Advanced Authentication of ArcIMS Services

Michal Batko

Institute of Computer Science
Masaryk University, Brno, Czech Republic
xbatko@ics.muni.cz

Abstract. While publishing data using ArcIMS software, it is sometimes necessary to restrict the access to authorized personnel only. Two standard authentication mechanisms are provided by default - a file based authentication and a JDBC connection to a database. However, both the methods use only plain-text passwords and require exactly defined structures in either the file or the database. Moreover, only one file or one database can be used for authentication. Our proposal describes two different methods that enhance ArcIMS with advanced authentication. These methods provide the ability to authenticate an user with a password against any number of databases with different structures using different hash functions on a per IMS service basis.

1 Introduction

Nowadays, the information is the most valuable business article. It is estimated that more than 90% of information is produced in digital form and a part of it is produced in GIS systems. In general, it is a difficult task to maintain all the data gathered from different places all up-to-date. In such situation, it is handy to use an Internet Mapping Service to provide and distribute geographic always actual data to all its consumers. The ESRI provides an excellent piece of software ArcIMS, which in fact allows a common user to easily publish his spatial data through a webpage.

However, webpages are usually publicly available, so that anyone connected to the Internet and knowing the proper address can actually access all the data provided. Of course, this is not desirable always and sometimes, we need to restrict the access to authorized personnel only. Therefore, we need to gather the information about the user in some trusted way, so that we can be more or less sure of his identity. Then we can check our list of people allowed to access the service and either let the user in or refuse his request. The process of identity verification is usually called authentication and the second check is called authorization.

ArcIMS software provides some tools for authentication and authorization, but it has some limitations. We go through it briefly in Section 2. Our objectives for a more advanced authentication are provided in Section 3. The description of our solution according to the proposed objectives is available in Section 4.

2 Built-in ArcIMS Authentication

Figure 1 depicts standard flow of a request through different components of ArcIMS. Client's request is received by a web server (1), and it is immediately passed to a so called connector (2). In the usual installation, that we have focused on, the connector is a Java servlet running on Tomcat servlet engine (the Tomcat is also a web server itself, so it can provide both the steps). The connector contacts ArcIMS server (3), which is responsible for generating the requested data, e.g. an image with the map, and sends an answer back (4). The response is then propagated to the client by the same path (5,6).

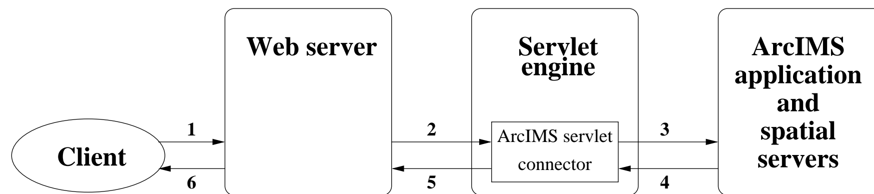

Fig. 1. Flow of a map request

The standard authentication in ArcIMS is provided by the connector. It uses so called HTTP authentication, which establishes the identity of an user through his login name confirmed by a password. The HTTP authentication is a well known standard and is implemented by almost every web browser. It has two modes – a *Basic* and a *Digest*. When the basic mode is used, client (e.g. web browser) passes the login and password provided by the user as is. The digest authentication first hashes the password using a one-way function and passes the login along with the scrambled password.

The servlet connector must verify the login/password pair get from the client in order to confirm the user identity. In particular, two forms are provided – a file based and a database based. The file authentication stores all the valid user/password pairs in a XML file along with the names of services, that that particular users can access. The second way, the database, provides similar functionality, but the logins, passwords and service names are stored in a database.

The built-in authentication is usually sufficient, but it has some drawbacks especially when a complex authentication mechanisms already exist in the institution and we want to use them transparently also for the ArcIMS services. The file based authentication is applicable only to a limited number of users and it is generally insecure. The database one is better, but the connector dictates the structure of the tables containing authentication specific data, which is a real problem in databases without view capabilities. Moreover, the passwords must be stored in plain text in both cases.

3 Objectives

In our university, we have multiple information systems (for example students agenda, employee intranet, etc.) with different authentication mechanisms (e.g. kerberos, active directory, database with crypt-hashed passwords, etc.). Therefore, we have set the following objectives for the ArcIMS authentication.

- User can be authenticated by any existing infrastructure service
- Modules for different services should be pluggable
- Several one-way functions for scrambling passwords should be supported
- User identity can be tried against multiple authentication services
- Once established identity should be cached for improved performance
- Request ArcXML document can be modified

4 Proposed Solution

In this section, we propose two strategies for satisfying the previous objectives. The first approach relies on the web server capabilities of authentication, but has some limitations. The second one provides fully transparent integration of the advanced authentication, but it is restricted to the Tomcat servlet connector only (which in fact is not so painful, since it is the most common installation setting). We have implemented the second approach and we successfully use it now.

4.1 Web Server Based Solution

Modern web servers, like IIS or Apache, allows transparent HTTP authentication to restrict particular URL provided by a server. The web server itself establishes the identity of an user, while he is accessing the restricted web pages tree. For every restricted ArcIMS service, there must be a separate directory with configured restricted area. Therefore, it is very difficult to maintain many map services. Also, the last objective, the modification of requests, is simply not possible.

4.2 Tomcat Solution

Our second proposed solution exploits the Tomcat features for authenticating, authorizing and filtering requests. In particular, we can design a java class that is plugged into Tomcat at the appropriate place so that related requests flow through it. In our solution, the authentication is strictly detached from authorization. See Figure 2 for clarification.

Once a request, that is in a restricted area, arrives, it is first passed to the authentication module. This module establishes the user identity as an instance of *principal*, which holds the required information about the user. Once a user identity is known, its request is passed to the authorization module. It checks the provided principal against the name of the service accessed and either allows or refuses to process the request. Finally, the request may be passed to a XSL transformation based on the user principal and the service name.

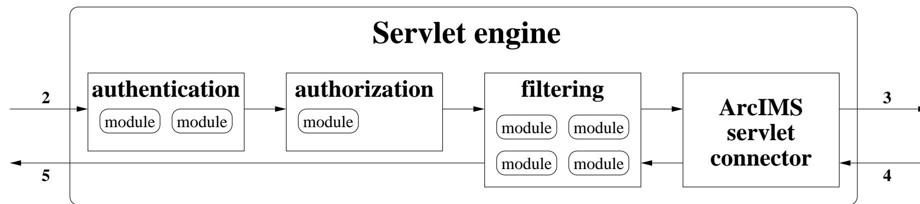

Fig. 2. Plugged modules for authentication, authorization and filter

Authentication Modules

In general, the authentication modules must confirm the login/password pair received from the client. As stated earlier, the client passes this information using HTTP authentication in either Digest or Basic form. In the Basic case, the password is first hashed using *hashfunct*, in the Digest case the client provide the hash functions that were used (usually MD5-DIGEST) and the authenticator must decide whether the hashed password is compatible with *hashfunct* or not.

In our implementation, the authentication module has a method authenticate, that receives login and password and either returns user principal or refuses the user as unauthorized. The implementation of *hashfunct* has two methods, one for hashing the password and one for retrieving the set of Digest compatible values.

In the following, we show an example of configuration for our JDBCAuthenticator module with UNIXCryptHashfunct plugin. JDBCAuthenticator is a generic module that allows to specify any JDBC database connection using any JDBC driver. In the example, a table *users* is accessed in a Microsoft SQL Server via its native JDBC driver. Observe also the `cacheSize` and `cacheTimeout` parameters, which specify that this particular authenticator will cache maximally one hundred login/password pairs each for one hour maximaly. Whenever a login/password is checked, the cache (if it has size greater than zero) is consulted and only if it is not successful, the database itself is queried and cache is updated. The hashfunct plugin makes password digests by the UNIX crypt algorithm.

```
<authenticator id="mmsqlusers"
    className="imsauth.authenticators.JDBCAuthenticator"
    driverName="com.microsoft.jdbc.sqlserver.SQLServerDriver"
    connectionURL="jdbc:microsoft:sqlserver://somehost:1432;
                   User=someuser;Password=somepassword"
    credentialsSQL="SELECT passwd = :password FROM users
                      WHERE login = :login"
    cacheSize="100"
    cacheTimeout="3600">
  <hashfunct className="imsauth.hashfuncts.UNIXCryptHashfunct"/>
</authenticator>
```

Authorization Module

To actually check whether a specified user has access to a specified service, we need to define the relationship. It is done using authorization module. Its implementation is similar to the authentication, it contains one method with the user principal and the service name arguments. It returns whether the user is allowed to access the service or not. In the example below, we use a comma separated text file authorization module for linking the login names with service names. The implementation of our CSVFileAuthorizer uses inner node text to parse the file. We can use any method provided by principal or a service name along with any text to define the format of a line.

```
<authorizer className="imsauth.authorizers.CSVFileAuthorizer">
  &quot;<principal methodName="getAuthenticatorName"/>&quot;,
  &quot;<principal methodName="getLoginName"/>&quot;,
  &quot;<servicename/>&quot;
</authorizer>
```

Example of the text file parsed by the authorizer above follows. Observe that first value is the authenticator name, which was introduced in `id` parameter. We can also see, that wildcard * is available. It is implemented in such a way, that the respective argument is not compared when the file is traversed. In the example, user with login *batko* authenticated by the module identified as *mmsqlusers* is allowed to access any ArcIMS services.

```
# This is a CSV file for ArcIMS CSVFileAuthorizer
"mmsqlusers", "batko", "*"
"*", "someone", "some arcims service"
"*", "someoneelse", "some arcims service"
"*", "someone", "some other arcims service"
```

XSL Transformation Module

This module allows to modify the client's ArcXML request using a XSL transformation. It can be used for restricting the map extent for a specific users, to restrict the allowed elements, etc. Provided the user principal and the name of the service, this module returns an XSL transformation that modifies the request. If there is no transformation defined, the request is passed as is. We have implemented a JDBC based filter module, which configuration is shown below. Its parameters are similar to JDBC authenticator. The only difference is in binding the principal information and service name to the SQL command, which is quite self-explanatory.

```
<xsltranform
    className="imsauth.xsltranforms.JDBCXSLTransform"
    driverName="sun.jdbc.odbc.JdbcOdbcDriver"
    connectionURL="jdbc:odbc:somedb;UID=someuser;PWD=somepasswd"
    xsltranSQL="SELECT transformation FROM xsltrans
                  WHERE login = :login AND serv_name = :srvname">
  <principal methodName="getLoginName" bindName="login"/>
  <servicename bindName="srvname"/>
</xsltranform>
```

5 Conclusion

Our new authentication schema for ArcIMS servlet connector allows much greater variability. We have successfully implemented all the tree modules for JDBC drivers, we also have a text file modules for authenticator and authorizer, and we have an experimental implementation of kerberos authentication.

In our university environment, we provide several ArcIMS services that are restricted to different groups of people. For example, we provide maps from GIS for keeping records about metropolitan academic computer network. Technicians are authenticated against the GIS itself and they can see everything. Employees of the university are authenticated against our personnel evidence system and they cannot use identify functions. University students are denied to see labels in some layers and they also have applied some attribute filters.

We use the advanced authentication for a half year now, and our experience is very good, the schema proves to be stable and reasonably fast. The installation of the advanced authentication is quite easy, it consists of copying a few Java class files to appropriate directories of the Tomcat installation.