Authentication Factory 2.0 Component Specification

1. Design

This upgrade of the Authentication Factory is intended to make the component truly pluggable. Although the current design applies Factory pattern, it's heavily bound with HTTP authentication backend. The new design will make the component independent of underlying authentication mechanism.

Since back compatibility is not required, this design chooses to redesign the framework in order to incorporate the new requirements easily, and remove those obsolete classes.

AuthenticationFactory is now able to create the Authenticator from the configuration values defined in the configuration file via Configuration Manager component. And all the Authenticators are configuration based too. Client can change the Authenticator's behavior by simply modify the parameters in the configuration file rather than modify the application code.

Authenticator is now authenticating against a Principal object, and returns a Response object as the authentication result.

Principal is rather similar to the Map argument in the old design, except that it contains an ID property, which is used to uniquely identify a principal, and cache the associated response in the simple cache.

Response replaces the old empty Resource interface. It contains a successful flag, an optional message string, and a details object. In the old design, AuthenticateException will be thrown if authentication fails. It is not appropriate if client want to know more details about the failures. So in current design, AuthenticateException is only used to encapsulate the underlying implementation specific exceptions, the authentication result will be saved to the response object, more specifically the resource returned from the authentication will be stored to the details property of the response.

AbstractAuthenticator is completely redesigned, and is now able to cache the response to the principal to authenticate and convert authenticator specific keys to ones recognizable by Principal. The caching functionality could greatly improve the response of the application, and it could be turned on or off via the configuration. The key conversion might help if we switch to another Authenticator in the application, which the new one uses a different collection of keys for authentication. We only have to change the key conversion configuration to make the new authenticator function well.

The new requirement expects that the application could switch to another Authenticator implementation without changing the code. Current design attempts to provide the maximum flexibility, but there are still some limitations – which is reflected in the details property in the Response class. There is no general interface for details returned from authentications, and application makes a switch, it should switch to an Authenticator of the same kind if it doesn't want to change the code. The same kind here means the details returned from the authenticator implement the same interface.

Here is more discussion per requirement:

1.1.1 Independent of Authentication Backend

The Authenticator could be created from the configuration values defined in the configuration file via Configuration Manager component in

AuthenticationFactory, client could switch to another authenticator using a different name.

1.1.2 Support Multiple Means of Authentication

Principal class is responsible for holding all the information for authentication, and PrincipalKeyConverter is used during the authentication to convert the authenticator specific keys to the ones recognizable to the Principal.

1.1.3 A Common Interface for Authentication Return Value

Response class is designed to hold the authentication result; it contains two properties, which can be shared by all kinds of authentications – successful flag and message string. Extra details will be saved to Response's details property, since there is NOTHING in common in this aspect, object type is used, but the authenticator implementations should make the details object implement an interface, just like what HttpResource and HttpResourceImpl do.

1.1.4 Support Current Web/HTTP Based Authentication

HttpBasicAuthenticator implementation is modified to load the authentication arguments from Principal, and save the result in the Response, the authentication algorithm is not changed.

1.1.5 Separate Web / HTTP Authentication Plug-in From Component Archive

They are moved in the com.topcoder.security.authenticationfactory.http.basicimpl package, and could be put into a separate jar.

1.1.6 An Optional Caching Mechanism

The caching functionality is provided in the AbstractAuthenticator class. We could turn off the caching by removing the cache_factory configuration from the configuration file, and turn on by adding it.

1.2 Design Patterns

AuthenticationFactory implements the singleton pattern.

Authenticator, CacheFactory, and PrincipalKeyConverter implement the strategy pattern.

AbstractAuthenticator implements the template pattern, and its subclasses could inherit the caching functionality automatically.

CacheFactory implements the factory pattern to create a simple cache with specified timeout.

Principal uses the Value Object pattern.

1.3 Industry Standards

Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, (http://www.ietf.org/rfc/rfc2616.txt)
HTTP Authentication: Basic and Digest Access Authentication, RFC2617, (http://www.ietf.org/rfc/rfc2617.txt)

1.4 Required Algorithms

Developer is encouraged to optimize algorithm.

1.4.1 *Http Basic Authentication*

The http basic authentication algorithm is the same as in the old version, except the implementation will now read the protocol, host, port, file, username, and password from the Principal argument rather than the Map argument. And it will return a Response object, saving the created HttpResourceImpl object into the details property of the Response.

```
// get the properties from the Principal argument

// create new URL and open stream
URL u = new URL(protocol, host, port, file)

redirect:

// creates HttpURLConnection
HttpURLConnection conn = u.openConnection()

// HTTP (including authorization) configuration
conn.setRequestMethod(HttpURLConnection.POST)
conn.setRequestProperty("Authorization","Basic " + base64encode
(username+":"+password))

// additional HTTP protocol properties may be provided by client
java.util.Map otherProps =
(java.util.Map)getValue(REQUESTPROPERTIES_KEY)

foreach (key,value) in otherProps
conn.setRequestProperty(key,value)

// read/send something to perform authorization
// get response code
int rcode = conn.getResponseCode();

if (rcode is any redirect)   // for example HTTP_MOVED_PERM
{
      // request new URL
      String npath = conn.getHeaderField("location");
      u = new URL(u,npath);
      goto redirect;
}
HttpResource resource = HttpResourceImpl(conn,originalURL);

rcode = conn.getResponseCode();

if (rcode >= 200 && rcode < 300 {
      // return a Response object.
      return new Response(true, "successful", resource);
} else {
      return new Response(false, "failed", resource);
}
```

Note that HTTPBasicAuthentication.doAuthenticate() have to convert user-password information to Base-64 encoding (see RFC2184)..

1.4.2 Caching functionality in the AbstractAuthenticator

AbstractAuthenticator is able to create a CacheFactory instance basing one the configuration values, and the CacheFactory is responsible to create the Cache to be used in the authenticator.

Authenticator implementations should extend the AbstractAuthenticator abstract class and override the doAuthenticator method to inherit the caching functionality automatically.

1.5 Component Class Overview

AuthenticationFactory

AuthenticationFactory class maintains a collection of Authenticator instances, which are created from the configuration values via the Configuration Manager. Each Authenticator has a unique name to identify it. AuthenticationFactory provides a method to retrieve the Authenticator by its name. If the configuration values are changed, refresh can be called to recreate the Authenticators, so that we don't have to restart the application to be ware of the change.

Authenticator

Authenticator interface defines the contract to authenticate a principal, which holds the necessary information for specific authentication implementation, and return a response indicating the authentication is successful or failed. In current release, AbstractAuthenticator implements this interface to provide some common functionality that can be shared among all implementations - such as caching authentication response etc.

Principal

Principal class represents the user entity of an application to be authenticated. It contains an object type unique id, which is used to uniquely identify the principal, so that if the cache mechanism is turning on, the authentication response can be retrieved from the cache directly by the principal's id. It also holds a collection of attributes, which provide necessary information for specific authentication implementation.

Response

Response class represents the authentication result returned from the Authenticator's authenticate method. It contains a successful flag indicating the authentication succeeded or failed, an optional authentication message, and a details object. The message could be possibly the error message or error code specified in the authentication, and more detailed information should be assign to the details property. For example, in the HttpBasicAuthenticator, the details would be of HttpResource type. Specific authenticator implementation should always design an interface for the object assigned to details property, so that we don't have to modify the application when switching to another authenticator of the same kind.

CacheFactory

CachFactory interface defines the contract to create the Cache instance. The Cache interface is defined in the Simple Cache component. This interface defines a single method to return the created Cache, extra information should be provided in the implementation to specify what kind of cache should be created. Current release provides a TimeoutCacheFactory implementation, which creates a timeout simple cache.

TimeoutCacheFactory

TimeoutCacheFactory class implements the CacheFactory interface, which will return a timeout simple cache. The createCache method will return a SimpleCache instance from the Simple Cache component with the specified time out, SimpleCache.NO_MAX_SIZE,

and LRUCacheEvictStrategy. If the time out is 0, SimpleCache.NO_TIMEOUT will be used instead, indicating the entries in the cache will never timeout.

PrincipalKeyConverter

PrincipalKeyConverter interface defines the contract to convert the keys used by the specific authenticator implementation into those in the Principal. Current release provides the DefaultKeyConverter implementation, which is able to load the key mappings from the configuration files via Configuration Manager component. So if we switch to another authenticator, which might use a different collection of keys, we only have to modify the configuration file, rather than change the application code.

DefaultKeyConverter

DefaultKeyConverter class implements the PrincipalKeyConverter interface; it is able to load the key mappings from the configuration file via the Configuration Manager component. The key mappings are loaded in the constructor once and saved to the mappings variable.

AbstractAuthenticator

AbstractAuthenticator abstract class implements Authenticator interface, and provides some common functionalities that can be shared by specific authenticator implementations. It provides a conversion from the specific authenticator's key to principal's key, so that if we switch to another authenticator, we only have to update the conversion configuration. It holds a cache to cache the response of the authenticated principal, so that we can return the cached response to the same principal without do the time-consuming authentication repeatedly. And the cache mechanism can be turned off / on in the configuration file.

HttpCookie

This class represents a cookie provided by HTTP server to client in the 'Set-Cookie' header field.

HttpResource

Represents a HTTP resource wrapper. Defines several methods to retrieve different information about HTTP server response such as HTTP header fields, server cookies, redirection URL, content type and content itself. Web authenticators can use this interface to provide common access to returned HTTP resources.

HttpBasicAuthenticator

This class performs HTTP basic authentication. As noted, the HTTP server might ask for, in its response to the client's request, for the client to provide credentials. The client would send another request with the desired information, and the server would then verify the information. This class defines several keys that are used during authentication, and are stored in the map. All keys are Strings, and all Values are defined to be of specific types. All keynames are case-sensitive. Certain keys are immutable, in as much as they can't be changed, and no values can be used in their stead.

HttpResourceImpl

HttpResourceImpl implements HttpResource interface. It maintains reference to HTTP connection with server

1.6 Component Exception Definitions

AuthenticateException

AuthenticationException is thrown from the Authenticator's authenticate method if some unexpected errors occur during the authentication. It is used to encapsulate the exception thrown from the underlying authentication implementation, such as the network fails for the http authentication.

InvalidPrincipalException

InvaildPrincipalException is thrown from the authenticator's authenticate method if the principal is invalid, for example, the value of a certain key is invalid, etc.

MissingPrincipalKeyException

MissingPrincipalKeyExceptionis thrown from the Authenticator's authenticate method if certain key is missing in the principal, which is required for the authentication. For example, the username property is missing in NTLM authentication.

ConfigurationException

This exception is thrown when fail to loading configuration values via the Configuration Manager, or the loaded values are in invalid format. Refer to the documentation in Poseidon for more details.

NullPointerException:

This exception is thrown in various methods where null value is not acceptable.

Refer to the documentation in Poseidon for more details.

IllegalArgumentException

This exception is thrown if empty string is specified. Refer to the documentation in Poseidon for more details.

IOException

This exception is thrown if exception occur during HttpResource contents retrieval.

1.7 Thread Safety

This component could be used in web application for authentication, so thread-safety is quite important to it. And all the classes in this component are thread-safe except the Principal class, which is not expected to be shared by multiple threads; each thread should hold a separate Principal instance.

AuthenticationFactory, Response, TimeoutCacheFactory, DefaultKeyConverter, HttpCookie, HttpBasicAuthenticator, and HttpResourceImpl are all thread-safe since they are immutable.

AuthenticatorFactory’s getInstance method must be synchronized properly.

And if new implementations of CacheFactory and PrincipalKeyConverter, they are expected to follow the immutability convention too.

AbstractAuthenticator contains a protected field; its subclasses are expected to change this field inside the constructor only.

2. Environment Requirements

2.1 Environment

jdk 1.3 or higher

2.2 TopCoder Software Components

Configuration Manager 2.1.3

It is used to load the configuration values to create the Authenticators.

Base Exception 1.0

Custom exceptions extend from the BaseException and BaseRuntimeException from this component.

Simple Cache 1.0

AbstractAuthenticator will cache the principal's ID and principal's authenticated response in the cache.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.security.authenticationfactory

com.topcoder.security.authenticationfactory.http

com.topcoder.security.authenticationfactory.http.basicimpl

3.2 Configuration Parameters

Configuration values for AuthenticationFactory

Parameter	Description	Values
authenticators	Contains all the configured authenticator property. **Required**.	http, ftp etc.
xx/class	Full qualified class name for specific authenticator. xx is the name of the authenticator. **Required**.	HttpBasicAuthenticator
xx/namespace	The namespace for specific authenticator's constructor. xx is the name of the authenticator. **Required**.	com.HttpBasicAuth

Configuration values for AbstractAuthenticator

Parameter	Description	Values
principal_key_converter	Contains class/namespace sub properties to create the converter. **Optional**.	none
principal_key_converter /class	Full qualified class name for the converter. **Required**.	DefaultKeyConverter
principal_key_converter /namespace	The namespace for converter's constructor. **Required**.	com.DefaultConverter
cache_factory	Contains class/namespace sub properties to create the converter. **Optional**.	none
cache_factory/class	Full qualified class name for the cache factory. **Required**.	TimeoutCacheFactory
cache_factory/namespace	The namespace for cache factory's constructor. **Required**.	com.TimeoutFactory

Configuration values for HttpBasicAuthenticator

Parameter	Description	Values
default_mappings	Contains all the mappings saved to the defaultMappings field. **Optional**.	none.
protocol	The protocol for the http authentication. If not specified, default to http. **Optional**.	http
port	The port for the http authentication. If not specified, default to 80. **Optional**.	com.HttpBasicAuth
host	The name of the host. **Optional**.	www.topcoder.com
file	The file in the realm the client wants to access. **Optional**.	index.html
request_properties	String to string map of Http request properties. Its sub properties will be stored into a Map and then associated with this key. Optional.	Content-Type – text/html

Configuration values for DefaultKeyConverter

Parameter	Description	Values
mappings	Contains all the mappings, the name of its sub property is principal's key, and the value is authenticator's key. **Optional**.	username - UserName

Configuration values for TimeoutCacheFactory

Parameter	Description	Values
timeout	The timeout of the created cache, in millisecond. Non negative values. **Required**.	1000.

Here is an example of the configuration file:

```
       <?xml version="1.0" ?>
<CMConfig>
  <Config
       name="com.topcoder.security.authenticationfactory.AuthenticationFactory">
     <Property name="authenticators">
        <!-- http authenticator -->
        <Property name="http">
          <!-- the full qualified class name, required. -->
          <Property name="class">

<Value>com.topcoder.security.authenticationfactory.http.basicimpl.HTTPBasicAuth
enticator</Value>
          </Property>

          <!-- the namespace to load configuration values, required. -->
          <Property name="namespace">
            <Value>com.topcoder.security.http.HTTPBasicAuthenticator</Value>
          </Property>
        </Property>

        <!-- another authenticator, only an demonstration here -->
        <Property name="ftp">
          <Property name="class">
            <Value>com.topcoder.security.FTPAuthenticator</Value>
          </Property>
          <Property name="namespace">
            <Value>com.topcoder.security.CustomNamespace</Value>
          </Property>
        </Property>
     </Property>

  </Config>

  <Config name="com.topcoder.security.http.HTTPBasicAuthenticator">
    <!-- properties for the AbstractAuthenticator -->

    <!-- default mappings, optional, if not specified, default mapping is empty
-->
    <Property name="default_mappings">
      <Property name="protocol"><Value>http</Value></Property>
      <Property name="port"><Value>80</Value></Property>
      <Property name="host"><Value>www.topcoder.com</Value></Property>
      <Property name="file"><Value>index.html</Value></Property>
      <Property name="request_properties">
        <Property name="accept"><Value>text/html</Value></Property>
      </Property>
    </Property>

    <!-- principal key converter, optional, if not specified,
        DefaultKeyConverter is used. -->
    <Property name="principal_key_converter">
      <!-- full qualified class name for converter, required. -->
      <Property name="class">
```

```
<Value>com.topcoder.security.authenticationfactory.DefaultKeyConverter</Value>
      </Property>
      <!-- namespace to create the converter, required. -->
      <Property name="namespace">

<Value>com.topcoder.security.authenticationfactory.DefaultKeyConverter</Value>
      </Property>
    </Property>

    <!-- cache factory, optional, if not specified,
              response is not cached.  -->
    <Property name="cache_factory">
      <Property name="class">

<Value>com.topcoder.security.authenticationfactory.TimeoutCacheFactory</Value>
      </Property>
      <Property name="namespace">

<Value>com.topcoder.security.authenticationfactory.TimeoutCacheFactory</Value>
      </Property>
    </Property>
  </Config>

  <!-- properties for DefaultKeyConverter -->
  <Config
name="com.topcoder.security.authenticationfactory.DefaultKeyConverter">
    <!-- mapping from the principal's key to Authenticator's key, optional. -->
    <Property name="mappings">
      <!-- the name contains authenticator's key, and the value contains
principal's key. -->
      <Property name="username">
        <Value>UserName</Value>
      </Property>
      <Property name="password">
        <Value>Pwd</Value>
      </Property>
  </Config>

  <!-- properties for TimeoutCacheFactory -->
  <Config
name="com.topcoder.security.authenticationfactory.TimeoutCacheFactory">
    <!-- timeout in milli-seconds, non-negative long value,
           0 means NO_TIMEOUT, required -->
    <Property name="timeout">
      <Value>1000</Value>
    </Property>
  </Config>

  <Config name="com.topcoder.security.CustomNamespace">
    <!-- use default values for AbstractAuthenticator,
      it will create the DefaultKeyConverter using
      'com.topcoder.security.CustomNamespace' namespace -->
  </Config>
</CMConfig>
```

3.3 Dependencies Configuration

The ConfigurationManager should be properly configured.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Install webapp1 from the test_files\tomcat directory into the webapps directory of your Tomcat installation.
- Copy the tomcat-users.xml file from the test_files directory into the conf directory of your Tomcat installation.
- The default port the testcase will connected is 8080, also you can configure this port by set 'jvmarg value="-Dtestport=${port}' in test task in build.xml to other port, and then make sure Tomcat is configured to listen on localhost port ${port}, and that the machine you are running the test on can connect to www.topcoder.com.
- Execute 'ant test' within the directory that the distribution was extracted to.

4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.

4.3 Demo

```
The configuration file listed in the section 3.2 will be used in the
following demo.


// get the singleton instance
AuthenticationFactory factory = AuthenticationFactory.getInstance();

// get the authenticator defined in the configuration file
// the HttpBasicAuthenticator will be returned as defined in the
// example configuration file above.
Authenticator auth = factory.getAuthenticator("http");

// create a Principal to authenticate, use the username as the id.
Principal principal = new Principal("mess");
principal.addMapping("username", "mess");
principal.addMapping("password", "******");

// authenticate the principal
Response response = auth.authenticate(principal);

// check the authentication fails or not
if (response.isSuccessful()) {
      System.out.println("authentication succeeds");
}
// get the authentication message if there is one
System.out.println(response.getMessage());

// get the response details, we should know in advance it is an
// http authentication implementation
HttpResource resource = (HttpResource) response.getDetails();

// print the properties of HttpResource
System.out.println(resource.getOriginalURL());
System.out.println(resource.getActualURL());
System.out.println(resource.getContentType());
System.out.println(resource.getContent());
System.out.println(resource.getSetCookie());
```

```
..

// to turn off the caching functionality
// simply comment out the ‘cach_factory’ section in the configuration
// file for the ‘http’ authenticator and refresh the configuration
factory.refresh();

auth = factory.getAuthenticator(“http”);

// in order to switch to another http implementation using different
// collection of keys, we could change the mappings in the
// PrincipalKeyConverter used by the ‘http’ authenticator without
// changing the application code.
// assume the new authenticator use the ‘UserName’ rather than ‘username’
// and there is a mapping defined from ‘UserName’ to ‘username’, the
// following authentication will work as above.
response = auth.authenticate(principal);

// add port, protocol, host, file info into the principal, which will
// override the default values specified in the configuration file.
principal.addMapping(“protocol”, “ftp”);
principal.addMapping(“port”, “21”);
principal.addMapping(“host”, “ftp.topcoder.com”);
principal.addMapping(“file”, “authenticator.jar”);
response = auth.authenticate(principal);
```

5. Future Enhancements

More authenticator implementations could be added. And more PrincipalKeyConverter and CacheFactory implementations could be added if they cannot satisfy new requirements.