称为会话 Cookie。会话 Cookie 一般不存储在硬盘上而是保存在内存里，当然这种行为并不是规范规定的。如果设置了过期时间，浏览器就会把 Cookie 保存到硬

盘上,关闭后再次打开浏览器,这些 Cookie 仍然有效直到超过设定的过期时间。

存储在硬盘上的 Cookie 可以在不同的浏览器进程间共享,比如两个 IE 窗口。而对于保存在内存里的 Cookie,不同的浏览器有不同的处理方式。对于 IE,在一个打开的窗口上按 Ctrl – N(或者从文件菜单)打开的窗口可以与原窗口共享,而使用其他方式新开的 IE 进程则不能共享已经打开的窗口的内存 Cookie;对于 Mozilla Firefox0.8,所有的进程和标签页都可以共享同样的 Cookie。一般来说是用 javascript 的 window.open 打开的窗口会与原窗口共享内存 Cookie。

4.2 Session 机制分析

Session 机制采用的是在服务器端保持状态的方案。当程序需要为某个客户端的请求创建一个 Session 的时候,服务器首先检查这个客户端的请求里是否已包含了一个 Session 标识,称为 Session id,如果已包含一个 Session id 则说明以前已经为此客户端创建过 Session,服务器就按照 Session id 把这个 Session 检索出来使用,如果客户端请求不包含 Session id,则为此客户端创建一个 Session 并且生成一个与此 Session 相关联的 Session id,Session id 的值应该是一个既不会重复,又不容易被找到规律以仿造的字符串,这个 Session id 将被在本次响应中返回给客户端保存。

保存这个 Session id 的方式可以采用 Cookie,这样在交互过程中浏览器可以把这个标识发还给服务器。由于 Cookie 可以被人为的禁止,必须有其他机制以便在 Cookie 被禁止时仍然能够把 Session id 传递回服务器。经常被使用的一种技术叫做 URL 重写,就是把 Session id 直接附加在 URL 路径的后面,附加方式也有两种,一种是作为 URL 路径的附加信息,另一种是作为查询字符串附加在 URL 后面。为了在整个交互过程中始终保持状态,就必须在每个客户端可能请求的路径后面都包含这个 Session id。另一种技术叫做表单隐藏字段。这种技术现在已较少应用。实际上这种技术可以简单的用对 action 应用 URL 重写来代替。

那么 Session 是何时被删除的呢?如果程序不通知服务器删除一个 Session,服务器就会一直保留,程序一般都是在用户做注销时时发个指令去删除 Session。浏览器从来不会主动在关闭之前通知服务器它将要关闭,因此关闭浏览器不会导致 Session 被删除。那么为什么有时关闭浏览器可以删除 Session 呢?那是因为大部分 Session 机制都使用会话 Cookie 来保存 Session id,而关闭浏览器后这个 Session id 就消失了,再次连接服务器时也就无法找到原来的 Session。如果服务器设置的 Cookie 被保存到硬盘上,或者使用某种手段改写浏览器发出的 HTTP 请求头,把原来的 Session id 发送给服务器,则再次打开浏览器仍然能够找到原来的 Session。由于关闭浏览器可能不能删除 Session,因此在服务器上为 Session 设置了一个失效时间,当距离客户端上一次使用 Session 的时间超过这个失效时间时,服务器就可以认为客户端已经停止了活动,把 Session 删除以节省存储空间。

5 小结

本文较系统地阐述了基于 JAVA 平台的 Web 应用系统常用的用户认证方法的原理和实现,希望对 WEB 开发人员选择认证技术时有所帮助。此外,LDAP 认证方案也是一种选择,限于篇幅未加以阐述。

参考文献

1 曹天杰、张永平、苏成,计算机系统安全[M],北京:高等教育出版社,2003.134 ~ 162.

2 张大治、邵勇、王欢,JSP 实用教程[M],北京:清华大学出版社,2006.173 – 184.

3 Java Servlet Technology Documentation. http://java.sun.com/products/servlet/docs.html.2003.05.

4 RFC2616 Hypertext Transfer Protocol (HTTP/1.1) . http://www.ietf.org/rfc/rfc2616.txt.1999.06.

5 RFC2459 Internet X.509 Public Key Infrastructure Certificate and CRL Profile. http://www.ietf.org/rfc/rfc2459.txt.1999.01