

Crawl Proxy – Installation and Configuration Guide

Google Enterprise EMEA

Google Search Appliance is able to natively crawl secure content coming from multiple sources using for instance the following main methods:

- **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.

The above offering is useful in most cases but in some others we would need to use an extended functionality. This is usually occurring when we have some content sources that are not able to manage any of the above security mechanisms or we have more than one Forms based authentication solution, not being protected by a central SSO server.

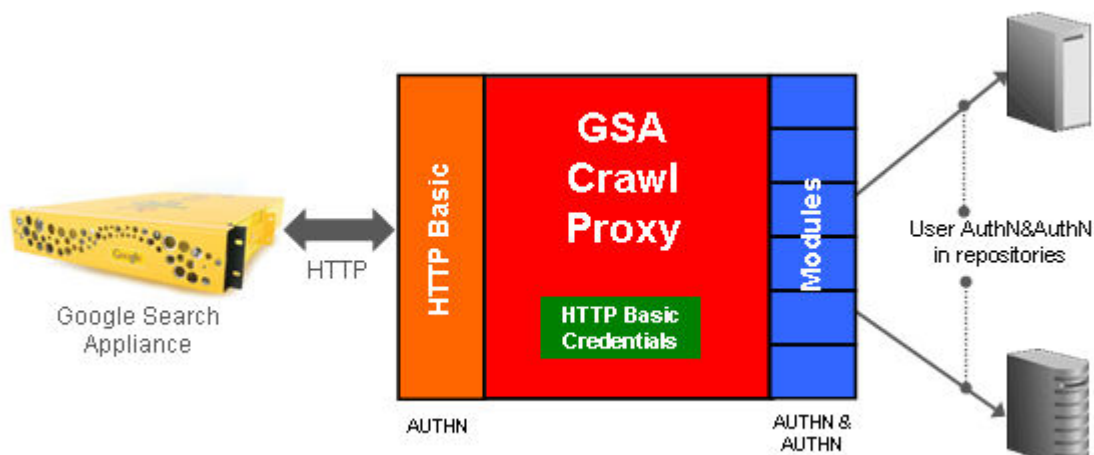
GSA Crawl Proxy, this open source project, is meant to extend the crawling possibilities taking the advantage of the connectivity provided by the Authentication/Authorization modules for the GSA Valve Security Framework <<http://code.google.com/p/gsa-valve-security-framework/>>. These modules implement the integration complexity to securely access to the content sources where documents are. The GSA Crawl Proxy is able to authenticate the crawler user coming from the search appliance using HTTP Basic, and send those credentials to any of these modules that can convert them to any security mechanism the content servers would understand.

It's particularly useful if you are using that security framework and you want to crawl through this utility, as the configuration files and security modules would be the same. This utility was born as a consequence of the need to crawl content from multiple sources when using the GSA Valve Security Framework, as it supports serving securely through multiple content sources but lacks a crawling functionality when using SAML as a secure frontend. SAML <<http://www.oasis-open.org/committees/security/>> is a security standard that can control both the authentication and the authorization when accessing services. Although you were not either using that security framework or SAML, you can get the advantage to use this crawling tool.

The integration between the GSA and this crawling tool is done configuring a proxy access in the appliance. This application is acting as a proxy that gets the crawling requests from the appliance and returns back the result, either the document's content if the request was successful or an HTTP error code if does not.

How it works

The architecture is fairly simple as this utility acts as a proxy between the search appliance and the content source. The GSA's crawler accesses the remote content through this proxy application that serves it securely in a completely transparent way to the appliance. This is represented in the following picture:



This middleware is located in between the appliance(s) and the backend sources we want to crawl content from. The initial **authentication** is done by HTTP Basic, as the appliance sends the encoded credentials in the HTTP header, and the Crawl Proxy authenticates the user with the corporate policy. For instance, during that authentication process we can validate the crawler credentials against the corporate LDAP server and starts the authentication against the content sources. At that time we can get additional credentials, like for example a Kerberos ticket, and access using them back to the content servers using the security technology they work with, for instance: HTTP Basic, NTLM, Kerberos, ACLs, Database security, custom protocol, etc. The authentication process happens once or, at least, whenever we decide to invalidate the internal session, for example every 30 minutes, where the authentication information is kept. All these functionalities are obtained from the GSA Valve Security framework modules that provide both connectivity and security when accessing these systems. You have some of these modules already provided by this framework; some others are provided by third party people and are available at the project's web site; and you always have to option to build your own ones. It's recommended to take a look at this security framework's documents to see the possibilities around it.

Those credentials obtained during the authentication process are kept in the internal session that can be reused as many times as needed for **authorization**, a process that is executed whenever the appliance is crawling a document. Apart from these credentials, we can keep there the cookies created by the backend servers during the authentication process and reuse them during the authorization if these systems are

cookie-driven applications. This way we can crawl different application protected by different Single Sign-On (SSO) or, at least, they have distinct security modules that manage authentication cookies separately.

Installation

Crawl Proxy is a Java based application that runs on Tomcat application server. As this is purely standard application, it can runs on any standard J2EE compliant server but it has not been tested.

The application contains both a Jar deployment file (`crawlproxy.jar`) 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 this application makes use of them. And some other Google libraries that work exactly the same way.

Requirements

The initial requirements for the GSA Crawl Proxy 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 1.5.X or upwards. It's recommended JDK 1.6.X in case you are using Kerberos/SPNEGO features
- 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

If you want to build the application from scratch, you can do it very easily using any third party Java deployment tool. It's just a simple web application that uses a web.xml file, the applications classes and the external libraries. All of them can be found at the application sources in the downloading area of the project's web site.

Libraries

The Java application 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 application's 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 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/>>
 - *Apache Commons Configuration* (1.5) :
<<http://commons.apache.org/configuration/>>
 - *Apache Commons Lang* (2.4) : <<http://commons.apache.org/lang/>>
 - *Html Parser* (1.6) <<http://htmlparser.sourceforge.net/>>
 - *JDom* (1.1) <www.jdom.org>
 - *Servlet API* (servlet-api.jar)
- **Google libraries:** they offer some functionality for this application. Most of them are part of the GSA Valve Security Framework or some other applications that have been independently packed for better usability:
 - *GSA Valve Security Framework:* it contains all the security framework classes <valve_extlib.jar or valve_2.0_extlib.jar>
 - *GSA Kerberos libraries:* Kerberos implementation <ValveKrb5.jar>
 - *GSA Valve Session libraries:* contains the session managements implementation <ValveSessions.jar>
 - *Sniffer* (1.3 or onwards): Http libraries that act as a standard browser <Sniffer.jar or Sniffer_1.3.jar>
- The above packages are available in the following Google project sites:
- *GSA Valve Security Framework:* <<http://code.google.com/p/gsa-valve-security-framework/>>
 - *GSA HTTP Sniffer:* <<http://code.google.com/p/gsa-http-sniffer/>>

Application Deployment

The application is deployed at the root level of the application server. For example in Tomcat this is the “ROOT” web application located at \$TOMCAT_HOME\webapps\ROOT directory. In other application servers, the name of this root application may vary (for instance, in Oracle Application Server-OC4J its name is “default-web-app”). The reason why it’s needed to be deployed at that level is, as this application is acting as a proxy, the appliance sends the request to the server name and port the application server is running, with no other parameters. So, it’s required the proxy application to treat requests coming from the appliance at the root level of the server.

If you didn’t build the application by your own, you can find a Java WAR deployment file (CrawlProxy_XX.jar where XX is the version number) that contains the files needed to deploy it. It contains the standard WEB-INF directory with the following structure:

- *web.xml:* this is the web application configuration file that contains all the information needed to run the application like for example the Java Filter. It’s also used to populate some internal variables as we’ll see later on.
- *classes:* this directory contains all the compiled application classes.

Follow the next steps to successfully deploy the application on Tomcat:

1. **Install the application at ROOT webapps directory.** It’s recommended to backup the existing deployed root application before proceeding. The fastest way to install it is as follows:
 - Unzip the WAR file
 - Copy its content at \$TOMCAT_HOME\webapps\ROOT directory

2. **Install libraries.** The libraries the application is using have to be available at the application’s “lib” directory located at \$TOMCAT_HOME\webapps\ROOT\lib. These libraries at least should be the following:

- Sniffer <sniffer.jar or sniffer_1.3.jar >
- GSA Valve Security Framework <valve_extlib.jar or valve_2.0_extlib.jar>
- GSA Valve Kerberos <ValveKrb5.jar>
- GSA 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>
- Apache Commons Configuration <commons-configuration-1.5.jar>
- Apache Commons Lang <commons-lang-2.4.jar>
- Html Parser <htmlparser.jar>
- Log4J <log4j-1.2.14.jar>

3. **Set up the configuration files.** The proxy application is using two different config files:

- *GSA Valve Security Framework configuration file (gsaValveConfig.xml):* this file should contain the repository information to access to your secure content servers. Each repository includes the needed authentication and authorization Java classes that fit with the security access technology that it’s enabled there. Have a look at the Security Framework documents to see the config file format before proceeding. You will see some examples on how this file looks like later on.
- *GSA Crawl Proxy (crawlConfig.xml):* the application has a very simple config file that includes the basic parameters that are needed to have the application up and running. Here is a brief description on what each parameter means:

Element	Description
crawler.CredentialID	This is the credential ID used to reference the username and password obtained during the HTTP Basic authentication method. This is then sent back to the Valve’s repositories to process the authentication there. It’s recommended to set it to “ root ”, unless you would need a specific credential ID in your authentication/authorization modules.
crawler.Realm	This is the HTTP Basic realm the proxy is going to use when getting Basic credentials. This could either be the name of the Basic domain or a specific hostname.
gsa.ipAddress	It references the IP address(es) that can send request to this proxy. It protects then IP addresses out of the scope of the ones specify here to crawl content. You can either use individual or range IP addresses. Some valid examples are: “192.168.1.2” “192.168.1.2-192.168.1.4” “192.168.1.2\, 192.168.1.4” “192.168.1.2\, 192.168.1.4\, 192.168.1.8-192.168.1.12”

session.maxSessionAge	Once the user is successfully authenticated, all the information got during the authentication process is kept in a session, so that we can use during the authorization process. This parameter sets the number of minutes the crawl session is going to be valid. Once that time is reached, the session is deleted and a new one has to be issued.
------------------------------	---

Here it's how a common Crawl Proxy config file looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<proxy_configuration>
  <crawler>
    <credentialID>root</credentialID>
    <realm>corporate.company.com</realm>
  </crawler>
  <gsa>
    <ipAddress>192.168.1.2\,192.168.1.11-192.168.1.15
    </ipAddress>
  </gsa>
  <session>
    <maxSessionAge>20</maxSessionAge>
  </session>
</proxy_configuration>
```

4. **Set up the configuration file location:** the last step in the installation process is configuring the application to point to the configuration files. This is done in the application's *web.xml* file where two properties, *gsaValveConfigPath* and *crawlerConfigPath*, have to be configured there. The default *web.xml* file included in the deployment application includes them, so just change the path to the right locations in your environment.

```
<web-app ...>
...
<!-- Crawl Proxy Config Location -->
  <env-entry>
    <description>Crawl Proxy Config Path</description>
    <env-entry-name>crawlerConfigPath</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>
      C:\Program Files\Tomcat 5.5\common\classes\crawlConfig.xml
    </env-entry-value>
  </env-entry>
  <!-- Valve Config Location -->
  <env-entry>
    <description>GSA Valve Config Path</description>
    <env-entry-name>gsaValveConfigPath</env-entry-name>
    <env-entry-type>java.lang.String</env-entry-type>
    <env-entry-value>
      C:\Program Files\Tomcat 5.5\common\classes\gsaValveConfig.xml
    </env-entry-value>
  </env-entry>
...
</web-app>
```

In the case you'd like to use the GSA Valve Security Framework for serving content, the whole configuration will be much easier as the Crawl Proxy uses the same main configuration file when crawling. You just have to write one main config file (`gsaValveConfig.xml`) that can be used as many times as needed in all your GSA Crawl Proxy instances. If this is not the case, so you want to use this crawling application but not that security framework for serving, then you'd have to create a config file that includes all the content repositories to be accessed by the proxy, associated to the corresponding security modules. We strongly recommend you to have a look at the GSA Valve Security Framework where the config file format is deeply explained.

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/crawlproxy.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

#Crawl Proxy classes
log4j.logger.com.google.gsa.proxy.Crawler=DEBUG
log4j.logger.com.google.gsa.proxy.config.Config=DEBUG
log4j.logger.com.google.gsa.proxy.auth.CrawlingUtils=DEBUG
log4j.logger.com.google.gsa.proxy.auth.session.CrawlingSession=DEBUG
log4j.logger.com.google.gsa.proxy.auth.ipaddress.IPAddress=DEBUG
log4j.logger.com.google.gsa.proxy.auth.ipaddress.IPAddresses=DEBUG
log4j.logger.com.google.gsa.proxy.auth.ipaddress.IPAddressChecker=DEBUG

#Session
log4j.logger.com.google.gsa.sessions.UserIDEncoder=DEBUG
log4j.logger.com.google.gsa.sessions.SessionTimer=DEBUG
log4j.logger.com.google.gsa.sessions.Sessions=DEBUG

#Put here the AuthN/AuthZ modules you want to debug, for example:
log4j.logger.com.google.gsa.proxy.auth.ldap.Ldap=DEBUG
log4j.logger.com.google.gsa.proxy.auth.ldap.LDAPUniqueCreds=DEBUG
log4j.logger.com.google.gsa.valve.modules.httpbasic.HTTPBasicAuthenticationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.krb.KerberosAuthenticationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.krb.KerberosAuthorizationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.oracle.OraclePortalAuthenticationProcess=DEBUG
log4j.logger.com.google.gsa.valve.modules.oracle.OraclePortalAuthorizationProcess=DEBUG
```

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

```
$TOMCAT_HOME/logs/crawlproxy.log
```

Testing

Before integrating the Crawl Proxy with the appliance, you should make sure this application is perfectly working and integrated with your content sources. A possible way to test it is simulating HTTP connections against the proxy application using for example *telnet* connections.

Put in a text file a potential content request the crawler will make. You have to include information like:

- The URL and the hostname/IP address
- Crawler's username and password. The HTTP Basic format is "username:password", sent encoded in the Authorization header using Base64.
- User-Agent has to be "gsa-crawler"

The following example simulates a request the crawler would make against the <http://contentserver.corp.com:90/1.html> URL using the credentials "crawler" as a user name and "google" as password. The result of encoding this username and password is "Y3Jhd2xlcjpb29nbGU=" that is set in the Authorization header.

```
GET http://contentserver.corp.com:90/1.html HTTP/1.1
User-Agent: gsa-crawler
Host: contentserver.corp.com:90
Connection: Keep-Alive
Authorization: Basic Y3Jhd2xlcjpb29nbGU=
```

This request has to be sent to the crawler using telnet. Let's say the proxy is located at `crawlproxy.corp.com`, listening on port 8080. So you have to execute:

```
telnet crawlproxy.corp.com 8080
```

Copy the request previously generated and pastes it in the telnet command window. Check both the result that will be presented in the terminal window and, if there is any problem, the Crawl Proxy's error log.

Google Search Appliance Integration

The next step is integrating the search appliance with the Crawl Proxy. This integration is done through Proxy Servers rules that let the appliance to redirect the requests to a proxy for those content sources we want to crawl securely. This way, the appliance will send an HTTP request to the proxy for every document, but containing information as it was directly sent to the final content source. So there is no need to rewrite URLs as the requests are processed in the proxy that provides the content in case the security rules permit it.

This can be configured in the GSA console at the “Crawl and Index” menu. Follow the next steps to set it up:

1. **Configure the HTTP Basic rules:** firstly it's needed to declare the HTTP Basic rules to configure the secure access to all the content sources you want the crawler to retrieve data from. Define the list of content sources you want this application to access to, following the next rules:
 - If a content server is protected by any of the native authentication mechanisms provided by the appliance, it's highly recommended to let the appliance crawl directly.
 - If you have more than one Forms Based enabled applications, decide the crawling policy here as the GSA can only crawl one independent Forms Based system, although it can protect multiple URL patterns, at the same time. The best way is to do so using the Crawl Proxy for all the independent Forms Based applications whenever it'd be possible.
 - Make sure you have a GSA Valve Security Framework's Authentication/Authorization classes ready for the security technology interface each application supports. If you don't have those classes ready, consider to create them “ad-hoc” as a custom Authentication/Authorization module.

For each content source that is included in the list of resources the Crawl Proxy will be accessing to, create an HTTP Basic Crawler Access rule in the GSA console. This is available at “Crawl and Index” -> “Crawl Access”. The next sample configuration shows here how it looks like:

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).

For URLs Matching Pattern, Use:	Username:	In Domain:	Password:	Confirm Password:	Make Public:
<input type="text" value="http://contentserver1.corp.com:90/"/>	<input type="text" value="crawler"/>	<input type="text"/>	<input type="password" value="....."/>	<input type="password" value="....."/>	<input type="checkbox"/>
<input type="text" value="http://contentserver2.corp.com/"/>	<input type="text" value="user1"/>	<input type="text"/>	<input type="password" value="....."/>	<input type="password" value="....."/>	<input type="checkbox"/>
<input type="text" value="http://contentserver3.corp.com/"/>	<input type="text" value="google"/>	<input type="text"/>	<input type="password" value="....."/>	<input type="password" value="....."/>	<input type="checkbox"/>
<input type="text" value="http://contentserver3.corp.com:83/"/>	<input type="text" value="google"/>	<input type="text"/>	<input type="password" value="....."/>	<input type="password" value="....."/>	<input type="checkbox"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="password"/>	<input type="password"/>	<input type="checkbox"/>

* Stored passwords are not displayed on these entries

[Add More Rows](#)

If you want all this content to be publicly available click on “Make Public”. If you want secure access when the appliance is serving content, you have to plan the

way this is going to be implemented in your environment either by using native implemented features in the appliance or using external tools like the GSA Valve Security Framework.

One major current caveat is HTTPS servers can not be used with the crawling application. It's intended to be implemented in the future, so stay tuned.

2. **Set up the Proxy Server rules:** in order to let the appliance always access to the content servers through the proxy, it's needed to configure the Proxy Server rules associated to the Crawl Access ones previously defined. This is done in the GSA console at "Crawl and Index" -> "Proxy Servers". Add there all the URL patterns that matches with the previously defined access rules, and associate them with the proxy location, for example:

Proxy Servers: ([Help](#))

Some web servers may deny direct access to the appliance. Enter URL patterns for those web servers, and the corresponding proxies and ports you want to use.

For URLs Matching Pattern:	Use This Proxy Server:	On Port:
<input type="text" value="http://contentserver1.corp.com:90/"/>	<input type="text" value="crawlproxy.corp.com"/>	<input type="text" value="8080"/>
<input type="text" value="http://contentserver2.corp.com/"/>	<input type="text" value="crawlproxy.corp.com"/>	<input type="text" value="8080"/>
<input type="text" value="http://contentserver3.corp.com/"/>	<input type="text" value="crawlproxy.corp.com"/>	<input type="text" value="8080"/>
<input type="text" value="http://contentserver3.corp.com:83/"/>	<input type="text" value="crawlproxy.corp.com"/>	<input type="text" value="8080"/>

In the above example, all the URL patterns are configured with the same Proxy Server that is located at *crawlproxy.corp.com:8080*, although you could have more than one situated in multiple locations.

3. **Add the URL patterns to Crawl URLs:** finally in order to have all these URLs available to be accessed by the appliance, your content source's URL patterns have to be added in the GSA's Crawl URLs. This is located in the appliance's console at "Crawl and Index" -> "Crawl URLs". For the same example we've been showing here, this configuration would be as follows:

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

```
http://contentserver1.corp.com:90/
http://contentserver2.corp.com/
http://contentserver3.corp.com/
http://contentserver3.corp.com:83/
```

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

*required

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

```
http://contentserver1.corp.com:90/
http://contentserver2.corp.com/
http://contentserver3.corp.com/
http://contentserver3.corp.com:83/|
```

example: mycompany.com/

*required

Deployment Options

In the example shown there is just one Crawl Proxy instance. A unique instance can handle more than one user credentials, so we can access to any of the backend repositories using some user credentials (username and password) that can be completely different than the credentials used with another URL pattern. Even we can have different Authentication/Authorization modules defined in the *gsaValveConfig.xml* file, so that we can crawl, with just one proxy instance, content that is coming from multiple secure sources that are using security mechanisms like HTTP Basic, Kerberos, Forms Based (SSO), etc.

You can have obviously more than one Crawl Proxy instance linked to the same search appliance just defining the integration rules appropriately. The reasons to do so, among others, could be:

- **Security:** in the case you'd like to get content from different sources with different security requirements. HTTP Basic is not a very secure protocol, so that it might be required accessing the content in different way depending on the requirements. The securest way to deploy the proxy application is setting it as closer as possible to the appliance.
- **Performance:** if just one application server is not able to perform well with a high number of repositories it might be useful to have more than one. If **high availability** is required here as well when crawling, a load balancer can be put in place in front of a Crawl Proxy server farm.
- **Global deployments:** another possible case is having multiple appliances spread out all around the globe in different datacenters, configuring one or more local Crawl Proxy instances associated to the closest appliance.