GSA | Notes from the Field

# Google Search Appliance

## Security

May 2014

# Security

Security is a key consideration when designing and implementing solutions that integrate data from different sources for enterprise search. This can be one of the most complex things to deal with in these projects, especially on the Intranet side, where security is usually a strong requirement. It's important to allocate enough quality time to this area.

This paper provides insights into considerations for modeling security requirements and transforming them into the ultimate solution. It's important to understand the project's needs from the beginning, because security is one of the areas that is more difficult to change in the project after you've started implementing new phases.

## About this document

The recommendations and information in this document were gathered through our work with a variety of clients and environments in the field. We thank our customers and partners for sharing their experiences and insights.

| | |
|---|---|
| **What's covered** | This paper reviews all the implementation options with the Google Search Appliance (GSA), to help you better understand the security protocols the product supports. This paper complements the GSA product documentation, so you will find references to the product documents, where you can find detailed descriptions about how to configure features. |
| **Primary audience** | This guide is intended for anyone who is involved in an enterprise search project who has to deal with security requirements either by collecting them, designing the project, or implementing the eventual solution. These are the roles in the project that can benefit from going through this paper:<br>● Project manager<br>● Technical project manager<br>● Developer<br>● GSA administrator |
| **IT environment** | GSA configured with various authentication and authorization mechanisms for secure search. |
| **Deployment phases** | Designing the security configuration for the GSA. |
| **Other resources** | ● GSA product documentation provides complete information about the GSA.<br>● Configure GSA Security on Help Center<br>● Learngsa.com provides educational resources for the GSA.<br>● Google for Work Support Portal provides access to Google support.<br>● GSA Notes from the Field: Introduction to Content Integration<br>● GSA Notes from the Field: Deployment Scenario Handbook |

**Contents**

# Chapter 1 Designing Security in the GSA

## Overview

Enterprise search projects integrate data from different sources to enable users to find information easily. In most cases, especially in intranet projects, access to documents in source applications is protected. To provide relevant and secure results to users, the corporate search engine must apply the same authorization policies as the sources where documents are stored.

The search appliance acts as a hub, where content coming from different sources is indexed to facilitate access to the information. The search appliance must rely on the same security protocols as those that the applications use. If your enterprise search project includes indexing protected content, you need to invest time during the design phase to model the security relationships between your content sources and the Google Search Appliance.

Before actual implementation of security in the GSA begins, take time to understand the overall integration scenario and reference architecture. Because there are probably internal security policies and protocols already established in your organization, you have to explore the best options for implementing security in the search environment. And you have to design a security model for the search appliance that will be consistent with all project phases.

This chapter explains the key processes of GSA's secure search, and how you should approach the overall design.

We can divide the secure search into three distinct but related processes:

| | |
|---|---|
| **Secure Content Acquisition** | The mechanism GSA uses to acquire the secured content source. GSA has to pass through the protection that the content source puts in place in order to gain access. It's an integral part of content acquisition, but it has to be considered as part of the security design. |
| **Serve Time Authentication** | The mechanism used by GSA to identify end users. It could be one or more of the Internet authentication protocols. Note that this is the communication between GSA and the client (browser). |
| **Serve Time Authorization** | The process that GSA employs to check whether a search user has access to the search results. |

**Content Acquisition**

Once you have modeled the information about your content sources, you can design the authentication mechanism(s) the GSA will use to integrate with each secure source. This is the process portion of the project design phase that models the integration between the search appliance and an organization's systems. The search appliance permits the use of several authentication mechanisms simultaneously to

accommodate different applications when acquiring contents. The process generally involves using a system or super user account with broad access to the content source so that all the documents can be indexed by the GSA.

**Serve time authentication**
Serve time authentication is the integration between the search appliance and the end user. It can be the same authentication protocol as used by one of the content sources, but it doesn't have to be. Sometimes multiple authentication protocols are required in order to support the authorization of different content sources. However, you should always ask yourself the following questions:

- What authentication protocols are available in the customer's environment?
- How can I minimize the authentication mechanisms used during serving? Can I reduce it to just one?
- How can I minimize the impact on end users? Can authentication be silent?

**Serve time authorization**
Each content source uses its own security policies and infrastructure to authorize access to its information. Based on the information you gathered about the content sources, you select the authorization mechanisms based on answers to the following questions:

- What authorization mechanisms are possible for the given content source?
- Which mechanism gives the best performance?
- What needs to be implemented for the content acquisition process in order to support this mechanism?

Although serve time authentication happens before authorization during serving, you should evaluate the authorization options FIRST. What is required for authorization generally decides what authentication mechanisms you should consider. In any case, these three processes are interrelated and you have to consider the implications of every decision.

## Information Gathering

Google recommends that you take the following actions during initial analysis:

- Clarify all requirements related to security, even potential future needs that are not currently part of the scope of the project, but might be considered for a future phase.
- If one of the requirements is silent authentication, make sure that it is feasible to provide it before committing to it.
- Identify the security mechanisms certified by your organization. Is there a Single Sign-On (SSO) system? Is Kerberos enabled?

Use the following table to model each content source. Include information about security in the Security Mechanisms field.

| | |
|---|---|
| **System Info** | Name of the system, and underlying product name |
| **Description** | More description |
| **Content Type** | For example, are they office documents, web pages? database records? |
| **Content Size** | Document count—smaller content size could mean that late binding for serve time authorization might be OK |
| **Serve time authentication** | Is the content server using Windows Integrated Authentication? |
| | Is the content server integrated with an SSO system? |
| | If the content server has its own user directory in either database or LDAP, are the user names synced with the company wide directory? |
| **Serve time authorization** | How responsive is the content server? If it's not responsive, late binding is probably out of the question. |
| | Are the permissions on the documents fairly open, or very restrictive? If very restrictive, later binding is probably out of the question. |
| | Is there an API that can tell whether a user has access to a list of documents given the user name and document IDs? If there is such an API, connector or SAML authorization will be possible |
| | Is there a way to find out what groups/roles/users have access to each document? If the answer is "Yes," then ACL authorization is most likely the preferred approach. |

## Content Acquisition

The acquisition generally comes in the following forms. Note that the authentication protocol used would have to be what's supported by the content source. However, the content acquisition would usually allow different authorization mechanisms to be used.

| | Possible authorization mechanisms for serving | Notes |
|---|---|---|
| **Direct Crawl** | ACL, Head Request, SAML Authorization | It may involve developing a custom proxy server for extra processing |
| **Feeds** | ACL, Head Request, SAML Authorization | A simple, custom one-off implementation |
| **Connectors** | ACL, Connector Authorization | It could either be an off the shelf connector, or a custom connector to be developed |

## Single vs. Multiple identities

By examining all the content sources, you should be able to answer a very important question: Is one (default) Credential Group enough? This determines the model for serve time authentication. When there are multiple identities per user, you probably need to define multiple Credential Groups. Different authentication protocols for different content sources do not mean multiple identities. For example, one content source could be using a forms based authentication while another uses Kerberos. However, if the same Active Directory is used as the user directory for both systems, there is only one identity per user. Only if user information is stored in different repositories, there might be multiple identities needed by GSA. Still, there are two exceptions:

- If one user directory is a replicate of another, there is still one identity per user. Hence one Credential Group is enough. For example, when Documentum is integrated with Active Directory, one approach is for all the users to be replicated in Documentum database.
- If there is a strict matching of user names from two user directories, one Credential Group is enough as long as you put in place a user identity translation service—maybe as part of the authorization process.

## Selecting an authorization mechanism

Serve time authentication and authorization are tightly connected. As mentioned previously, although serve time authentication happens before authorization during serving, you should evaluate the authorization options FIRST. This is a very important point worth repeating here. This chapter describes the connections between these two processes in details.

Authorization is always considered on a per content source basis. The purpose of authorization is to make sure that users can see what they are entitled to see in the search results. Besides this ultimate goal, the most important criteria in selecting which authorization mechanism to use is **performance**. It implies that:

- Search results need to come back as fast as possible to give the end users the best experience possible. Based on usability studies, if search is too slow, many people would simply give up and usage of search would decrease.
- Performance needs to be good enough so that relevant results will not be missing due to time outs. If the authorization decision times out on certain results, the results will have an indeterminate authorization decision, thus won't be displayed in the search results list.
- When late binding authorization is used, you need to minimize the performance impact on the content server.

For deployment projects, if there is an existing connector provided by either Google or one of Google's partners, the authorization is already decided for you by the design of the connector. You have to select an authorization mechanism only under these circumstances:

- There are multiple connectors offered by different parties and they use different authorization mechanism. There will be many factors in deciding which connector to use including costs, and authorization mechanism is only one of them.
- A connector sometimes supports multiple authorization mechanisms. For example, the Google Search Appliance Connector for SharePoint supports three mechanisms: Per-URL ACL, Connector, and Head Requests.
- When there is no existing connector, you have to develop custom code to integrate the secure content. This is when you have to consider all options.

Below we discuss the authorization in the order of performance preference. GSA processes authorization based on two main approaches:

- Early binding authorization
- Late binding authorization

Generally speaking, early binding speeds up the authorization process in the GSA compared to late binding, but it doesn't necessarily mean early binding should be the method always used for all content sources.

With early binding, authorization is fully managed by the search appliance itself. Early binding requires authorization rules to be known to GSA. It doesn't have to contact an external security component such as the content source at serve time to validate whether the user has the right to access a document.

The GSA supports the following two types of ACLs:

**Per-URL ACLs**
With Per-URL ACLs, each document in the index can have its own authorization rules. Adding a Per-URL ACL to a document can be done through Feeds, metadata in HTML body, or custom HTTP headers. Per-URL ACLs can include both users and groups. Per-URL ACL is generally preferred since it is much more scalable with the number of documents and offers better performance.

Considerations for using Per-URL ACLs:

- This approach is very useful when you have fine-grained authorization rules and you want to have quick authorization responses. Fast authorization with ACLs is critical for such GSA features as Dynamic Navigation, duplicate directory filtering, and Dynamic Result Clusters.

- This approach introduces some complexity into resolving group membership in the search appliance. This resolution can be managed by the GSA in some instances, for example, whether those groups are in an LDAP directory as Active Directory. You can also create your own custom processes to pass groups to the search appliance. In 7.2, an onboard Groups Database is introduced as a beta feature that offers even tighter integration.
    - The Google Search Appliance Connector for Active Directory Groups is provided for resolving groups from single or multiple Active Directory domains.

- There is also a delay between when a security setting is changed in the source platform and when the search appliance is notified of this.

- The maximum number of principals that can be attached to a document is configurable, with a default of 10,000. The maximum is 100,000.
    - The following worst-case scenario has been tested with good performance (sub-second) in regards to ACL filtering:
        - 10k URLs to be filtered
        - Each URL has 10k items in the ACL
        - Search user belongs to 1k groups, but doesn't have access to any documents so the GSA has to exhaustively filter every URL that matches the search term.

**Policy ACLs**
A policy ACL focuses on protecting URL patterns rather than individual URLs. For this reason, it can group many documents behind it. You can configure policy ACLs based on URL patterns by using the GSA Admin Console, as well as the Policy ACL API. Use Policy ACLs when the number of authorization rules is low and a unique authorization rule can group multiple URLs.

Although not as commonly used as Per-URL ACLs, it is a very flexible tool that can come in handy in unique situation. For example, if there is a globally defined group that should be denied access to an easily identifiable content source, defining a single Policy ACL entry could be the option. Another case is when the content system uses coarse grained permission rules. For example, CA SiteMinder allows the definition of access control based on URL patterns. Those rules can be easily translated to Policy ACLs.

As of GSA version 7.0, Policy ACLs require the specification of: domain, namespace, and case sensitivity.

### Late binding authorization

With late binding, the search appliance doesn't have authorization information for secure content (that is, ACLs) itself. Before the GSA returns search results to the user, it has to check security by contacting a third-party component to validate if the user is able to read each protected document that is part of the results. In response, the third-party component returns the authorization decision to the search appliance. The third-party component could either be the content source itself or an authorization server that centralizes that decision.

The GSA supports the following late binding approaches:

#### Connectors

Google provides some connectors, including SharePoint and Documentum that can be used in your projects to integrate the search appliance with third-party sources, being fully supported by Google. They run on a connector framework created by Google that you can also use to create your own connectors. The main advantage of using this platform to create your own connectors is that it provides a tight integration from configuration and indexing, to security with the search appliance.

Connector framework provides the SPI interface for the authorization to be implemented by any connector. The interface works in batch mode (multiple documents in one call) so that it provides answers without too many round trips. There are also other Google partner-provided connectors that are based on the framework and use this approach.

#### SAML Authorization

SAML is an XML-based framework for communicating user authentication, entitlement, and attribute information. It is a standard that can be used for authentication, but optionally, it can also be used for authorization. Authorization SPI explains how SAML can be used for authorization. Using SAML for authorization doesn't require using it for authentication, and vice versa, as they both are totally independent. In this case, the search appliance sends SAML authorization requests in XML format to the external service you have configured, and that server responds with the authorization permissions for each document.

Off-the-shelf SAML authentication products (IDPs) are quite common, but authorization service providers are not so common. SAML Bridge provides such functionality for using Kerberos impersonation to authorize using batched Head Requests. That is considered a legacy feature from a time when connector and ACL authorization were not available. This means that this approach will most likely be a custom project developed by you instead of using an existing product.

SAML authorizations can be managed in batches, so that the search appliance can send a list of URLs for authorization per request, which can speed up the process. You can activate this option in the GSA Admin Console, but your SAML authorization provider has to support it.

**Head requests**
Finally, it's also possible to send an HTTP head request to the content source to validate authorizations. The GSA can send an HTTP request using the document URL and read the HTTP response from the source to determine authorization based on the HTTP error codes:

- 200: This code basically means that the user is able to access the document, so the search appliance would consider it as permission. It's also possible to define some exclusion rules in the search appliance, as there are some content sources that include 200 HTTP error codes, including a no access permitted message, as in the case of some web portal solutions.
- Any other error code means the user is not able to access that particular document.

To verify the permission for all the results, one Head Request is sent per document sequentially until enough permitted documents are found to fill up at least one search results page. That's why a Head request is the worst performing authorization mechanism. It is generally used when there is no way to extract the ACL, or to verify permissions using an API.

**Connector 4.0**(beta)
The Connectors Release 4.0 is a new connector framework based on a completely different architecture from previous releases. Note that it is still in a beta release. The security features it provides also work differently from previous releases. Here are some key differences for security:

- A connector can be built to provide authentication and authorization. The communication protocol between the appliance and the connector is no longer proprietary XML. Instead, SAML is used as the underlying message exchange mechanism. An example of this connector is the Google Authentication Adaptor, which provides Authentication to Google ID. The way to configure it is the same as configuring a SAML provider.
- A connector built on the 4.0 framework supports Per-URL-ACL.
- Connectors can be built to provide group resolution through SAML authentication. However, the preferred group resolution to use as of GSA 7.2 is onboard groups resolution (beta). Group resolution through SAML is part of SAML authentication, unlike the previous connector framework where connectors can solely perform group resolution while authentication is performed by another mechanism.

## Selecting an authentication mechanism

There are usually several authentication mechanisms at your disposal for a deployment. As stated in chapter one, the main goal is to use as few authentication mechanisms as possible. Quite often there is also an additional requirement: silent authentication. Not all authentication mechanisms can work with all authorization mechanisms. We can categorize all authorization mechanisms into two types:

- **User ID is required**
- **User ID is not required**

All authorization mechanisms require User ID except Head Requests. The following table lists authentication mechanisms that would result in a User ID:

| | Authentication mechanisms when user ID is required |
|---|---|
| **HTTP Basic/NTLM** | It is listed as HTTP Basic/NTLM. However, these are the authentication protocols used to verify the user credentials which happens between GSA and a back end server. To the end user, it is forms authentication. After the user credentials are verified by the configured “Sample URL”, the User ID entered by the user is treated as the verified ID. |
| **Client Certificate** | Certificate’s DN is passed as the verified ID. |
| **Kerberos** | The Windows user ID extracted from the Kerberos ticket is used as the verified user ID. |
| **SAML AuthN** | The “Subject” passed by the SAML IDP is the verified ID. |
| **LDAP Authentication** | User ID verified by the LDAP server is used as the verified ID. |
| **Forms Authn with Cookie Cracking** | User ID is passed back to the GSA by the Cookie Cracker. This involves some coding where a simple dynamic web page needs to be implemented to pass this. |
| **Connectors** | Connector Framework provides the authentication SPI which returns a trusted User ID. However, it must be implemented by the connector, which is optional. Not every connector available provides authentication. Connector implementers can even choose to require a password. For example, The File system connector 2.x requires both the username and password in order to perform late binding authorization when there are deny rules on documents. |

All the authentication mechanisms above can be mixed with ACLs, Connector and SAML authorizations. You can pick the one that fits customer requirements and is easiest to implement (also take into consideration any silent authentication requirements).

For Head Request authorization, you cannot pick just any possible authentication mechanism as the head requests are sent from the GSA to the content source, not by the client’s browser to the GSA. Depending on the authentication protocol used by the content source, different credentials must be obtained by the GSA during the user authentication process.

| | Authentication Mechanism when user ID is not required (Head Requests) |
|---|---|
| **Cookie** | This is the most common situation; the search appliance forwards user cookies to validate access rights. Forms authentication is required. Cookie Cracking is not needed as a user ID is not required. The rule is configured under **Universal Login Authentication Mechanism > Cookie.** |
| **HTTP** | An HTTP Basic/NTLM rule must be configured. The communication between the end user browser and the search appliance is in fact Forms authentication instead of using HTTP Basic authentication protocol. The rule is configured under **Universal Login Authentication Mechanism > HTTP**. |
| **Kerberos** | An HTTP Basic/NTLM rule must be configured. The communication between the end user browser and the search appliance is in fact Forms authentication. The rule is configured under **Universal Login Authentication Mechanism > Kerberos**. |

## Mapping authentication to authorization

Most of the time, you only need to select one authentication mechanism to verify users. You can configure multiple authentication mechanisms, but what are the implications, and why would you need them? Here are some rules to remember:

- When there are multiple Credential Groups, you will need to select one mechanism for each. They can be of the same type, for example, two forms authentications or two SAML authentications. Or they can be different:; one is forms authentication, and another is Kerberos.
- You can also configure multiple authentication mechanisms for the same Credential Group. This is a less common usage. The exception is when a connector is used for group resolution. In this case, one authentication mechanism (most likely silent) verifies the search user, and the connector resolves group membership for ACL authorization. We will discuss more in later chapters.
- All the authentication mechanisms will be fired. When two authentication mechanisms are defined for one Credential Group, the second will be fired up even if the first rule has been satisfied with a verified user ID.

To map authentication to authorization, the search appliance uses a feature called “flexible authorization”, which is similar to a routing table for authorization mechanisms. It allows the administrator to configure the authorization process for documents per URL pattern as they see fit for their deployment. Flexible Authorization is managed through configuring authorization rules. A rule would consist of the following: the content to which the rule applies (defined by URL pattern), an identity that maps the rule to a credential rule or authentication mechanism, and other information specific to the authorization mechanism.

Most of the time, you don’t have to make changes to the Flexible Authorization settings. The default settings would work. It’s a routing table where you can mix and match authentication to authorization, but

there are clear rules on what rules can or cannot be used together:

- Per-URL ACL
    - The ACLs are part of the index that can not be added or removed on the fly. If URLs don't have ACLs attached, Per-URL ACL can't be used as a mechanism for those URLs. The Credential Group associated with the ACLs is also determined during index time which cannot be changed in the Flexible Authorization settings.
    - When the Per-URL ACL rule is defined as the first rule, the authorization happens in the index when matching results are identified. This gives Per-URL ACL much better performance than other authorizations. That's also why it's listed even before Cache authorization by default.
    - If you define a specific URL pattern instead of "/", or move the rule down the list below other authorization rules, it will be the Security Manager that performs the authorization. This means the performance will be worse as Per-URL ACLs would then be evaluated out of the index.
- Connector
    - For content to be authorized using Connector authorization, the URLs must start with "googleconnector://".

After the GSA has authenticated a user through a configured authentication mechanism, authorization to documents will be applied in order of their definition in the flexible authorization table for the particular URL pattern of the document. If more than one authorization mechanism applies to the document, the GSA will cycle through all the rules that apply, in order, until one of them returns a status of PERMIT or DENY. For example, if a connector sends in documents with ACLs, the Per-URL ACL rule will be evaluated first. If PERMIT or DENY is returned, that's the final result. However, if INDETERMINATE is returned, the "Connector" rule will be used to evaluate the documents.

## Summary

In this chapter, we have reviewed the process of designing security for your enterprise search project with the Google Search Appliance. This requires a solid understanding of security in your organization, as well as the related content sources that will be part of the project. Here is a summary of the process in designing the solution:

- Spend time up front to analyze the content sources: How will the content sources be acquired, what authentication is used, etc.
- Determine how many Credential Groups will be needed.
- Determine the preferred authorization mechanism for each content source.
- Determine the minimum set of authentication mechanisms needed.
    - When possible, support silent authentication.
    - When possible, use supported, out of the box components.
    - Will these authentication mechanisms support the relevant content source authorization?
- Configure Universal Login Auth Mechanisms.
- Make changes to the Flexible Authorization Rules when necessary.

# Chapter 2 Using Out of box features

In this chapter, we will look at the details of some of the authentication and authorization mechanisms. We will also discuss common scenarios that are supported by Google Search Appliance and related products offered by Google. We will focus on scenarios that don't require writing code.

## Silent authentication

IT security aims to protect applications and data, providing accurate information to users, but in a secure manner. But It's also important for access control mechanisms to have a minimal impact on users. For example, if a user has already been authenticated by a trusted component, applications should rely on that process to avoid prompting the user multiple times for their credentials or to verify their identity. This is the concept behind silent authentication—verifying a user's identity on the GSA without prompting or requiring them to go through an additional login process.

Silent authentication can be implemented for a search service as for any other application in an organization. There are different authentication mechanisms that enable you to provide a silent authentication experience, including protocols, such as Kerberos or NTLM, or corporate applications such as an SSO system.

Before you implement silent authentication for your search environment, answer the following questions:

- What are the silent authentication options within your organization? Is there a preferred option?
- In the case where there is more than one silent authentication option (for instance, forms based and Kerberos), are they managing the different user identities and credentials needed for the authorization? You have to understand if just using one of them would be sufficient or you need to use both. Also consider if one can assert the identity of the other.
- Are there any multiple authentication domains? For instance, different Windows domains for Kerberos. This information is also important for modeling the authorization process.
- Which applications or content sources that you have to integrate with the search engine are also using the silent authentication mechanism? You might be able to leverage it.

The search appliance can be integrated out-of-the-box with the following silent authentication protocols/systems:

### Forms or cookie-based authentication

Forms or cookie-based authentication is the process driven by a session cookie, typically from a Single Sign-On system. This could potentially be silent if the user has already been authenticated before reaching the search application. If not, the user would be prompted for credentials to create the proper session cookies that provide the SSO experience. The Cookie Authentication Scenarios section in GSA documentation provides technical details about how to integrate with a SSO system. If it's also required to pass a user ID to the search appliance, you have to implement a cookie-cracking process.

### Kerberos

The Kerberos protocol is used by default in Windows networks. The search appliance can be configured to enable Kerberos so that the authentication is transparent to users.

### SAML

Many SSO systems support the SAML protocol, and provide a silent authentication process. Note that SAML protocol is a way for an external service to securely assert the user's identity to GSA. The actual authentication between the user and this service is still going to be standard authentication protocols such as Kerberos, NTLM, or Cookie-based. It is rare that you have to write a SAML identity provider (called a SAML IdP) from scratch. It's far more common to integrate GSA with a SAML IdP already deployed in the customer's network.

### Client Certificates

This is not a common scenario. However, in those environments where users do have client certificates, it's also possible to configure the search appliance to authenticate users through X.509 certificates, which can also provide silent authentication to users.

## SAML

The search appliance supports the integration with SAML, a security standard that enables you to create ad-hoc authentication processes off the search engine. If you build a SAML authentication provider, you can code in the authentication logic you might need. If the user is authenticated properly by this external process, the user identity is passed back to the search appliance.

Because SAML is a security standard, it is supported by some commercial and open source authentication solutions and some SSO systems provide a SAML interface. Check whether your organization's authentication solutions already provide such an authentication interface to facilitate the integration with the search appliance. If so, it might not be necessary to develop such a service.

Consider that it's also possible to configure a SAML authorization process as described in Chapter 3, but this is independent from whether SAML authentication is configured or not.

You can refer to the GSA product documentation to learn how to set up SAML in the search appliance.

## Early binding with Per-URL ACL

When utilizing ACLs for authorization, note that all components that make up an ACL must match the resolved identity for the ACL check to pass: domain, user principal, group principals, namespaces for group and user principals, case sensitivity specified, and ACL type (Permit/Deny).

### Group Resolution

Unlike any other authorization mechanism, there is an additional step for ACL authorization: Group Resolution for a verified user ID. The concept of group resolution is very important in the context of early binding ACL support in the GSA. Because a user can be a member of different groups in an identity management system, the same modeling of identity needs to be provided on the GSA. After authentication, the GSA stores the user ID along with the groups the user is member of. There are five options to resolve groups:

- **Groups database**(beta). Starting from release 7.2, the search appliance includes an internal database that stores ACLs. This is still a beta feature that has limited functions and scalability. Group memberships must be fed to the appliance, similar to how documents can be fed to the appliance's index.

- **Connectors**. The Connector Framework provides an interface to resolve groups. It's up to the connector developer to decide whether Per-URL ACL or group resolution is implemented. Of all the connectors that Google supports, the SharePoint connector, Active Directory Groups connector, and Documentum connector provide such a feature.

- **LDAP**. LDAP authentication can resolve nested LDAP groups. It is not recommended for use with Active Directory (use Active Directory Groups connector instead), but it can be used for other LDAP servers.

The three options above can be SOLELY used to resolve groups when authentication is performed by another mechanism. The following two options will resolve groups as part of the authentication process—they cannot be used for group resolution alone.

- **Cookie Cracking**. Groups can be returned in a custom header, together with the user ID. It has to be part of the cookie authentication process.

- **SAML**. Groups can be returned as part of the SAML authentication process. It has to be part of the SAML authentication process.

These two mechanisms are generally used at deployments that require custom development. The next chapter contains more in-depth documentation on this.

### Namespace

The GSA supports ACL namespacing. The concept of namespacing was introduced in order to avoid name clashes of users and groups from multiple sources in the index. Here is an example:

User John Smith has two identities, and we've set up two credential groups, jsmith in CG1, and johns in CG2. In the index, all ACLs pertaining to John Smith could be associated with either of the two identities. There must be a way to distinguish them. That's why namespace is introduced.

If the principal scope is user, the namespace is equivalent to Credential Group. In ACLs, the principal must be either:

jsmith in namespace CG1
or
johns in namespace CG2

However, if the principal scope is group, namespace doesn't have to be the same as the credential group of the user. As long as the namespace of the resolved groups matches what is defined in ACL in the index, the permission check will work. Here is an example:

John Smith's first identity, jsmith, is from the company-wide Active Directory. Of course, there are AD Groups that jsmith is a member of. Let's say one of the content sources is Plone, which is integrated with Active Directory, but has its own groups defined. How do we avoid conflicts when there are groups with the same names in both Active Directory and Plone? Groups from Active Directory will have namespace CG1. We can give groups from Plone a different namespace such as plone_space. The ACLs in the index will have the following entries:

```
<principal namespace="CG1" scope="user" access="permit">jsmith</principal>
…
<principal namespace="CG1" scope="group" access="permit">authors</principal>
…
<principal namespace="plone_space" scope="group"
access="deny">authors</principal>
```

As long as the right groups can be resolved for jsmith during group resolution after authentication, the right permissions will be applied:

> CG1:jsmith belongs to groups:
> CG1:authors, plone_spce:authors

### Domain parsing

Domain names are pretty common for user credentials and groups. The search appliance has a separate field for *domain* when the principal is stored for the following cases:

- After the user is authenticated, the resolved verified ID and associated groups contain both username and domain name.
- The principal on document ACLs for both users and groups contain the principal name and domain name.

From different authentication protocols, verified users can take different formats:

- bob@**google**.com
- **google**\bob

The search appliance parses these formats consistently and extracts the domain name and username during authentication and ACL indexing. From the 2 example above, **google** would be extracted as the domain.

### Late binding for ACLs

When using ACLs to govern access to documents in the GSA, you might want to configure a late binding fallback in case the ACLs in the index are not fully in synch with the content source due to timing issues. When the late binding fallback feature for Flexible Authorization is enabled, the GSA will only accept a DENY response for the POLICY and Per-URL ACL mechanisms. For PERMIT and INDETERMINATE, the GSA will apply subsequent rules until one of them returns a decision other than INDETERMINATE. If none do, the result will not be presented to the user.

## Connectors using Per-URL ACL

### Local Namespace

The Connector Framework introduced the concept of "Local Namespace." Note that this is a connector concept. For ACL definition, there is only one namespace attribute. In connector configuration, there are two namespace fields: one is "Global namespace", which is equivalent to the Credential Group in Authentication. The other is "Local namespace", which will be the name of the connector (or the name of another configured connector, selectable in the dropdown).

Let's use the previous example of the Plone content source. If a Plone connector is built based on the connector framework, with the instance name of "plone_connector", here are what the ACL principals look like in feeds sent by the connector:

```
<principal namespace="CG1" scope="user" access="permit">jsmith</principal>
...
<principal namespace="CG2" scope="user" access="permit">johns</principal>
...
<principal namespace="CG1" scope="group" access="permit">authors</principal>
...
<principal namespace="CG1_plone_connector" scope="group"
access="deny">authors</principal>
...
```

The search appliance concatenates the "Global namespace" and the "Local namespace" in the connector's configuration as the "namespace" attribute in ACL sent via feeds.

### Avoiding domain parsing

As the previous section described, the appliance will try to interpret the principal format and extract the domain out of it. However, there is only one exception: When ACLs are sent in via feeds, if “unqualified” is set for the attribute **principal_type** on a principal, the domain will not be parsed and the name will be treated as a literal no matter what format it takes. This attribute and behavior is designed as another option to avoid group name conflicts—mainly as a hack to keep the SharePoint connector backward compatible. SharePoint allows you to define groups at different levels of a hierarchical web site structure. If we are to use the “Local namespace” feature of the connector, there will be one namespace per site. GSA’s Connector for SharePoint prefixes all SharePoint local groups with the site URL which the groups belong to, and sets the principal_type to “unqualified.” The search appliance will store these groups as they are passed in so that there won’t be any conflicts of the same group name from different sites. Here is an example of SharePoint local groups being sent to GSA in feeds:

```
<principal principal-type="unqualified" namespace="Default_sp" case-
sensitivity-type="everything-case-insensitive" scope="group"
access="permit">[http://w2k8r2entsp1]Home Owners</principal>
```

On the other hand, if an AD group is sent, it will look like the following:

```
<principal namespace="Default" case-sensitivity-type="everything-case-
insensitive" scope="group" access="permit">mydomain\Home Owners</principal>
```

## Connector 4.0(beta)

### Working with Per-URL ACL

The indexing of ACLs by Connector 4.0 differs from that of previous versions:

- ACLs are not sent in via feeds. Instead, they are indexed as HTTP headers.
- If ACLs are hierarchical, they won't be flattened. Inheritance will always be used.
- Namespace needs to be handled by each connector. The File system connector and SharePoint connector use the name "*adaptor.namespace*" as the configuration entry.
- There is no more Local Namespace concept—you are free to specify any namespace. The ACLs from this connector will all use the same namespace. Except in the following scenario:
    - **principal-type** is no longer used by connector 4.0. The scope of SharePoint groups will be appended to namespace, and the principals will be sent without the prefix. For example, "My SP Group" group within http://sharepointhost/sitecollection/ will be processed by the SharePoint connector as follows (assuming the Credential Group is "Default"):

      *Namespace: Default*_http://sharepointhost/sitecollection/
      *Principal name: My SP Group*

      If the principal has a domain such as mydomain\mygroup, it will be processed as follows:

      *Namespace: Default*
      *Principal name: mygroup*
      *Domain: mydomain*

### Authentication

As discussed in Chapter 1, connector authentication uses the SAML protocol. The connector framework 4.0 provides SAML as the foundation for security. Connectors based on the new framework must provide its own implementation of the authentication process for the targeted content source. Here is how you configure it in the Admin Console: Under **Search > Secure Search > Universal Login Auth Mechanisms > SAML**, you need to enter the following values:

*IDP Entity ID*: The **server.samlEntityId** configuration entry from the connector configuration file.
Login URL: https://connector-host-name:port/samlip
Public Key: <public key of the IdP>

Here are some notes about the SAML implementation by the connector:

- You can have multiple connectors providing authentication. The Entity IDs will be different.
- Only Post Binding is supported.
- The Endpoint of the SAML IdP "samlip" is hardcoded
- Groups can be returned as part of the SAML assertion in the "member-of" attribute.

### Authorization

The “Authorization” in this section refers to late binding when using connector 4.0. In order to configure this, you need to perform the following: In Admin Console, under **Search > Secure Search > Flexible Authorization**, the **Authorization service URL** needs to be set to: https://connector-host-name:port/saml-authz.

## Security in Windows environments

The majority of deployments of the appliance, which incorporate secure content search, occur in a Microsoft Windows environment. Google provides two accompanying products for the integration: the SAML Bridge and the Active Directory Groups Connector.

### SAML Bridge

The search appliance directly supports Kerberos authentication in Windows without the need for installing any external components to the GSA. Since Kerberos is supported in all Windows environments, it is the recommended mechanism for silent authentication. However, it might not be sufficient for the following reasons:

1. Kerberos is quite sensitive to the environment. For example, a client device might not support Kerberos; some network scenarios might not support Kerberos. In those cases, native Windows clients fallback to NTLM authentication. However, the search appliance does not natively support NTLM so there is nothing to fall back to.

2. Some organizations do not allow the use of key tab files for Kerberos. GSA uses a key tab file in order to enable Kerberos.

3. When the GSA is Kerberos enabled and used for Head Request authorization, it can only perform unconstrained delegation. This is not acceptable for some organizations.

If you want to enable silent authentication when Kerberos cannot be used (or the key tab file cannot be used), you must set up an external authentication process. Google provides an open sourced tool called the SAML Bridge that supports these scenarios. This is a SAML-based solution that runs on the Windows infrastructure, so it must be installed on a separate host, able to authenticate users using either NTLM or Kerberos. For detailed information about how to set up the SAML bridge, see Enabling Windows Integrated Authentication.

### Active Directory Groups Connector

In a Windows environment, many content sources are integrated with Active Directory. Groups in Active Directory are used to control access to certain resources. The Google Search Appliance Connector for Active Directory Groups is a tool that can be used to support early binding. It is the preferred approach to resolving groups needed for early binding, versus LDAP authentication, which can be configured directly on the GSA. Although LDAP authentication can also be used for Active Directory groups resolution, it is “late binding” when it resolves groups, in that during the authentication process, the appliance will try to contact domain controllers directly to get associated groups for a user. Alternatively, the Active Directory Groups connector performs the “early binding” of groups resolution: It traverses Active Directory and stores all the user group membership information in its own database. During serve time, the connector just reads from this database instead of contacting domain controllers directly. It offers much better performance—especially in a large scale, multiple domain environment.

Here are some unique behaviors and deployment best practices:

- The connector will run for a long time—it could be days if the Active Directory has a lot of users and groups. It's recommended to:
    - Use dedicated AD Groups connector instances. This is true even for the SharePoint connector which has an embedded Active Directory Groups connector capability and can index both SharePoint content and Active Directory groups.
    - Increase the traversal time out. There are six stages to complete the traversal. You can see that in the logs. If you see repeated "update 1/6" and ""update 2/6", but it nevers goes beyond that, it's a sign that the traversal thread was interrupted before it could finish. You can increase the time by changing the variable traversal.time.limit in INSTALLROOT/INSTANCENAME/Tomcat/webapps/connector-manager/WEB-INF/applicationContext.properties
- Make sure you are binding directly to a non-load balanced Domain Controller Host to take advantage of incremental AD traversal.
    - The connector uses checkpointing that is unique to a specific Domain Controller, so in order to take advantage of the checkpointed updates, you must continue to connect to the same unique Domain Controller upon every request.
- Always use an offboard connector, and one connector instance per connector manager.
    - Easier to patch and troubleshoot
    - More scalable since you can control resource consumption easily.
- Use an external database to store the group information.
    - It is more reliable for production than using the embedded database.
    - As the embedded database is tied to a Connector Manager instance, it is also the only way to correctly resolve groups when multiple combinations of AD groups connectors and SharePoint connectors are used over multiple Connector Managers. For example, when there are multiple AD domains, there must be one connector for each domain. In order to resolve groups for users from different domains or if memberships cross domain, the groups information must be put in the same database and same tables. Since the database configuration is at the connector manager level, you have to configure these connector managers to use an external database in order for the multiple instances to share the same related data.

## Perimeter security

Documents in the search appliance index can be labeled as either "public" or "secure." How a document is labeled depends on how the content was indexed, either by crawling or feeding, as well as the configuration information in the GSA. In terms of security, an indexed document falls into one of the following two categories:

| Public document | Secure document |
| --- | --- |
| • Public crawled document<br>• Feed document with no security<br>• Content from a secure content source that has been marked as public by using the GSA Admin Console | • Securely crawled document<br>• Feed document declared as secure |

Users can search and get to public documents without authentication. However, there is an exception. GSA 6.14 introduced the perimeter security feature to the GSA, which ensures that the search appliance doesn't serve any results without user authentication. When perimeter security is enabled, the search appliance must authenticate a user with one of the configured authentication mechanisms before serving any results. If authentication fails, the GSA will not serve any results, even if they are public. Take note that only authentication is performed for documents marked as public, without the need to do any authorization.

To configure perimeter security, set up an authentication mechanism, which can be any mechanism described in Chapter 2. After that is done, navigate to **Serving -> Universal Login** and enable perimeter security. Take note that once perimeter security is enabled, it applies to the GSA globally and cannot be configured per collection or front end.

## Secure Search Example

Here are requirements for four content sources to be included in search (all secure):

1. SharePoint 2010, with Kerberos authentication. Google supported connector for SharePoint is used to index the content.
2. Salesforce content integrated with a SAML IdP which uses Forms authentication, but the user directory is still Active Directory. A Salesforce connector is deployed to index the content with ACLs. The connector is built based on Google's connector framework and sends in documents starting with "googleconnector://".
3. A custom IIS web site with Kerberos Authentication. No API is available for checking permissions or getting ACLs. GSA will crawl the content directly.
4. A legacy business application. Users and permissions are stored in the database. It's not integrated with Active Directory. There is no direct mapping of user names between Active Directory and this application. Google's database connector is used to index the content. A SQL Query statement can be used to determine whether a user has access to the database records in the search results.

It is also noted that there are various devices in the organization. Some don't support Kerberos.

**User Identities**

SharePoint, Salesforce and the custom IIS web site are backed by the same Active Directory, while the legacy application has its own. That means we need two Credential Groups: We can use the "Default" Credential Group for Active Directory, and add a "Legacy" Credential Group for the business application.

**Authorization**
When we try to come up with a solution, you need to start with authorization. It's obvious that we should use Per-URL ACL for SharePoint and Salesforce content. Because GSA's connector for Database supports authorization using a query, we can use connector authorization for this content. We will have to use Head Request for the custom IIS web site. Since it uses Kerberos, we can use Head Request with Kerberos. SharePoint, Salesforce and legacy applications need a verified user identity, while the custom IIS web site doesn't.

**Authentication**
Now that we have decided which authorization mechanisms to use, it's time to select authentication for each Credential Group. For the "Default" Credential Group, we cannot use Kerberos on the GSA because there are client devices that don't support Kerberos. It leaves us with the SAML Bridge as an alternative. Upon closer investigation, we might be able to use the already available SAML IdP used by the Salesforce integration. It will return the same verified identity, and it doesn't require an additional server to host the SAML Bridge.

Next, we have to validate whether this authentication strategy is sufficient for authorization requirements tied to the "Default" Credential Group. SharePoint and Salesforce content should be covered since we are getting a verified identity that will be used for the ACL checks. The custom IIS web site poses a challenge if we use the Salesforce SAML IdP with cookie authentication since no Kerberos ticket would be available for the Head Request Authorization. To fulfill this requirement, we can use the SAML Bridge for authorization only because it supports Kerberos delegation. It can perform batched Head Requests using Kerberos given a username. But this means we still need to deploy the SAML Bridge. If we need to deploy the SAML Bridge anyway, we can use it for Authentication as well.

For the "Legacy" Credential Group, we need to perform authentication against user credentials stored in the database. However, the GSA connector for Databases does not provide an authentication mechanism. In this case, customization is needed to implement the AuthenticationManager interface of the Connector Manager in the database connector.

As we now know what Authentication mechanisms we'll be using, in Universal Login Auth Mechanism, we configure the following two rules:

1. **SAML**. Using the "Default" Credential Group, SAML Bridge should be configured in POST Binding mode. See the Wiki documentation for detailed instructions.

2. **Connector**. Using the "Legacy" Credential Group, the customized Database connector should be configured.

**Flexible Authorization Rules**

In general, for most deployments, we can leave the first 3 entries of Flexible Authorization alone: PER_URL_ACL, CACHE, and POLICY. This also applies for this particular deployment. PER-URL-ACL rule will kick in for SharePoint and Salesforce content because ACLs are indexed with documents. We do have to make some changes to the CONNECTOR rule because the default configuration is only associated with the "Default" Credential Group.

- CONNECTOR
    - Change the **Authentication ID** to "Legacy"—it's equivalent to the selection of Credential Group here.
    - Fill in the database connector name in **Connector Name** field.

We also need to define a SAML rule. Although SAML Bridge uses Head Request to authorize custom IIS web sites, we cannot rely on the "HEADREQUEST" rule because that's for GSA to perform Head Request.

- SAML
    - It should be right after CONNECTOR rule in the Flexible Authorization order.
    - **Authentication ID** should be "Default" (maps to Credential Group).

**Authorization Service URL** should point to Saml Bridge's Authz.aspx.

# Chapter 3 Authentication for Developers

Whenever possible in your deployments, you should try to use existing products, either supported by Google, provided by Google's partners, or other 3rd party off-the-shelf products. In general, following this guidance minimizes project risks and reduces overall ownership costs. However, there may still be some requirements for which you have to develop external custom applications or processes in order to fully implement security or content integration with the GSA.

The following options are available with the GSA for building custom external authentication processes:

- Forms authentication with cookie cracking
- SAML
- Connector Framework
- Trusted Application*New*

## Forms authentication with cookie cracking

If your systems are already using cookie-based authentication, an option for security integration with the GSA is to reuse and customize the existing authentication process to create a silent authentication experience on the GSA. Cookie cracking is one possible customization to the existing forms authentication process, which permits you to extract the user identity behind the user's authentication cookie and forward that to the search appliance.

The cookie cracking process must be able to discover who the user is behind the cookie by contacting an external URL, using SSO APIs or the like, and sending user credentials as HTTP Headers to the search appliance in a secure manner. The user must already be authenticated by the SSO system that has created the session cookie before reaching the search appliance. If not, the user will be redirected to a login page to establish the identity with the SSO, after which the credentials would be sent to the GSA.

To implement cookie cracking with your Forms-Based/SSO system, you need to configure an external URL that is protected behind the SSO system. During the authentication process, the GSA contacts that URL, forwarding the session cookies already created by the SSO, so that the external service can verify the user identity and send it back to the search appliance in an HTTP Header named `X-Username` and optionally, `X-Groups`. You can fully customize this process in a way that lets you model security for your enterprise search project.

To create the cookie-cracking process, you must perform the following actions:

1. Create a web application that is able to validate a user's identity based on the SSO session cookie.
2. Configure a forms-based authentication rule in the GSA's Admin Console.

**Key considerations**
If you want to achieve a silent authentication experience with your SSO system, consider the following items:

- A session cookie must not be restricted to the same user IP—some SSO systems support this restriction as a security measure.
- The GSA must be part of the same domain as the session cookie used by the SSO system. For instance, if a cookie is using domain “foo.com”, the GSA must be configured as part of that domain, for instance “gsa.foo.com”. This way the browser will send SSO cookie to the GSA. The cookie and the GSA domain must be scoped properly and in synch.
- The cookie cracking application should be a simple web application written in any programming language supported by your web/application server, such as Java or .NET. The application sends the username back to the search appliance via a custom HTTP header. For example:
  `X-Username: luis.sanchez`
- The cookie cracking application is usually deployed behind the SSO system. It means that it can be installed behind the SSO web plug-in that authenticates the user and usually passes that identity to trusted web applications in an HTTP header in a secure way.
- It doesn’t have to be a standalone application. It can be an additional ASP, JSP or other dynamic web page in an existing application that’s protected by the same login form.
- The “cookie cracker” doesn’t really “crack a cookie”—it just means that after authentication, a cookie is generated that indicates a valid user session. The username might be obtained in various ways. For example, a commercial SSO system like SiteMinder uses HTTP_SM_USER header to add the user ID.
- If it’s not possible to use a web SSO plug-in in front of such a custom web application to facilitate the deployment of this solution, you can use the SSO API to extract the user behind the cookie. Never fake an SSO system in production that passes the user ID in a cookie in plain text as this is a high security risk.
- Take into account some SSO systems might be configurable to pass those credentials in a HTTP header without the need of developing a web application.

**Group resolution for early binding (ACL authorization)**
A cookie cracker authentication response can also be augmented to contain group information for the user being authenticated. The cookie cracking process would not change, the only change required would be to the headers on the response, as they would now also contain group information for the user. All groups resolved through the Cookie authentication mechanism will belong to the global namespace of the credential group selected for the mechanism. If groups are required for the authenticated session, an X-Groups HTTP header would also be added:

```
X-Groups: department_users, enterprise_users
```

## SAML

The search appliance supports SAML 2.0, an XML based protocol for an external identity provider. There might be cases where you will need to develop a custom SAML IdP. Note that building a SAML IdP from scratch is time consuming. You should start with an existing code base—such as OpenSAML. Google also provides an open source project SAML Bridge for silent authentication with Windows technologies.

Bear in mind that there are two different ways to set up SAML authentication with the search appliance:

- HTTP Artifact binding, where the browser is used as the main communication mechanism between the GSA (Service Provider) and the Identity Provider (your SAML server).
- HTTP POST binding, which requires a trust mechanism between the GSA (Service Provider) and the Identity Provider (your SAML server).

It's not the intention of this document to guide you through setting up this configuration, as this is already described in the GSA documentation, but the following table contains tips about which SAML binding to use.

| | SAML HTTP Artifact Binding | SAML HTTP POST Binding |
|---|---|---|
| **Requirements** | Multiple transparent redirections are required. Trust between the browser and the GSA/SAML Identity Provider is required. | Apart from the trust between the browser and the other servers, it also requires you to create a trusted link between the GSA and the Service Provider. |
| **Configuration** | Secure HTTP connections from the client with GSA and Identity Provider are desirable. An additional URL has to be configured on the GSA (Artifact Resolver URL) to provide final authentication information to the GSA. | Apart from having SSL connections between client and servers, it also requires you to configure certificates between the GSA and the Identity Provider. This is used for the GSA to get the authentication information directly from the Service Provider. |
| **Public key infrastructure** | It's less complex from a security perspective as it doesn't necessarily require you to have a PKI (Public Key Infrastructure) solution. | It does require issuing trusted certificates so a PKI infrastructure is needed. Take into account this can also be done by using OpenSSL solution. |
| **High availability** | It's more complex to provide a Highly Available solution for the GSA and the Identity Provider due to the multiple browser redirections and required persistence of the artifact between multiple calls. | High Availability can easily be implemented. |
| **SAML identity provider** | If you have to develop a SAML Identity Provider based on Artifact | Developing a SAML Identity Provider from scratch could be simpler, but it |

| | | |
|---|---|---|
| | binding from scratch, it could be more complex as it requires an extra service (Artifact Resolver URL). There are some open source frameworks like OpenSAML and also many code samples for this on the Internet. | requires managing Digital Signatures in your code. |

In general it's more desirable to use SAML HTTP post binding because it provides a stronger and simpler solution, mainly in terms of high availability.

**Group resolution for early binding (ACL authorization)**

The SAML authentication response can be augmented to provide user groups for the authenticated user back to the GSA. All groups resolved through the SAML authentication mechanism will belong to the global namespace of the credential group selected for the mechanism.

**Sample SAML Response:**

In this sample, you can see that the SAML response contains both the username ("Subject") and a "member-of" AttributeStatement containing resolved groups for the user.

```
<Assertion Version="2.0"
        ID="blahblah2"
        IssueInstant="2011-01-01T14:38:05Z"
        xmlns="urn:oasis:names:tc:SAML:2.0:assertion">
  <Issuer>ac.corp.company.com</Issuer>
  <Subject>
    <NameID>luis.sanchez</NameID>
  </Subject>
  <AuthnStatement AuthnInstant="20011-01-01 T14:38:05Z">
    <AuthnContext>
      <AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
      </AuthnContextClassRef>
    </AuthnContext>
  </AuthnStatement>
  <AttributeStatement>
    <Attribute Name="member-of">
      <AttributeValue>marketing</AttributeValue>
      <AttributeValue>us-employees</AttributeValue>
      <AttributeValue>SFO-office</AttributeValue>
    </Attribute>
  </AttributeStatement>
</Assertion>
```

## Cookie cracking vs. SAML

If you need to customize your authentication process, it's important to differentiate between cookie cracking and SAML so that you can plan the best approach before starting the project.

| | SAML | Cookie Cracking |
|---|---|---|
| **Integration** | Some Single Sign-On systems provide a SAML authentication interface that might be integrated out of the box with the appliance | Some Single Sign-On systems can be integrated easily through cookie cracking |
| **Complexity** | It could be more complex if you have to develop a SAML provider from scratch | Development costs to develop a cookie cracking solution for the appliance could be lower |
| **Authentication** | There is an interaction between the browser (user) and the Service Provider, so it can be used with any point-to-point authentication protocol like Kerberos or NTLM | In the authentication process, the appliance contacts the Sample URL with no interaction from the user so it's only valid for a cookie-based authentication approach |

To get the exact technical details about how to implement both approaches, see Cookie Cracking and Authentication and Authorization SPI. It's important to understand the interaction flows behind them both to implement those processes properly.

## Connector Framework for Group Resolution

The Connector Framework also provides an interface for user authentication. However, since it's not a silent authentication mechanism, connector authentication is not recommended. On the other hand, the connector can be implemented to provide group resolution for early binding which proves to be much more useful. It's common for a silent authentication mechanism such as Kerberos, SAML or Cookie cracker to be combined with connector-based group resolution.

**Interface support**

The Connector Framework defines the following interface to be implemented by a connector developer:

```
public AuthenticationResponse authenticate(final AuthenticationIdentity
identity)
      throws RepositoryLoginException, RepositoryException
```

When a connector is configured in Universal Login Authentication Mechanisms, there is an option to "Perform group resolution only".

When the connector is intended to provide both authentication and group resolution, the implementation can ignore what the GSA passes to it through the AuthenticationIdentity object and provide a verified username and groups back to the GSA through AuthenticationResponse. Here is the constructor of AuthenticationResponse:

```
public AuthenticationResponse(boolean valid, String data,Collection<?> groups);
```

“valid” indicates whether the authentication is successful or not, if the “authenticate()” method is used to perform authentication. “data” is reserved for future use. The collection “groups” will be used to hold user groups of the following class:

```
public class Principal;
```

This class holds information about groups: name, namespace, case sensitivity, and principal_type. These are attributes of a Per-URL ACL. The last attribute principle_type is used by Connector Framework to introduce a concept of “Local Namespace”. See the Namespace Overview section for more details on how namespaces should be mapped. Here is how it’s used during traversal:

1. The connector is configured in the GSA’s admin console with both a global namespace (credential group), and a local namespace.
2. The connector gathers all groups and creates the **Principal** object. If they are local to one content source, the principal_type is set to “unqualified”. The name of the local namespace is prefixed to the name of the group just like a domain.
3. The Connector Manager will translate the **Principal** objects to the ACL definition in XML Feeds with the properties of **Principal** objects to the corresponding ACL attributes.

**Database support**
There are two design options for connectors to resolve groups during serve time:

1. A connector can query the application where the group memberships are stored during serve time when there is an API available. The Google supported connector for Documentum (version 3.2), which supports early binding, uses this approach.
2. Users, groups, and their relationships could be discovered beforehand and saved in the connector’s own storage. During serve time, the connector reads from this storage and provides it to the appliance. The Google-provided Active Directory Groups connector and SharePoint connector are such examples. Sometimes this is the only option. For example, when there are multiple Active Directory Domains, querying from all of them during serve time can be very slow, thus a real time call is not feasible.

The Connector Framework includes database support. The configuration file applicationContext.properties has JDBC configuration settings for different databases. The Connector Framework comes with a built-in H2 database. When you develop connectors, you can store user memberships in the database with the second approach outlined above.

## Trusted Application(beta)

A very common use case is for the GSA to be deployed behind a portal to provide a search service. The search UI is provided in the portal and users don't interact with the appliance directly. A challenge in such a use case is how to pass the user's credential to GSA without asking the user to login. To achieve that, the search page must simulate browser behavior when interacting with the appliance, obtain the user's credential from the existing portal session, and pass that on to the appliance. Because of the variety of authentication protocols used by portal applications and how the end user credentials are handled, it could be quite a challenge.

In order to make the integration easier for portal developers, the Google Search Appliance added a new feature in 7.2: Trusted Application. The concept is simple: portals use a pre-configured trusted user account to establish the session with GSA using limited, but simple authentication protocols, and sends in secure search requests using an end user's username. It avoids the challenges mentioned earlier with the following:

1. Simple authentication protocols are easy to handle. Basic Authentication and Cookie Cracking are the two supported mechanisms.
2. Since only usernames are required, it solves the problem when user passwords are not available in an established user session in the portal.

End user's identities are sent in using the following two custom HTTP headers:

**X_GSA_USER**: header that holds the username
**X_GSA_CREDENTIAL_GROUP**: header that identifies the credential group of the end user.

**Key considerations**

1. All configured authentication mechanisms will be triggered when a secure search is performed. This includes the mechanism configured for the Trusted Application.
2. You can use any development languages: Python, Java, C#.
3. The "trusted user" must be verified. Since GSA supports cookie cracking and Basic Authentication, it means the performance will be affected by the back end content server. You should avoid performing authentication as much as possible.
4. You can use the returned GSA session cookie from a call to GSA for subsequent calls.
5. Each search request must be accompanied by the end user name and credential group in order to get results for the right user.
6. Group resolution will be triggered as part of the authentication process for the end user. If there is any group resolution configured, the group resolution will be invoked. Since Group resolution also has performance implications, you need to avoid group resolution for every single API call.
7. When the same end user is re-submitted with another search using the same and still valid GSA session cookie, group resolution won't be triggered again. When a different end user is sent in for the first time even with the same GSA session cookie, group resolution is triggered for that user.

8. When the trusted user session expires (cookie expired based on Session timeout setting under **Secure Search -> Access Control**), the GSA will return an error: "The remote server returned an error: (502) Bad Gateway."

9. When the trusted user session is valid (didn't exceed session timeout value), but the authN mechanism's trust duration expires, the appliance performs another authentication using the trusted user credential. The call becomes as slow as the first call that returned current GSA session cookie.

**Best practices**

1. If the integration is with a portal, the returned GSA session cookie should be stored in the portal user's active session. This means different portal users will have different GSA sessions and all must be stored. It also means that these GSA session cookies will be reused for the same end user to avoid trusted user authentication overhead.

2. The best scenario in terms of performance is for the GSA session to stay valid during the portal user's session. However, there is no such guarantee: the user might browse around the portal for quite a while before performing another search. In the code, you need to handle the case when the GSA session cookie has expired when another call is made.

3. Set the Trust duration of the authN mechanism to be the same as the session timeout. The default session timeout value is 1800 seconds. By doing so, you will avoid the performance hit from another implicit authentication using the Trusted User credential.

4. Domain name should be passed as a prefix to end users' usernames. Otherwise the call will fail.

Please see Appendix A for a sample client in C#. The class needs to be instantiated for each portal user's session, and it retries once when the GSA session expires.

## Connector 4.0 Authentication **(beta)**

A connector only needs to provide implementation for the the following interface:

*public interface AuthnAuthority*

and register it with:

*AdaptorContext.setAuthnAuthority()*

For reference implementation, you can take take a look at the Google Authentication Adaptor. After authentication, an object of class **AuthnIdentity** is returned. It contains the username, and optionally groups or password.

# Chapter 4 Authorization for Developers

## Overview

An enterprise search engine must return relevant results to the user, but only those that the user has access to. This is managed through the authorization process that applies to every secure document in the index. In this chapter we focus on custom solutions when designing the authorization process in your enterprise search project with Google.

The section Select an Authorization Approach introduced the following main options for building a custom authorization process:

- Per-URL ACLs
- Policy ACLs
- SAML authorization
- Connectors

The following sections provide more details on using these options in a custom solution.

## Per-URL ACLs

The biggest challenge of using early binding in a custom connector or feeds is to simulate the authorization model of the target system. Every system's security model can be different.

There are a couple of ways to associate ACLs with documents, such as in HTML headers as metadata, or through custom HTTP headers. However, only feeds allow you to specify all the possible ACL attributes. Since the Google Connector Framework is based on feeds, this discussion covers the case when the ACLs are sent by a connector. See Specifying Per-URL ACLs for information on how to fully define the ACL. Among the features that GSA offers to simulate different security models, ACL inheritance is a very important one.

**ACL inheritance** makes it more efficient to deal with ACL changes. As ACLs no longer have to be expanded and attached to each level in a hierarchy, it makes it more efficient to deal with ACL changes, as you only have to re-index the level at which the permission changed.

**The attribute “inheritance-type”** makes it possible to model the different security mechanisms of various content systems. In an inheritance chain, the permission check always traces back to the top and permissions are evaluated according to the inheritance type that was set:

- PARENT_OVERRIDES
    - The permission of the parent ACL dominates the child ACL, except when the parent permission is INDETERMINATE. In this case, the child permission dominates. If both parent and child are INDETERMINATE, then the permission is INDETERMINATE.
- CHILD_OVERRIDES
    - The permission of the child ACL dominates the parent ACL, except when the child permission is INDETERMINATE. In this case, the parent permission dominates. If both parent and child are INDETERMINATE, then the permission is INDETERMINATE.
- AND_BOTH_PERMIT
    - The permission is PERMIT only if both the parent ACL and child ACL permissions are PERMIT. Otherwise, the permission is DENY.

**Inheritance chain example**
URLs

- "FileUrl" (USER:joe access:PERMIT type:LEAF) inherits
- "FolderUrl" (GROUP:eng access:PERMIT type:CHILD_OVERRIDES) inherits
- "ShareUrl" (GROUP:interns access:DENY type:PARENT_OVERRIDES

Authorization Decisions

- PERMITs identity (USER:joe, GROUP:eng)
    - PERMIT by FileUrl ACL, not overridden = PERMIT
- PERMITs identity (USER:moe, GROUP:eng)
    - INDETERMINATE + PERMIT + not overridden = PERMIT
- DENYs (USER:adam, GROUP:eng, GROUP:interns)
    - INDETERMINATE + PERMIT + DENY (override) = DENY

**ACLs can be “Free” or “Bound.”** ACLs that are attached to indexed documents are “Bound”. “Free” ACLs can represent non-document elements. For example, some content systems define permission objects which can be used by different documents. ACLs are maintained on these special objects instead of on documents. Content systems such as File systems have hierarchies and ACLs can be defined on folders which are not documents. “Free” ACLs can be used in both of these scenarios. They are not counted as indexed documents so they don’t count against a GSA’s license.

**“Free” ACL example**

```
<group>
  <acl url='http://dummyhost.corp.google.com/'
      inheritance-type="child-overrides" inherit-
from='http://corp.google.com/'>
    <principal scope="user" access="permit">edward</principal>
    <principal scope="user" access="deny"> william</principal>
    <principal scope="user" access="deny"> ben </principal>
    <principal scope="group" access="permit">nobles</principal>
    <principal scope="group" access="deny">playwrights</principal>
  </acl>
  ...
  ...
</group>
```

In this example, `http://dummyhost.corp.google.com/` is a free ACL, which inherits from `http://corp.google.com/` and defines further principals. Since the ACL is of inheritance type child-overrides, its child will override this ACL if any.

## SAML authorization

You can fully customize the authorization process through an external SAML provider that resolves authorization. It would be best to build such a SAML authorization process using the program language that you are most familiar with. The SAML Authorization request is an XML-formatted request that the search appliance sends to the service URL that you have configured in the Admin Console. That request contains information about the user and the URLs to be authorized. SAML also supports batch processes, so that multiple URLs can be sent at the same time, something that is very desirable to implement when using this approach for performance benefits in avoiding Authorization chattiness.

The Authentication/Authorization for Enterprise SPI Guide contains more information about the SAML XML format, which you can use to build a custom SAML authorization process. You have to implement the service that runs on an external application server that parses the response, extracts the information about whether the user has rights to access the document, and returns an XML-formatted response to the search appliance. An example is the SAML Bridge which can perform batch authorization of Kerberized content using Head Requests.

**Key considerations**
Considerations for using SAML authorization:

- The main advantage of implementing this authorization model is that you can fully control the security process at search time.
- The main inconvenience of this approach is that it is intrinsically related to the late binding method. That is, it might take more time to manage authorization, although batch processing can mitigate it.

## Connector Framework for Authorization

Another option for modeling security is implementing a custom connector. As it's explained in this paper and GSA documentation, a connector can be created to "traverse" or feed public or secure content into the search appliance as well as to support serve time authentication and authorization. We have discussed connectors using Per-URL ACL. Here we will discuss using connectors to perform authorization as a late binding mechanism.

**Interface support**
The Connector Framework defines the following interface to be implemented by a connector developer:

```
public interface AuthorizationManager;
```

It has the following method for authorization:

```
public List authorizeDocids(Collection docids, AuthenticationIdentity
identity)throws RepositoryException;
```

"docids" is a collection of unique document IDs for matched search results. Multiple docids are passed from the appliance to a connector. When enough documents are authorized based on search user's identity, the appliance stops calling the connector. Otherwise, the appliance will keep calling this API—each time with more docids than previous call until either the allocated time runs out, docids run out, or enough documents with "PERMIT" are returned to the search user.

**AuthenticationIdentity** holds the verified identity of the user. Depending on the authentication protocol used, it can contain username, domain, or even password (if the authentication protocol deployed gathers password). A connector implementation should decide what minimum information in **AuthenticationIdentity** is required.

## Connector 4.0 Authorization (beta)

A connector only needs to provide implementation for the following interface:

```
public interface AuthzAuthority
```

and register it with:

*AdaptorContext.setAuthzAuthority()*

## Web proxy

The options described above are the most common platforms used to implement the security side of the interconnection with a content source. There are others, such as using a web proxy to manage the authorization.

In this case, the authorization is centralized in a web proxy that requires all URLs to be rewritten to go through it. So the search appliance sends HTTP head requests to validate security before serving results.

**Key considerations**

Using a web proxy is similar to using a SAML authorization provider, but with the following disadvantages:

- Authorization requests are not batched.
- URLs have to be rewritten to go through the web proxy for authorization. For example: `http://proxy.corp.com/proxy?returnPath=http//cont.corp.com/doc.html`
- Those URLs that were rewritten are stored in the index in this manner and may have to be translated again to the original URL in the search front end.

## Summary

In this paper, we have reviewed the process of designing security for your enterprise search project with the Google Search Appliance. This requires a solid understanding of security in your organization, as well as the related content sources that will be part of the project. You need to invest quality time in analyzing this scenario and modeling authentication and authorization in the search appliance.

### Security Best Practices Overview

- Spend time up front to analyze the following:
    - Which identity providers you'll have to integrate with for Authentication
    - How you'll authorize documents from each content source integrated with the GSA
- When possible, use supported, out of the box components to integrate security on the GSA, such as:
    - Kerberos
    - Google Search Appliance SAML Bridge for Windows
    - LDAP
    - Google Search Appliance Connector for Active Directory
- Model each identity provider you have to integrate with a credential group
- Classify credential groups per corporate security systems (identity providers) and associate them with content sources.
    - Whenever possible, use only one credential group per identity provider.
    - Credential groups should be mapped to unique identity mechanisms, not necessarily content sources.
    - One set of credentials can be used across many content sources that share the same identity source.
- Use ACLs to security trim documents as this makes authorization faster and creates a better overall search experience.

# Appendix A

## Sample Trusted Application client code in C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.IO;
using System.Text;

namespace TrustedApp
{
  class GSAClient
  {

    String GSA_SESSION_ID = "GSA_SESSION_ID";
    String _gsaSessionId = null;
    String _trustedUser;
    String _trustedPwd;
    String _gsaHostName;
    String _endUser;
    String _credentialGroup;
    static void Main(string[] args)
      {
        String gsaHostName = "gsa.acme.com";
        String userName = "trusteduser_a", userPassword = "pwd";
        GSAClient gsaClient = new GSAClient(gsaHostName, userName, userPassword,
          "Default", "enduser_a");
        gsaClient.search("access=a&q=some_keyword&site=default_frontend");
      }

    public GSAClient(String gsaHostName, String trustedUser, String trustedPwd,
      String credentialGroup, String endUser)
      {
        _gsaHostName = gsaHostName;
        _trustedUser = trustedUser;
        _trustedPwd = trustedPwd;
        _credentialGroup = credentialGroup;
        _endUser = endUser;
      }

    String search(String q)
      {
        int iRetry = 0;
        HttpWebRequest request;

        Initiate:
        request = (HttpWebRequest)WebRequest.Create("https://" + _gsaHostName +
          "/search");
        request.Method = "POST";
```

```
request.ContentType = "application/x-www-form-urlencoded";
ServicePointManager.ServerCertificateValidationCallback = new
System.Net.Security.RemoteCertificateValidationCallback(AcceptAllCertifications);
request.Proxy = WebRequest.DefaultWebProxy;
request.CookieContainer = new CookieContainer();
if (_gsaSessionId != null)
  {
    request.CookieContainer.Add(new Cookie(GSA_SESSION_ID, _gsaSessionId)
      { Domain = _gsaHostName });
  }
else
  {
    string authInfo = _trustedUser + ":" + _trustedPwd;
    authInfo = Convert.ToBase64String(Encoding.Default.GetBytes(authInfo));
    request.Headers["Authorization"] = "Basic " + authInfo;
  }
request.Proxy.Credentials = CredentialCache.DefaultCredentials;
((HttpWebRequest)request).KeepAlive = true;

//specific to the end user
String strRsps = null;
request.Headers["X_GSA_USER"] = _endUser;
request.Headers["X_GSA_CREDENTIAL_GROUP"] = _credentialGroup;

//"X_GSA_USER: usera1" --header "X_GSA_CREDENTIAL_GROUP: Default" -d "access=a&q=
byte[] byteData = UTF8Encoding.UTF8.GetBytes(q);
request.ContentLength = byteData.Length;
try
  {
    using (Stream postStream = request.GetRequestStream())
      {
        postStream.Write(byteData, 0, byteData.Length);
        postStream.Close();
      }

    HttpWebResponse response = (HttpWebResponse)request.GetResponse();
    if (_gsaSessionId == null)
      {
        _gsaSessionId = response.Cookies["GSA_SESSION_ID"].Value;
      }
    StreamReader rsps = new
    StreamReader(request.GetResponse().GetResponseStream());
    strRsps = rsps.ReadToEnd();
    rsps.Close();
    response.Close();
  }
catch (WebException e)
  {
    if (e.Status == WebExceptionStatus.ProtocolError)
      {
        WebResponse resp = e.Response;
          using (StreamReader sr = new StreamReader(resp.GetResponseStream()))
            {
              Console.Write(sr.ReadToEnd());
            }
      }
    if (iRetry == 0)
      {
        //assume session timed out
        _gsaSessionId = null;
```

```
                iRetry++;
                goto Initiate;
              }
            else
              throw e; //if still fails, it might be some other cause.
          }

        return strRsps;
      }

  public static bool AcceptAllCertifications(object sender,
    System.Security.Cryptography.X509Certificates.X509Certificate certification,
    System.Security.Cryptography.X509Certificates.X509Chain chain,
    System.Net.Security.SslPolicyErrors sslPolicyErrors)
      {
        return true;
      }

    }
}
```