

## Valve – Installation and Configuration Guide

### Google Enterprise EMEA

This document explains how to install the GSA Valve Security Framework on a Tomcat instance and the basics about how to configure it. There are some other documents available that can help you on understanding how to deploy the Valve like the Scenario Guide.

The Valve is a Java application that has been tested on Tomcat but theoretically it can run on any standard J2EE web container. It uses some external open source libraries, most of them are Apache ones. It also uses some Google libraries that offer some external functionality that you can find in the code repository.

The Tomcat version you can use has to be 5.5 release or upwards. You have to use JDK version 6 at least.

The configuration is driven by a flexible XML file where you can define tons of different repositories and architecture. If you need to extend it you can easily do just creating new modules using Java code.

We strongly recommend you to read the Introduction guide before proceeding if it was not done yet. It'll introduce you on the technical aspects of the security framework and explains you the two main technical interfaces provided: SAML and Forms Based. The last one is the default interface and it's used unless the following parameter is set:

```
<saml    isSAML="true"  
...  
>
```

Another worth reading document is the GSA guide that explains the complete lifecycle for secure documents in the appliance (GSA 5.0 version): [http://code.google.com/apis/searchappliance/documentation/50/secure\\_search/secure\\_search\\_crwlsrv.html](http://code.google.com/apis/searchappliance/documentation/50/secure_search/secure_search_crwlsrv.html).

## Installation

The GSA Valve Security Framework is built as a Tomcat application and has to be deployed as any other standard application. You can find in the source code some files that help on creating a fully standard Java application ready to be deployed on Tomcat.

The application contains both a Jar deployment file (`valve.jar` or `valve_XX.jar` where XX is the version number) and some classes that have to be deployed on Tomcat. There are also some external open source libraries that you can download and put as well in the application server to let the Valve make use of them. And some other Google libraries that work exactly the same way.

### Requirements

The initial requirements for the Valve are the following:

- Tomcat 5.5.23 or upwards (It's been fully tested using 5.5.25). Some Tomcat bugs have been faced using 5.5.26, so it's not recommended it.
- JDK 6 or upwards.
- The application is independent of the O.S. platform
- There could be some specific requirements depending on the way it's deployed. Have a look at the deployment scenario guide for further information.

### How to Build the Application

You have the Valve binaries available at [code.google.com](http://code.google.com), and the external libraries needed, but just in case you'd like to build the application by your own from scratch you can do so. The source code includes an Ant's `build.xml` file that compiles the classes and creates the libraries required but you can use any other Java deployment tool.

If you want to use the local `build.xml` file, you should modify the location of the directories included in the file to point to the right directories in your environment. This file is interpreted by an Ant tool that you can download it or just use a plug-in that is usually available for the main Java IDE like Eclipse or JDeveloper.

It will create a deployment directory with the deployment files, organized in some directories:

- `classes`: it contains the application classes
- `deploy`: includes the application deployment architecture:
  - `extlib`: it contains all the Valve framework classes in one library. This is meant to be used as a external library in third party applications.
  - `webAppServerCommon`: it includes the common deployment directories both for the SAML and the Forms Based interface:
    - `common/classes`: Valve configuration and logging property files. You can leave this files in the same location in the Tomcat server (CATALINA\_HOME or TOMCAT\_INSTALL).
    - `src`: source code (not needed for deployment)
    - `webapps/valve`: application deployment directory. You can copy it

at the Tomcat's `webapps` location. You can use an alternative name to "valve".

- `webAppServerFormsBased`: this directory only applies for the Forms Based interface. The reason of using this additional content is because the Valve filter and the application are running in two different classloaders. It holds the following directory:
  - `server`: contains the application `classes` and `lib` directories at the server classloader. Point them at the Tomcat's `server` directory.

This structure would help you when deploying the application. Follow the following rules to deploy the application:

- **SAML interface**: if you are using the SAML interface, you can use the directory structure contained in `webAppServerCommon` directory. Use the `CATALINA_HOME` directory as a reference.
- **Forms Based interface**: apart from deploying the content held in `webAppServerCommon` directory, you also have to deploy what it's included at `webAppServerFormsBased`. The reason is the Forms Based interface makes use of two separate classloaders as Valve filter is running on the Server classloader, different than where the application is really running.

The SAML interface is set up when the `saml` configuration tag contains the attribute `isSAML` set to true. If it's false then you are using the Forms Based interface.

Apart from the above files, you have to also deploy the external libraries the Valve is making use of as it'll be explained later on.

## Libraries

The framework is using different libraries that have to be included both when building the application and running it. Here is the list of libraries used when building the application:

- **Open source and Third-Party libraries**: they offer some functionality that Valve framework classes are using. Most of them are part of the Apache projects. The following ones have to be used for compiling all the classes:
  - *Apache Log4J* (1.2.14 or upwards) <<http://logging.apache.org/log4j/>>
  - *Apache Jakarta RegExp* (1.5 or upwards) <<http://jakarta.apache.org/regexp/>>
  - *Apache Commons HttpClient* (3.0.1) <<http://hc.apache.org/>>
  - *Apache Commons Collections* (3.2 or upwards) <<http://commons.apache.org/collections/>>
  - *Apache Commons EL* (1.0) <<http://commons.apache.org/el/>>
  - *Apache Xerces* (2.9.0) <<http://xerces.apache.org/xerces2-j/>>
  - *Apache Commons Digester* (1.8 or upwards) <<http://commons.apache.org/digester/>>
  - *Apache Commons BeanUtils* (1.7 or upwards): <<http://commons.apache.org/beanutils/>>
  - *Apache Commons Codec* (1.3) <<http://commons.apache.org/codec/>>
  - *Html Parser* (1.6) <<http://htmlparser.sourceforge.net/>>
  - *JDom* (1.1) <[www.jdom.org](http://www.jdom.org)>
  - *Axiom* (1.2.5) <<http://ws.apache.org/commons/axiom/>>
  - *Stax* (1.0.1 or onwards) <<http://stax.codehaus.org/>>
  - *Tomcat* (Catalina.jar)
  - *Servlet API* (servlet-api.jar)

- **Google libraries:** they offer some functionality for the Valve or some other applications that have been independently packed for better usability:
  - *GSA Valve Kerberos libraries:* Kerberos implementation for SPNEGO services in the Valve <ValveKrb5.jar>
  - *GSA Valve Session libraries:* Valve session management <ValveSessions.jar>
  - *Sniffer* (1.3 or onwards): Http libraries that act as a standard browser <Sniffer.jar or Sniffer\_1.3.jar>

### Forms Based Interface

If you decide to use the Forms Based interface to integrate the framework with the GSA, then you have to follow the following steps:

#### A.- Library Deployment

The following libraries have to be deployed at least on Tomcat when using the **Forms Based interface**. You also have the option to deploy all the libraries mentioned in the previous chapter, but the following ones is the minimum set to make the application up and running:

- `$(TOMCAT_INSTALL)/common/lib`: the following libraries have to been located here:
  - Valve Sessions <ValveSessions.jar>
  - Xerces <xercesImpl.jar>
  - Log4J <log4j-1.2.14.jar>
  - Commons Codec <commons-codec-1.3.jar>
  - Commons EL <commons-el.jar> (if not already exists)
- `$(TOMCAT_INSTALL)/server/lib`: include the following libraries:
  - Sniffer <Sniffer.jar or Sniffer\_1.3.jar >
  - Valve <Valve.jar or Valve\_2.0.jar >
  - Valve Kerberos <ValveKrb5.jar>
  - Apache Jakarta RegExp <jakarta-regexp-1.5.jar>
  - Apache Commons HttpClient <commons-httpclient-3.0.1.jar>
  - Apache Commons Collections <commons-collections-3.2.jar>
  - Apache Commons Digester <commons-digester-1.8.jar>
  - Apache Commons BeanUtils <commons-beanutils.jar>
  - Html Parser <htmlparser.jar>
  - JDom < dom.jar>
- `$(TOMCAT_INSTALL)/webapps/valve/WEB-INF/lib`: include the next libraries (the same as the previous directory):
  - Sniffer <sniffer.jar or sniffer\_1.3.jar >
  - Valve <valve.jar or valve\_2.0.jar >
  - Valve Kerberos <ValveKrb5.jar>
  - Apache Jakarta RegExp <jakarta-regexp-1.5.jar>
  - Apache Commons HttpClient <commons-httpclient-3.0.1.jar>
  - Apache Commons Collections <commons-collections-3.2.jar>
  - Apache Commons Digester <commons-digester-1.8.jar>
  - Apache Commons BeanUtils <commons-beanutils.jar>
  - Html Parser <htmlparser.jar>
  - JDom < dom.jar>

## B.- Tomcat Configuration

The **Forms Based interface** requires the Valve filter context to be included in the Tomcat configuration. It will be done just adding the following line to Tomcat's `server.xml`:

```
<Valve className="com.google.gsa.Valve"
  gsaValveConfigPath="{full path to valve configuration file}"/>
```

This file, `gsaValveConfig.xml`, is usually located at `$TOMCAT_HOME/common/classes`.

## SAML Interface

In the case you decided to use the SAML interface, the deployment is going to be much easier as everything will be packaged at the application deployment directory.

## A.- Library Deployment

The following libraries at least have to be deployed at `$TOMCAT_INSTALL/webapps/valve/WEB-INF/lib`:

- Sniffer <sniffer.jar or sniffer\_1.3.jar >
- Valve <valve.jar or valve\_2.0.jar >
- Valve Kerberos <ValveKrb5.jar>
- Valve Sessions <ValveSessions.jar>
- Apache Jakarta RegExp <jakarta-regexp-1.5.jar>
- Apache Commons HttpClient <commons-httpclient-3.0.1.jar>
- Apache Commons Collections <commons-collections-3.2.jar>
- Apache Commons Digester <commons-digester-1.8.jar>
- Apache Commons Codec <commons-codec-1.3.jar>
- Apache Commons BeanUtils <commons-beanutils.jar>
- Html Parser <htmlparser.jar>
- JDom < dom.jar>
- Xerces <xercesImpl.jar>
- Log4J <log4j-1.2.14.jar>
- Axiom <axiom-\*.jar>

## B.- Tomcat/Web Application Configuration

As well as when Forms Based interface is in place, we have to tell the SAML frontend where the security framework configuration file is. As we are not using separate Java classloaders, we can set it up in the application's `web.xml` file that should be located at `$TOMCAT_HOME/webapps/valve/WEB-INF`.

```
<!-- Valve Config Location. Only if SAML is enabled -->
<env-entry>
  <description>GSA Valve Config Path</description>
  <env-entry-name>gsaValveConfigPath</env-entry-name>
  <env-entry-value>
    C:\Tomcat\common\classes\gsaValveConfig.xml
  </env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Just substitute in the above example, the exact location of your `gsaValveConfig.xml` file.

## Logging

Logging is implemented in all classes using the standard Apache Log4j library. The logging configuration file is located at:

```
$TOMCAT_HOME/common/classes/log4j.properties
```

Below is how a sample logging configuration file looks like. You can change it to just logging those classes you're interested in and with the proper logging level:

```
# By default all com.google.gsa classes log DEBUG messages to a single log file
log4j.rootLogger=DEBUG
log4j.logger.com.google.gsa=DEBUG, R
log4j.logger.org.apache.catalina.startup.Catalina=DEBUG, R
log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=${catalina.home}/logs/valve.log
log4j.appender.R.MaxFileSize=10MB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.layout=org.apache.log4j.PatternLayout

# Date/time priority thread class message linefeed
log4j.appender.R.layout.ConversionPattern=%d{dd MMM yyyy HH:mm:ss,SSS} %p %t %c - %m%n

#Individual logging control for specific classes, adjust as needed
log4j.logger.com.google.gsa.IWebProcess=DEBUG
log4j.logger.com.google.gsa.Valve=DEBUG
log4j.logger.com.google.gsa.Authenticate=DEBUG
log4j.logger.com.google.gsa.valve.rootAuth.RootAuthenticationProcess=DEBUG
log4j.logger.com.google.gsa.valve.rootAuth.RootAuthorizationProcess=DEBUG
log4j.logger.com.google.gsa.valve.configuration.ValveKerberosConfiguration=DEBUG
log4j.logger.com.google.gsa.valve.configuration.ValveRepositoryConfiguration=DEBUG
log4j.logger.com.google.gsa.valve.configuration.ValveSessionConfiguration=DEBUG
log4j.logger.com.google.gsa.valve.modules.httpbasic.HTTPBasicAuthenticationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.httpbasic.HTTPBasicAuthorizationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.utils.HTTPVisitor=DEBUG
log4j.logger.com.google.gsa.valve.modules.utils.HTTPAuthZProcessor=DEBUG
```

This configuration just output the logging data at the following file:

```
$TOMCAT_HOME/logs/valve.log
```

## Integration Frontends

This security framework can be integrated with the search appliance using two different interfaces supported by the GSA: **Forms Based (SSO) and SAML**. Both interfaces can not be used at the same time and that's why it's important to decide which one you're going to use on production. Apart from that, you also have to decide which authentication mechanisms will be implemented to collect user credentials, that could be for example a username/password based login form, silent Kerberos authentication or a mixed environment. Have a look at the scenarios guide on how to implement them.

There are differences when using the Forms Based or the SAML frontend, so that it's important to take the right decision on which one you'd like to implement. Here is a list of some considerations that can help you on taking the best interface for your environment:

- The Forms Based interface requires rewriting URLs as this will force any secure content indexed in the appliance to be served completely securely with SSO experience. It makes the crawling process a little bit more complex as any URL has to be injected in the appliance using the Forms Based URL format (see the Introduction guide for more information). As a best practice when using Forms Based interface, it's recommended to modify the appliance's XSLT frontend using JavaScript functions that let the users see the secure URLs in the search result not using the Valve's adapted format and let them to always access thru the original URL. The SAML interface does not require using these rewritten URLs though that makes both the crawling and serving much simpler.
- If you want to use the Forms Based frontend and want the Valve Security Framework to crawl content so that the URLs can be rewritten, you have to take into account that both the serving and crawling process will go through the same application server instances that have to be sized accordingly.
- As the SAML frontend does not use rewritten URLs you can use the native crawl features implemented in the appliance to let the search infrastructure to get the content for being indexed. You also have the option to use third party tools like for example the open-sourced GSA Crawl Proxy available at <http://code.google.com/p/gsa-crawl-proxy/>.
- SAML is very tightly coupled with the appliance. During the authentication and authorization process, this standard lets both the appliance and the SAML provider, in this case the Valve Security Framework, to exchange XML standard messages that drive the security lifecycle. If any issue arises, this process would help the Google Support engineers to offer you the best service as it can help them on knowing where the problem could be.
- If you need high availability in the whole search infrastructure including more than an application server instance with a load balancer in the front, the Forms Based interface should be used as the Valve's SAML one does not offer such support as of today.
- If you want to re-authenticate your users from time to time, for example at least once per day, the SAML interface offers a better solution as this is already implemented in the standard by default.

Our recommendation is using the SAML interface whenever it'd be possible, as the integration is more standard, but considering the above points and your own corporate scenario that can drive you to select the Forms Based frontend instead.

## Valve Configuration

Security Framework configuration is driven by the XML file mentioned above. It makes the overall configuration much easier as this component has been designed to support multiple flexible environments.

This configuration file has two main sections. The first one includes all the general configuration parameters for the entire application. This should be the first attributes configured in the XML file. The second one is the list of repositories the secure proxy application integrates in the search environment. Repositories are represented by the `<Repository>` tag.

A sample configuration file looks like as the one below. In this example we have taken the domain “google.com” as the sample corporate one so just substitute it with your own domain. As you can see the `loginUrl` parameter shown here contains a server name (`valveserver.google.com`) or the `searchHost` parameters also include the hosts `search01.google.com` or `search02.google.com` that do not correspond to any online Google service but it’s just an example of a search infrastructure enterprise deployment and points to the exact location of both the GSA Security Framework (`loginUrl` parameter) and the Google Search Appliances (`searchHost` parameters):

```
<?xml version="1.0"?>
<GSAValveConfiguration xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation="file:///C:/xmldocs/gsaValveConfiguration.xsd">

  <!-- FIRST SECTION: General configuration area -->
  <loginUrl>http://valveserver.google.com:8080/valve/login.jsp</loginUrl>
  <authCookieDomain>.google.com</authCookieDomain>
  <authenticationProcessImpl>com.google.gsa.valve.rootAuth.RootAuthenticationProcess
    </authenticationProcessImpl>
  <authenticateServletPath>/Authenticate</authenticateServletPath>
  <authorizationProcessImpl>com.google.gsa.valve.rootAuth.RootAuthorizationProcess
    </authorizationProcessImpl>
  <authCookiePath></authCookiePath>
  <authMaxAge>-1</authMaxAge>
  <authCookieName>gsaSSOCookie</authCookieName>
  <refererCookieName>gsaReferer</refererCookieName>
  <searchHost>http://search.google.com</searchHost>
  <maxConnectionsPerHost>10</maxConnectionsPerHost>
  <maxTotalConnections>10</maxTotalConnections>
  <testFormsCrawlUrl>http://valveserver.google.com:8080/valve/test.html</testFormsCrawlUrl>
  <errorLocation>C:\\Tomcat\\webapps\\valve\\WEB-INF\\error</errorLocation>

  <kerberos isKerberos="false"
    isNegotiate="false"/>

  <sessions isSessionEnabled="false"/>

  <saml isSAML="false"/>

  <!-- SECOND SECTION: Repository definition area -->
  <repository id="root" pattern="valveserver.google.com"
    authN="com.google.gsa.valve.modules.ldap.LDAPUniqueCreds" authZ=""
    failureAllow="true" checkAuthN="true">
    <P N="ldapBaseuser" V="dc=enterprise,dc=google,dc=com"/>
    <P N="ldapHost" V="ldap://ad.google.com:389"/>
    <P N="ldapDomain" V="@enterprise.google.com"/>
    <P N="rdnAttr" V="cn"/>
  </repository>

  <repository id="HTTPBasicAuth" pattern="http://server.google.com:90"
    authN="com.google.gsa.valve.modules.httpbasic.HTTPBasicAuthenticationProcess"
    authZ="com.google.gsa.valve.modules.httpbasic.HTTPBasicAuthorizationProcess"
    failureAllow="true" checkAuthN="true">
    <P N="HTTPAuthPage" V="http://server.google.com:90"/>
  </repository>

</GSAValveConfiguration>
```

As you can see in the above sample configuration, in the first part of the XML file the general configuration parameters are defined that affect the entire Valve application. These parameters include information among others like the authentication cookie name, the login page URL or how many HTTP connections are open per repository.

In the second sections is where the repository definitions are. In this case, we have a main authentication against the corporate Active Directory instance and we have another repository protected by HTTP Basic. It means that the Valve during the authentication process checks user credentials against such directory and if those are correct, then create HTTP Basic authentication (`authN` attribute) against such a server that will be reused when authorizing. In that case, whenever users are trying to access a document that fits with the pattern definition, the Valve uses the HTTP Basic authorization class (the one that is defined in the `authZ` attribute).

This is just an example on how to define a Valve configuration file. We strongly recommend you to have a look at the Scenario Guide that explains how to configure the Valve in different situations.

Below you can find a brief description on every parameter grouped by the main configuration element:

Elements	Attributes	Description
<b>loginUrl</b>		This is the absolute Valve login URL. It could be either an HTML/JSP login page if the user is authenticated thru a login page ( <code>login.jsp</code> ), or any other authentication server component like the Kerberos servlet ( <code>kerberos</code> ).
<b>authCookieDomain</b>		Domain used for any cookies created by the valve. This must be in the same domain as the GSA
<b>authenticationProcessImpl</b>		It points to the main authentication class that manages the entire process. It's recommended to use the default one (the same used in the above example)
<b>authenticateServletPath</b>		Authentication servlet path
<b>authorizationProcessImpl</b>		The same as <code>authenticationProcessImpl</code> for authorization
<b>authCookiePath</b>		Cookie path used for cookies created by the Valve
<b>authMaxAge</b>		Default age set (seconds) on cookies created by the Valve
<b>authCookieName</b>		GSA Valve authentication cookie name
<b>refererCookieName</b>		GSA Valve referer cookie name
<b>searchHost</b>		Search host (i.e. Google Search Appliance) used with this implementation of the valve. You can only have one search host (if you have more they must be load balanced using a public name)
<b>maxConnectionsPerHost</b>		Maximum number of HTTP connections created for each repository. This parameter and the one below permit to control the scalability when accessing the repositories
<b>maxTotalConnections</b>		Maximum total number of HTTP connections
<b>testFormsCrawlUrl</b>		This is the test URL configured in the crawling rule definition in the GSA console
<b>errorLocation</b>		Absolute path where the customized error code pages are located

## GSA Valve Security Framework – Installation

<b>Kerberos</b>	isKerberos	Kerberos setting attribute [true false]. The default value is false.
	isNegotiate	If Kerberos is set, this parameter says if the user ticket authentication process is going to be silent (thru SPNEGO) or creating it on the fly with username and password [true false]
	krbini	It points to the Kerberos network configuration file: krb5.ini (Win) or krb5.conf (Linux/Unix)
	krbconfig	Java config file for Kerberos
	krbAdditionalAuthN	If isNegotiate is set to true, this parameter says if there is an additional authentication process using a form [true false]
	krbLoginUrl	Path to a login form when the previous parameter is “true”.
	krbUsrPwdCrawler	The Kerberos crawling process is going to be done thru a login form that creates the crawler Kerberos tickets using his username and password [true false] (Only Forms Based interface)
	krbUsrPwdCrawlerUrl	It points to the crawling form set in the previous parameter (Only Forms Based interface)
<b>Sessions</b>	isSessionEnabled	Session setting parameter [true false]
	sessionTimeout	Maximum user inactivity time in minutes (-1 if you want an infinite value). It's recommended to use “-1” as otherwise it'll have an impact on performance
	maxSessionAge	Default age (in minutes) set on sessions created by the Valve (-1 if you want an infinite value). If it's used, this value has to fit with the authMaxAge parameter as they have to be basically the same time but taking into account this parameter is in minutes whereas authMaxAge is in seconds.
	sessionCleanup	How long the session cleanup process is inactive (in minutes)
	sendCookies	Cookies created by the Valve could be sent back to the user browser [true] or keep them in the session [false]
<b>saml</b>	isSAML	Activates SAML interface [true false]. If false Forms Based interface will be used (default)
	maxArtifactAge	Maximum age the saml artifact will be alive (in seconds). This is just used during authentication
	samlTimeout	Sets when the SAML will expire (in minutes)
<b>repository</b>	id	Repository identification key. This id should be unique per repository
	pattern	URL pattern for identifying the sources protected by an AuthN/AuthZ module
	authN	It points to the class that processes the authentication flow for this repository
	authZ	Java class that controls authorization for this repository
	failureAllow	If an authentication error is coming from this repository it can be obviated [true false]

	checkAuthN	If this is true, this repository will be evaluated during authentication, so it means the authentication class defined in “authN” parameter will be executed. Set it to “false” if you don’t want it to happen [true false]
--	------------	---

Repositories can be configured using additional parameters in the Valve configuration file. Those are really dependent on the kinds of authentication and authorization classes and are represented by the `<P>` tag the following way:

```
<P N="Parameter_Name" V="Parameter_Value"/>
```

You can find more details on the repository parameters needed for each main AuthN/AuthZ module at the Valve’s Deployment Guide, where there is more specific information on other configuration options like sessions or Kerberos. There is a Kerberos guide as well to help you in the case you want to deploy this authentication solution.

## Customize Error Messages

HTTP standard error messages can be customized in the Security Framework. It has its own error management component that internally handles the error shown to the user and can be used in combination with the standard Servlet/JSP error. Here you are the main steps to customize your error messages:

1. Create your error HTML pages in one specific location that by default is `<TOMCAT_HOME>/webapps/valve/WEB-INF/error`. The error files should include as the name a valid error code number (f.e.: 401) and some file extension (f.e.: .html). Some examples are 401.html, 403.html or 500.html.
2. The standard Servlet/JSP error management is declarative defined in web.xml. You should do the same so that the GSA Valve, as any other Java application, can read it and adapt your error messages, excepting the error code 401 (unauthorized) as the Valve uses it during Kerberos negotiation can completely change the way this security framework behaves. Here you are an example on how to define the error message for Internal Server Error (500) in application’s web.xml file:

```
<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/error/500.html</location>
</error-page>
```

3. You should also indicate to the GSA Valve Security Framework where those customized error files are located as the unauthorized (401) error message has to be treated by the application itself. You can do it using the `errorLocation` tag pointing it to the absolute error directory where your files are located.

## Multiple Credentials and Credential Identification

As it's mentioned in some GSA Valve Security Framework documents, we can use more than one credential per user. This is especially useful in those situations where users don't have unique credentials and can potentially have multiple different credentials in corporate applications. A valid Security Framework credential could be not only a username and password that identify a user, but also whatever any other security information that uniquely identifies him/her like for example a Kerberos ticket.

The way it works is based on getting user credentials during the authentication process. This is usually done by login page that collects username and password that are the main credentials for the user, but could be done by any authentication mechanism as for example the Valve's Kerberos implementation can silently authenticate users without a login form. Even we can have a scenario where during the authentication we can collect both the main username and password from a login page and his/her Kerberos ticket based on a negotiation process completely transparent for the user, only prompting him/her once (see Scenario Guide on how it's can be implemented). In this case we got a couple of credentials during the interaction with the user, main username/password and Kerberos ticket, but imagine that we need more application-specific username and passwords that can not be recovered during such authentication process. This case can be implemented as well as any Valve's authentication module is able to populate new credentials for each user taking them from wherever they'd be following the same approach of Identity Management solutions. For example, if the users are authenticated using Forms-based login page, we can get those main credentials to authenticate the user against a central LDAP directory and get there additional credentials for third party applications (for example Documentum), reusing them in order to authenticate users using the configuration information for those applications. This is the way for example the internal LDAPSSO module works as it gets some new credentials from standard LDAP attributes. You can find more information on how this module works in the Scenario Guide, but here you are an example on how you can configure it:

```
<repository id="root"
  pattern="perth.enterprise.lon.corp.google.com"
  authN="com.google.gsa.valve.modules.ldap.LDAPSSO" authZ=""
  failureAllow="true">

  <!-- LDAP Configuration Part -->
  <P N="ldapBaseuser"
    V="dc=enterprise,dc=google,dc=com"/>
  <P N="ldapHost" V="ldap://ad.google.com:389"/>
  <P N="ldapDomain" V="@enterprise.google.com"/>
  <P N="rdnAttr" V="cn"/>

  <!-- Multi-Credentials Configuration Part -->
  <P N="id1" V="Documentum"/>
  <P N="username1" V="usernameDCTM"/>
  <P N="password1" V="passwordDCTM"/>
  <P N="id2" V="Notes"/>
  <P N="username2" V="usernameNOTES"/>
  <P N="password2" V="passwordNOTES"/>

</repository>
```

In the above sample configuration, it's been defined a new LDAP connection that not only does a root authentication (have a look below at what "root" identity means), but also creates a couple of user credentials taking them from the LDAP server. The following credentials ids are created:

- Documentum: this credential is populated with the information stored on the usernameDCTM and passwordDCTM attributes at the user entry that fits with the root credentials sent by the user in the login form.
- Notes: the same as above with usernameNOTES and passwordNOTES attributes

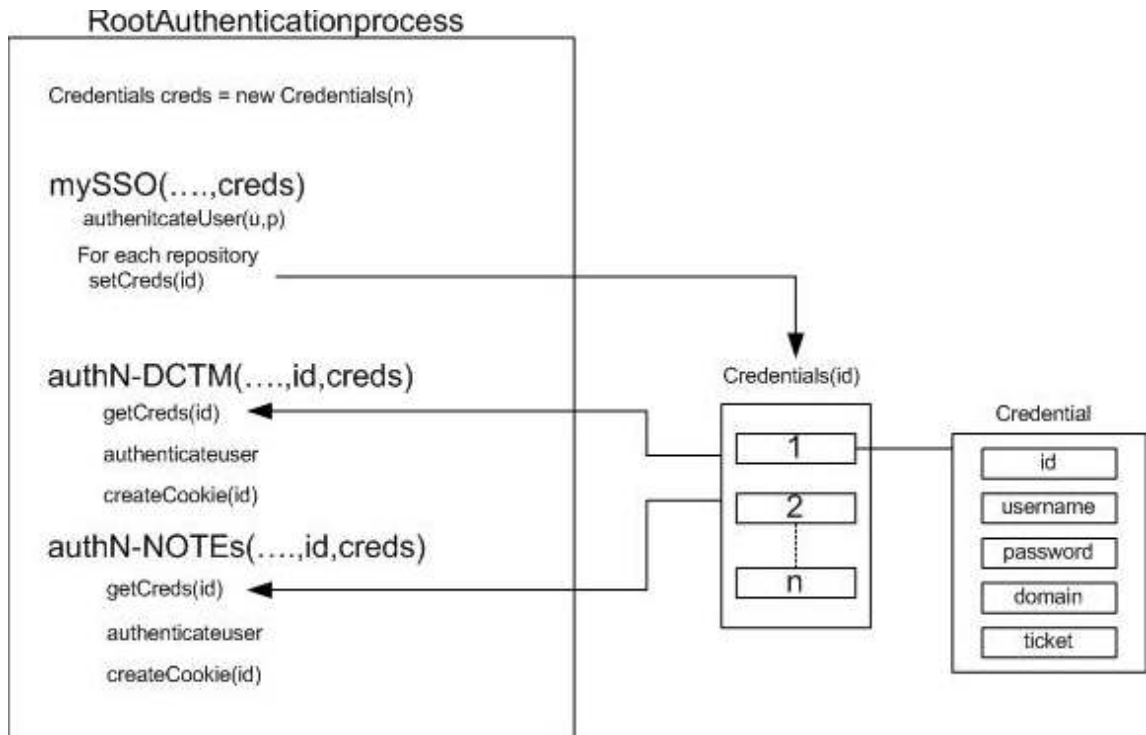
These new credentials are added to the main one (root) provided by the user during the authentication process and they can be used by the next repositories configured in the XML file. They are kept in the internal Credentials object, a credential list sent all over the Valve's authentication repositories. There is a link between such a credential id and the repository id as it's explained in the following identification rules:

- If you are using forms-based authentication through a login page, the username and password collected are uniquely internally identify as "root".
- In the case Kerberos authentication is set, such Kerberos user ticket is kept in the "krb5" credential.
- As mentioned before, any Valve's authentication module can add its own additional credentials that are going to be added to the previous ones in case they were created based on the configuration. Those credentials can be just used by the next repositories in the configuration file.
- A repository can make use of any credential just getting it from the Valve's credential list being referenced by its name. In order to make it as flexible as possible, the authentication modules have been defined in such a way that work the following way:
  - By default the authentication module tries to get a credential id that exactly fits with its repository id.
  - If there is not such a credential id, it just gets the "root" ones

For instance, based on the above Documentum and Notes example, these repositories in order to make use of such credentials have to be defined the following way:

```
...  
<repository id="Documentum" .../>  
<repository id="Notes" .../>  
...
```

This internal processing could be represented the following way:



- The above rule does not apply in the case of those authentication mechanisms that are not using username/password based credential like for example KerberosAuthenticationProcess. In this case, such process just looks for the “krb5” credential that contains the user ticket.

Regarding the repository identification, you have to take into account the next important rules:

- You should use “root” repository id just as the first repository definition only if your authentication process has to initially check credentials against a main repository. This is usually the case of a corporate LDAP server that holds user credentials as you can see in the above sample. There is no need to create such a “root” repository if you are not using this main authentication mechanism approach. Root authentication is managed the following way:
  - If the authentication is successful, the authentication process continues with the next repositories
  - If it fails, the authentication process stops here and sends the authentication error back to the user
- Do not use the word “krb5” as a repository identity

## Crawling

Another important thing to plan is the way the documents will be available in the appliance to be indexed. Secure content can be crawled in multiple ways but the best option depends on the security interface that each content server provides.

As it's explained in the Valve's Introduction guide, the URIs are different if we are using the SAML frontend or the Forms Based one. The **Forms Based interface** requires the appliance to go through the Valve security server to check authorization so that the URLs are always proxied through this server. A valid URL here could be something like <http://valveserver/valve/login.jsp?returnPath=http://contentserver/doc1.html>. It requires crawling URLs using the same pattern and that's why internal links in each document

are also rewritten. If you plan to use this frontend you have to take this into account, and for example the Valve is able to automatically crawl this way.

The **SAML interface** though does not need to rewrite URLs as the Authentication/Authorization SPI architecture is natively centralized. In this case you can use the out-of-the-box crawling features offered by the appliance, as it's able to securely crawl using the following authentication mechanisms:

- **HTTP Basic/NTLM authentication:** if your backend content systems either support HTTP Basic or NTLM (Windows native authentication), you can create crawling rules to let the appliance access securely the documents.
- **Cookie sites:** if the access to your content servers is protected by cookie, you can define cookie rules that authenticate the crawler against some URL patterns. If any URL matches with any of these rules, the appliance will send cookies back to the server where documents are.
- **Forms based authentication:** if you have a Single Sign-On system like CA Siteminder (aka Netegrity) or Oracle Access Manager (aka Oblix) protecting applications, a Forms Based crawling rule can be used. The appliance will recover the content if the right credentials are provided exactly the same way as a user is accessing those SSO protected applications. As these systems are driven by a cookie, the appliance can manage that SSO authentication cookie to know if the user is already authenticated or not. As of today the appliance only supports one Forms Based authentication crawling rule.

Another possibility here when the SAML interface is in place, is making use of another open source project available at [code.google.com](http://code.google.com) that acts as a Proxy server. Its name is GSA Crawl Proxy and it's available at <http://code.google.com>. This component serves the content securely protecting the access thru HTTP Basic. As this is a proxy, the appliance will authenticate against it using HTTP Basic credentials that are reused then to access to securely access to any backend content server whatever authentication mechanism this would be using. This way, for example, we can crawl two systems that are using different security mechanisms at the same time. The same philosophy behind the Valve Security Framework resides in this Proxy; in fact they share the same architecture as the Proxy is making use of the Valve Security Framework libraries and configuration. This way you can serve using the Valve and the crawling process can be done this component, sharing the same configuration.

## Testing

Once you have configured your Valve application properly you should check this is fully working. Start your Tomcat instance and check the Valve.log, the following output should be displayed only in the case of using the Forms interface:

```
23 ene 2008 17:23:15,378 DEBUG main com.google.gsa.Valve - Valve loaded
(com.google.gsa.Valve/1.4)
23 ene 2008 17:23:15,378 DEBUG main com.google.gsa.Valve - Loading configuration from
C:\Program Files\Tomcat 5.5\common\classes\gsaValveConfig.xml
23 ene 2008 17:23:15,378 DEBUG main
com.google.gsa.valve.configuration.ValveConfigurationDigester - Initilise valve
configuration digester
23 ene 2008 17:23:15,378 DEBUG main com.google.gsa.Valve - Configuration
```

The above message is created by the Valve filter. Note that when the SAML interface is in place you do not see this log as it's not making use of that filter. The log file will be empty till you will make the first request with the proper debug level configured.

## Forms Based Interface

In order to test the Forms Based interface is working, if this is the one you initially chose, open a browser and then try the Valve's test page at:

```
http://<apache_tomcat_server>:<port>/valve/test.html
```

It should redirect to wherever the `loginUrl` configuration parameter points to. For example:

```
http://<apache_tomcat_server>:<port>/valve/login.jsp
```

Once the authentication process is done successfully, you should return to the initial test page and see the message:

### GSA Authentication rule successful

Also check that the Valve's authentication cookie (by default its name is `gsaSSOCookie`) and the other ones used by the authentication modules configured are created.

You should then test the access to each repository defined in the Valve just testing some URLs. Test both with documents a user should be able to see and other he is restricted to access to. If you have for example one repository's document located at [http://repository\\_server/document1.doc](http://repository_server/document1.doc), try to access to it the following way:

```
<Valve_Login_URL>?returnPath=http://repository_server/document1.doc
```

Where `<Valve_Login_URL>` is just the value of the `loginUrl` parameter. If you are privileged to see the content you'll receive it, if not you will just get a 401 (unauthorized) HTTP error code.

## SAML Interface

The best way to test your configuration with the framework's SAML interface is perfectly working, is directly integrating with the appliance due to the several interaction it has between the different components. Set up the GSA integration through SAML (Serving - > Access Control) the same way as it's explained below. You should configure the logging in a very verbose mode (see the logging chapter and use there `DEBUG` or `TRACE` options) during the testing and check there how your modules are doing.

## Configuring the GSA

The final configuration step is the integration with the Google Search Appliance. Once you checked your whole Valve instance is working properly, you have to access to your GSA web console located at:

```
https://<GSA_Fully_Qualified_Name>:8443/EnterpriseController
```

The way the appliance is configured here is different if we are using either SAML or Forms Based interface. Even the configuration step could be slightly different depending on the crawling strategy you'd have chosen based on what was explained in the Crawling chapter.

## Forms Based Interface

If you want the GSA Valve Security Framework to crawl the repository you setup, the first thing you should do is add the URLs. Go to Crawl and Index → Crawl URLs and setup there the proper URL patterns you want to be crawled. Bear in mind URLs look like differently when Valve's Forms Based is in place and its crawling format should be the same as the way we previously tested, as you can see in the following example:

Start Crawling from the Following URLs: \* [\(Help\)](#)

```
#First repository
http://valve.google.com:8080/valve/login.jsp?returnPath=http://
/server01.google.com:82/documents/

#Second repository
http://valve.google.com:8080/valve/login.jsp?returnPath=http://
/server02.google.com/docs/
```

example: <http://www.myorganization.mycompany.com>

\*required

Follow and Crawl Only URLs with the Following Patterns: \* [\(Help - Test these patterns\)](#)

```
http://valve.google.com:8080/valve/
```

example: [mycompany.com/](http://mycompany.com/)

\*required

You can use the GSA features for crawling and taking an advantage of the SSO approach this Security Framework offers. This crawling integration could be configure in the admin console at Crawl and Index → Forms Authentication. In “*Sample Forms Authentication protected URL*” enter the URL to the test.html page on the server running the Valve (f.e: <http://valve.google.com:8080/valve/test.html>). In “*URL Pattern for this rule*” enter the base URL (f.e: <http://valve.google.com:8080/valve/>) for all requests that will use this rule. If you are using complex scenarios with Kerberos, please review how to configure such a rule at the Deployment Scenario Guide.

## GSA Valve Security Framework – Installation

### Create New Forms Authentication Rule: [\(Help\)](#)

To create a new Forms Authentication rule, enter a sample URL for the restricted content and the URL pattern. When the crawler finds the specified URL pattern, the rule is applied. After creating this rule, you can add more patterns that are sharing this rule.

Sample Forms Authentication protected URL:

(Example:

URL pattern for this rule:

(Example:

**Click Create a New Forms Authentication Rule to display a new browser window containing your login page. To properly set up a Forms Authentication rule, submit the form on the login page before returning to this page.**

Press 'Create a New Forms Authentication Rule'. A new window will pop up a window containing the login page that is part of the Valve. Below you have a sample login process with a customized Valve's login page as it's embedded in such new window:

Complete the login form by logging in to the page you want to access. Then click Save Forms Authentication Rule and Close Window button to return to the Admin Console.

[How does this work?](#)

©2002-2006 Google - [About](#)



Username	<input type="text" value="googleadmin"/>
Password	<input type="password" value=""/>
<input type="button" value="Enter"/>	

Finally save Forms Authentication Rule and Close Window

http://appliance:8080/valve/test.html

**Google** Google Search Appliance > Forms Authentication Login Wizard

Complete the login form by logging in to the page you want to access. Then click Save Forms Authentication Rule and Close Window button to return to the Admin Console.

[How does this work?](#)

©2002-2006 Google - [About](#)

GSA Authentication rule successful

The second part is to configure the GSA to use the Valve when performing searches. Go to Serving → Forms Authentication from the Admin interface.

Select there “*Always redirect to external login server*”, then enter the URL to the login page in “*URL:*” and the authentication cookie name (by default `gsaSSOCookie`) in “*Forms Authentication cookie name.*” as follows:

## Serve Protected Sites: [\(Help\)](#)

To serve pages that are secured behind Forms Authentication access control, select the option that your system uses. Then enter either the sample protected URL or the external login server URL, and the Forms Authentication cookie name.

- Login against a sample protected URL  
 Only User Impersonation    User impersonated cookie duration:
- Always redirect to external login server

URL:

Forms Authentication cookie name:

## Serving Logs [\(Help\)](#)

These logs show HTTP and HTTPS headers from the Forms Authentication process.

Logging is disabled.

©2002-2006 Google - [About](#)

When a user performs a search of secure content, the GSA will redirect to this login page if the specified cookie does not exist. Once the user is authenticated in the Valve, he/she will inherit the authorization permissions in all the backend repositories.

## SAML Interface

The Valve Security Framework has to be configured in such a way that accepts SAML requests. Apart from the other configuration, this is driven by the following lines in the config file:

```
<saml     isSAML="true"
          maxArtifactAge="60"
          samlTimeout="360" />
```

You have also to configure the crawling process. As you can have here multiple options we'll assume all your content repositories are accessible either using HTTP Basic or NTLM. If this is the case, you can add the needed crawler access rules at Crawl and Index -> Crawler Access, for example the following way:

## Users and Passwords for Crawling: [\(Help\)](#)

To allow the appliance to crawl web servers protected by user authentication, add a username and password to the HTTP header of each request. Specify a domain only if needed (typically when crawling Microsoft IIS web servers).

Crawl Access configuration saved

For URLs Matching Pattern, Use:	Username:	In Domain:	Password:	Confirm Password:	Make Public:	Position:
<input type="text" value="http://windows.google.com:81/"/>	<input type="text" value="clazaro"/>	<input type="text" value="WINDOWS"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>	<a href="#">Move Down</a>
<input type="text" value="http://linux.google.com:90/"/>	<input type="text" value="clazaro"/>	<input type="text"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>	<a href="#">Move Up</a> <a href="#">Move Down</a>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>	<a href="#">Move Up</a> <a href="#">Move Down</a>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>	<a href="#">Move Up</a>

\* Stored passwords are not displayed on these entries

In this case we have just configured two crawling rules, the first one will access to the content against a Windows server using NTLM authentication and the second one is using HTTP Basic to access to a web application running on a Linux host. These authentication rules have to be used in combination with the correspondent Crawl Rules definitions at Crawling and Index -> Crawl URLs as follows:

## Start Crawling from the Following URLs: \* [\(Help\)](#)

```
http://windows.google.com:81/  
http://linux.google.com:90/
```

example: <http://www.myorganization.mycompany.com/>

\*required

## Follow and Crawl Only URLs with the Following Patterns: \* [\(Help - Test these patterns\)](#)

```
http://windows.google.com:81/  
http://linux.google.com:90/
```

example: [mycompany.com/](http://mycompany.com/)

\*required

These are just some fictitious samples that simulate corporate sites inside the firewall, but they don't have anything to do with an online service provided by Google. The same applies for the Valve's SAML interface integration that is the next configuration step. In the next picture it is represented the integration with the appliance.

This is done at Serving -> Access Control. There you have to specify first the Authentication SPI URLs and then the Authorization SPI.

In some versions of the appliance you have an option named "Disable prompt for Basic authentication or NTLM authentication". In this example, as we are using both HTTP Basic and NTLM we could have an annoying popup window when serving content that can be easily avoided just checking that option.

### Authentication SPI [\(Help\)](#)

If you used the Google Search Authentication SPI to create an Access Connector between your authentication server and the appliance, enter the login URL for authenticating users and the artifact service URL that will identify the user to the appliance.

User Login URL:

**Warning - using http instead of https URLs is insecure and should normally only be used in testing.**

Artifact Service URL:

**Warning - using http instead of https URLs is insecure and should normally only be used in testing.**

Be sure to configure the Authorization SPI, below, to use Authentication SPI.

Session cookie timeout, in minutes:

### Authorization SPI [\(Help\)](#)

If using an authorization service implemented through the Google Search Authorization SPI, enter the service URL in this box.

**Warning - using http instead of https URLs is insecure and should normally only be used in testing.**

Authorization service URL:

## GSA Valve Security Framework – Installation

This integration has been done using the default values of the Valve Security Framework frontends. Adapt these URLs to the real ones used in your environment.

You can find more information on the other guides about how specifically do deployments either using the SAML or the Forms Based interface.