Mapping Security Requirements to Enterprise Search

By Mark Bennett
Chief Technology Officer, New Idea Engineering

Defining Specific Security Requirements

Introduction

Expanding the scope of search within an enterprise to enable employees and partners to more easily find data seems almost directly at odds with the security requirements that mandate precisely controlled access to that same data. Amid this turmoil, search vendors have remained uncharacteristically quiet on the subject. While they may offer a few buzzword compliant check boxes on their data sheets, public information about tightly integrating security with search is scarce.

Why the Increasing Interest?

The media has made the general public more aware of security, or more importantly, the high profile failures of security. Within the enterprise, corporate legal, IT, and PR departments have tried to protect their own companies by being more proactive, both in terms of technology and procedures. The government has also stepped in to add additional compliance regulations and penalties. Judges have even mandated the search enabling of archives as part of the discovery phase of large lawsuits.

Perhaps a more fundamental reason for the increased need for security is the large amount of data that is now being stuffed into corporate data stores and subsequently being "search enabled." For example, the amount of email stored inside corporations is growing, and there is a trend within companies to "throw the switch" and turn the corporate search engine lose on that data. Storage manufacturers are accelerating the amount of fully-indexed data by turning their products into "smart" devices where the data stored within can be searched directly, without the need for an external search engine.

As more data is indexed, there are more chances for sensitive data to be easily retrieved. If you don't believe it, try searching for the word "confidential" on your internal portal.

Turning the genie loose on these vast amounts of data does seem at odds, at least on the surface, with maintaining security.

Why the complexity?

Security is reasonably well understood for things like bank accounts, and shared file network storage and document management companies have extended security deep within their systems for some time now. So how hard should it be to add security to searchable text?

Part of the answer may be that, relatively speaking, search is still the new kid on the block, and companies are still climbing the learning curve. Some enterprises are still struggling to get basic search working system wide, and once they have it the next priority is inevitably "fixing relevancy."

As companies progress with search, they eventually start asking about security, as they rightfully should. The next phase usually involves the search vendor answering every question with, "Oh yeah, sure, no problem." In the extreme, the conversation degrades into an alphabet soup of abbreviations and technical terms that usually give managers headaches.

Here are some of the factors that can make search engine security a bit complicated:

1. A lack of structure in many document stores.
2. The complex structure of other document stores.
3. Existing and numerous security standards and products.
4. A complicated mix of homegrown security and/or legacy systems.
5. The need for the search engine "spider" to both fully access all data and to restrict results.
6. Dense search engine and security vendor vocabulary and product names.
7. Sparse vendor websites.
8. The difficulty of defining business requirements for various classes of users and

data, including an occasional requirement for "partial" access.

Defining Requirements

Though it may seem obvious, the first step in implementing security is to define the requirements with a particular company. This is not as easy as it may sound. Many clients haven't thoroughly done this and, as the design progresses, some unusual requirements may surface.

Security Granularity

Granularity refers to the precision with which particular pieces of data can be secured. Companies have very particular rules about which documents, or which portions of which documents, various users can see. This is, by far, where the most interesting business requirements come from.

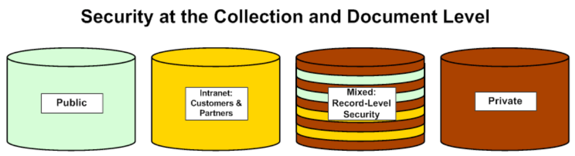
All or Nothing

There is an idea that if you are "logged in" or otherwise validated, that you can search for information. As an example, smaller companies may allow all employees on the internal network to search all of the indexed data sources. If you are logged in, you can search; if you aren't, you can't.

This had been the traditional model of usage, especially in smaller organizations, but it is becoming obsolete. Today, most companies have at least some public content and even non-authenticated users are allowed to see it. At the other extreme, most employees are not allowed to see financial and human resources files. Because of this, most companies have outgrown this security model.

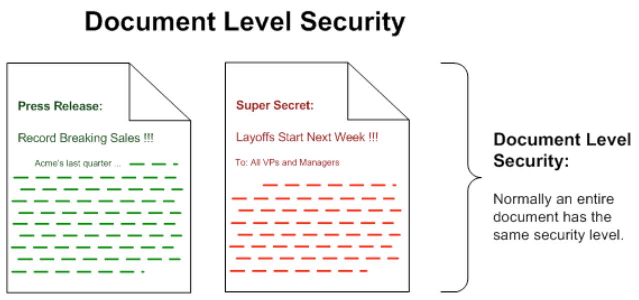
Search By Collection / Repository

One of the easiest and still reasonably useful control techniques is to simply segregate data by security requirements — public data is grouped into one section, restricted data into a second, highly confidential data into a third, etc. Most search engines support the concept of collections, which may also be referred to as "repositories," "sources," "document indexes," "spokes," or "document sets." Search engines typically allow each of these to be turned on and off in various combinations for each search. Once the credentials and access level of an individual user is determined, the appropriate collections are enabled for their search.


Security at the Collection and Document Level
Public
Intranet: Customers & Partners
Mixed: Record-Level Security
Private

Search By Document / Record

This method of securing data will feel very familiar to those with a database background --



certain groups or users can see certain documents. Databases and Content Management Systems have had this technology for a very long time and enterprise search engines are quickly catching up.

Conceptually, in the realm of full-text search engines the terms "record," "document," and "web page" mean almost the same thing: a retrievable unit of data. The specific terms used vary based on the background of the people working on the system or the physical source of the data.

Note: If you are relatively new to search engines and have a database background you might want also want to read "Contrasting Relational and Full-Text Engines" at http://ideaeng.com/pub/entsrch/issue09/article01.html.

Complexities of this Security Model

One of the complexities with this model is the rendering of the results list. Typically a document or record will be well secured, but the search engine has indexed all of the content and is engine indexed all of the content and is displaying lists of titles and summaries in a results list. It's not enough to secure the actual document; the results list should not display even the title or summary from a document that the user cannot see. Often even a title or summary can convey important information. This can be the first surprise a company has when it implements this level of security. For example, a title of "Indictment of John Smith Expected Tomorrow" tips off John Smith, regardless of whether he can read the entire or not.

A more subtle detail of the secured results list is the display of the number of matching documents and the links that allow users to page through a long results list. A simple engine might display the total matching count of documents, whereas a highly restricted user may only have access to 10% of those records, so the count is quite misleading. Beyond cosmetics, the engine needs to have an accurate idea about what documents that user can see when it is offering links to pages 2, 3 and 4 of the results list.

An even more subtle detail, but one which can still be a requirement in highly secure systems, is the confirmation of whether certain terms appear in the document index at all.

For example, searches for terms like "layoffs," "indictments," or the names of specific people can partially confirm the presence of information, even if no document titles are shown. A highly secure search will not confirm or deny the presence of terms in its index outside the context of what the user can search on. A more common example may be to not confirm the presence of obscenities or defamatory terms in non-accessible content.

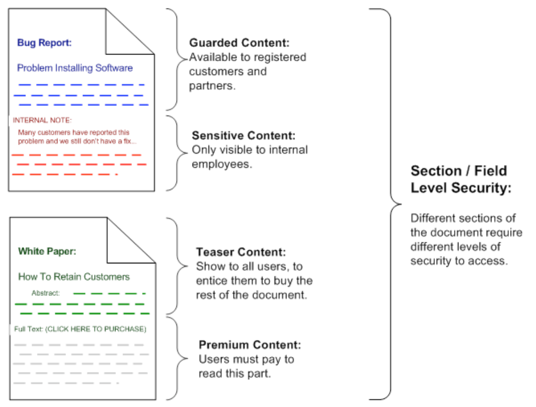
Search By Field / Subdocument

At this level of detail the design and implementation complexity starts to ramp up. The general idea is that different users can see different portions of the same document.

Some examples:

- All managers can see summaries of sales documents, but only VPs, Finance, and Sales can see the specific financial terms.
- Partners can see the text of bug reports, but can't see the company that the logged the issue.
- Sales Engineers can view technical design documents, but can't read certain proprietary details.

Sub-Document Security: Based on Well-Defined Fields


Bug Report:
Problem Installing Software
INTERNAL NOTE:
Many customers have reported this problem and we still don't have a fix...
Guarded Content: Available to registered customers and partners.
Sensitive Content: Only visible to internal employees.
White Paper:
How To Retain Customers
Abstract:
Full Text: (CLICK HERE TO PURCHASE)
Teaser Content: Show to all users, to entice them to buy the rest of the document.
Premium Content: Users must pay to read this part.
Section / Field Level Security:
Different sections of the document require different levels of security to access.

- Medical researchers can read legal cases, but not patient details.

This still sounds straight forward, but the implementation details can get a bit sticky. In the previous section we mentioned that, conceptually, "documents" and "records" are quite similar in the scope of search engines. However, from an implementation standpoint, subdividing a database record on field boundaries is much easier than subdividing a physical document, so when it comes to implementation, document versus record *does* matter.

Selecting only certain search fields from a database is fairly easy, but automatically detecting and removing certain parts of unstructured documents can prove difficult. If a set of documents was designed from the start for this purpose, tools like XSLT could be used to break them apart; in practice the search engine team inherits somewhat random sets of documents. In some cases formatting can be used to infer security context, but some document formats are harder to subdivide than others.

Typical ease of document subdivision:

- Database record: easy (via select statement or view)
- XML: easy (via XSLT)
- HTML: moderate (HTML is not always well-formed)
- PDF: moderate to difficult (depends on PDF format)
- Proprietary office documents: difficult (often requires a document filtering library and custom code or document conversion)

More open document standards are coming into use, and even Microsoft has plans to embrace them, so in the future subdividing documents should become easier.

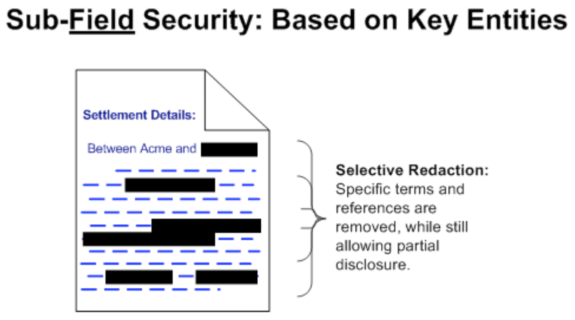
A Somewhat Odd Combination: "Title Teasers"

We've seen this implementation enough to call it out separately. Some sites that charge for content allow users to see the title of documents in their results list, and perhaps even a summary, but the user must then pay to see the entire text of the article.

This is a bit atypical because until the user has paid, they do not have rights to read the document. We said previously that results lists shouldn't even show a title if the user doesn't have rights to see the document, but this case is an exception. It could be viewed as a rather extreme form of field level security. The other oddity is that the users' access to particular documents can change if they decide to pay. On the implementation side, this may require some adjustments to the system.

Search by Sub Field

Vocabulary: Redacting. The act of removing very specific pieces of information from a document, such as specific words and phrases, or perhaps specific names and locations. The removed information may be represented by black-boxes, or removed entirely with no specific visual cue.


Sub-Field Security: Based on Key Entities
Settlement Details:
Between Acme and
Selective Redaction:
Specific terms and references are removed, while still allowing partial disclosure.

In some cases it is a requirement to restrict information at the sub-field level. For example, we've all seen news reports that show documents where specific peoples' names have been blacked out. In this case the removal of information isn't bounded by a neat field or document boundary; it involves removal of more specific words and phrases at a very fine level of granularity and control. In some respects this is an extension of sub-document retrieval; if a document is unstructured, then removing portions of it use some of the same techniques as sub-field removal.

And yes, search engines can even be coaxed into handling this type of situation. Remember, it is not enough to remove these terms from the actual document when being viewed; most secure environments would also stipulate that these words and phrases not show up in the results list titles or summaries. Furthermore, a really secure system shouldn't confirm that the removed words appear in the index at all.

Hybrid: Record AND Field

We realize that some of these scenarios sound like "overkill," but we have personally seen these requirements and worked to implement them at specific clients.

Moreover, some business requirements require a combination of one or more of the techniques mentioned above. Some data is all public, whereas other repositories have a document-by-document access model. Some documents have further restrictions within the document or fields. The organizations that spend money to implement these highly customized systems do so out of necessity; they need to share data in a very controlled way, but with the convenience and efficiency of a search engine.

These are not weird theoretical edge cases. Big organizations have a lot of data and a lot of folks who need to access it. In the past few years, as they have embraced search technology, their requirements have come along for the ride.

Levels of Users

This is the other side of the security equation. Generally this area is much more widely understood, and is about the same for search as it is for other systems. The details of implementation may present the only challenge.

Generally, users can be classified by:

Global Status

All users who can access the system, or are otherwise "verified," share the same security credentials. As with data, this "one size fits all" model is often inadequate.

The one exception where this model may make sense is for completely public services, where every piece of data is intended to be public and the search engine is not used for any internal data.

General Status

In this model, access is assigned by title or rank within the organization. Levels of access might include Partner/VP, Management, Employee, Customer, Public. A similar model could be adopted based on military rank or some other system.

Group / Role

In this model, arbitrary groups of users can be defined. Some of these groups may still be based on management level or rank, but roles such as "Human Resources" and "Finance" can be defined to allow some subordinates in specific roles to have access to additional appropriate data. Other examples would be allowing customer service personnel to access customer data, or grouping by a user's current workgroup, allowing them to easily share information with immediate coworkers.

Specific User

This model may be combined with the group model mentioned above. Security can be doled out on a user-by-user basis.

This model may be difficult to implement, depending on the method-specific search vendor used. As we will discuss later, the preferred "early binding" security filter method may be overwhelmed by the potentially enormous security filter this model may require.

A special class of "user" is often "self" or "owner." Almost all systems allow users access to their own documents and content, unless their job is simple data entry. This could be considered a special "role" or group.

These last two security models are commonly associated with Access Control Lists (ACLs), a long-standing security model. Data about specific users and groups may be implemented with Lightweight Directory Access Protocol (LDAP).

Implementing the system

Vocabulary Note

Although these terms can have broad meanings in the computer and software industry, our definitions here are specific to their use in relation to search engines.

Document Level Security

This is security that can be controlled at a document-by-document level. User A does a search and can find matching documents that he has access to. User B does the exact same search, but sees a different set of matching documents, which are the ones she would have permission to view.

ACL / Access Control List

This is a list of security permissions associated with a particular document or web page, and an electronic representation of who is and is not allowed to see the document. These permissions store the unique ID for each group of people who can see the document. It is also possible to store the ID for specific users (vs. entire groups) though this is less common and overuse can lead to inefficiencies. Some ACL systems also allow for a list of group and user IDs that are specifically *not* allowed to see a document; these "deny" lists typically override all "allow" listings.

ACLs are managed and stored in some other system, such as LDAP or Active Directory pages, or in a content management system.

LDAP and Active Directory

LDAP and Active Directory are standards for storing information about users, groups of users, and other company resources. LDAP stands for Lightweight Directory Access Protocol and is supported by many vendors. Active Directory is an alternative standard supported by Microsoft. Adapters exist to allow systems using the two different protocols to interact with each other.

CMS / Content Management System

A CMS is software that stores and manages large numbers of documents. Examples include Documentum, Microsoft SharePoint, Lotus Notes, and Vignette. Content from these systems is often indexed and searched by enterprise search engines.

SSO / Single Sign On

SSO is a network service that allows an employee to login once and then have access to all secured applications without the need to login again for each application. In order for search engines to implement security, they usually need to interact with one or more of these systems.

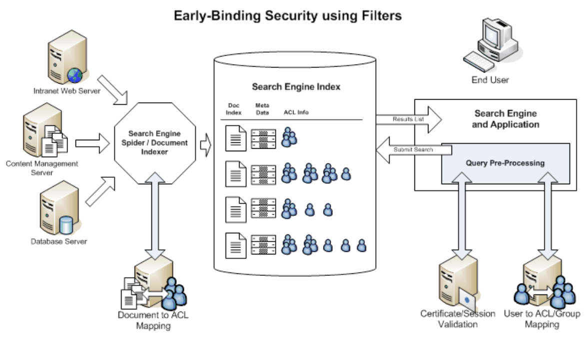
Two General Types of Implementation

Document level security, where each group can have access to different documents on a group-by-group basis, is the fastest growing segment of high end search engine installations. Document level security is used when the simpler application level security workarounds, such as collection level security, start to fail. To have different permissions for each document, you need to have some type of existing ACL system and/or SSO system in place and integration software from the search engine vendor to connect to it.

Early vs. Late Filtering

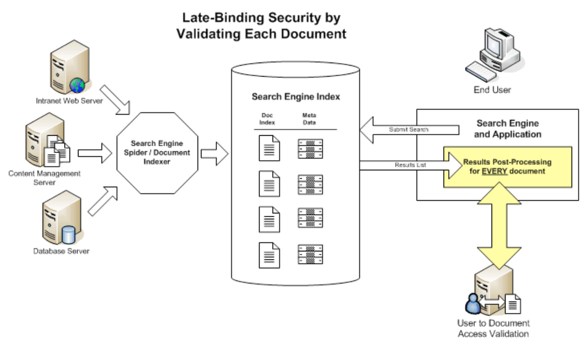
Although implementations are vendor specific, there are two primary designs for providing document level security: "early binding" and "late binding" document filtering.

"Early binding" document filtering is set up before the query is sent to the core search engine. Detailed information about the user's permissions is automatically added to the query that the user typed, just before the query is submitted, so that the core engine will only bring back documents that the user can access.


Early-Binding Security using Filters
Intranet Web Server
Content Management Server
Database Server
Search Engine Spider / Document Indexer
Document to ACL Mapping
Search Engine Index
Doc Index
Meta Data
ACL Info
Results List
Submit Search
End User
Search Engine and Application
Query Pre-Processing
Certificate/Session Validation
User to ACL/Group Mapping

Early-binding document security is often more complex to setup, but is strongly preferred since it should provide much better performance and avoid some odd display issues. If the underlying engine understands the user's security limitations, it will only return documents that they can see; time is not wasted gathering titles and summaries of documents that can't be seen. Since the filtering happens at the lowest level of the search engine, it should also happen much more efficiently.

"Late binding" document filtering handles document security after the search has been submitted to the core engine, while the results list of matching documents is being displayed to the user. Each document's access level is checked against the user's security credentials. The results list formatter will check every document against an external server to see if the user has access.


Late-Binding Security by Validating Each Document
Intranet Web Server
Content Management Server
Database Server
Search Engine Spider / Document Indexer
Search Engine Index
Doc Index
Meta Data
End User
Search Engine and Application
Submit Search
Results List
Results Post-Processing for EVERY document
User to Document Access Validation

Late Binding Security

Late-binding document filtering can potentially be very slow and can strain corporate security systems. Consider a relatively limited access user, who belongs to only one low privileged group. Let's assume, on average, that this user can only see 10% of intranet content. Since most engines show 10 documents on the first page of results, then on average 100 documents will need to be considered before 10 are found that are acceptable to show. So for every search by every user in this group, 100 documents will need to be checked.

Vendors have many different names for these two systems, so sadly you may need to do a little digging.

If early-binding security is so much better than late binding, why would anyone bother with late-binding? The answer is that from a technical standpoint late-binding was much simpler to design and implement and, until very recently, was much more common.

If you think about it, early-binding security requires much more up-front work. For each document, URL, database record, etc., its entire access details must be downloaded and stored into the search index. Getting the detailed ACL info for a document depends on how the document was stored. If a document is stored on a Windows file server, then Microsoft based security information for that file must be gathered; any reference to specific groups or users will be references to Microsoft domain groups and users. On the other hand, if the document was stored inside of Documentum, then that content management system must be consulted for user and group information. Those user and group references will be specific to the Documentum security database and may have no connection to Microsoft domain groups and users. In a large company, there can easily be a half dozen different document repositories, each with their own idea of "groups" and "users." Gathering ACL information from each of these unique sources and them mapping each to actual users and groups inside of a company is a complex task.

With late-binding security, a single question can be asked of any matching document and a user. A simple "yes/no" request is made to retrieve the URL of each document, and the user who issued the search has his credentials forwarded to whatever remote system hosts that particular URL. The remote system will either return the document or not, depending on the remote system's opinion of whether that user can see that document. From the search engine's standpoint it will get either a "yes" or "no" answer and decide to display or discard that document from the results list accordingly.

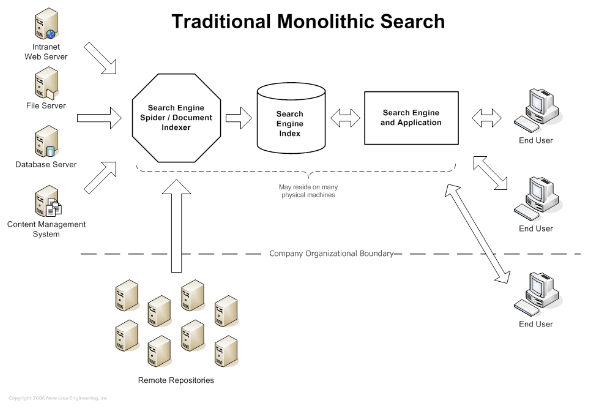
Problems to Look Out For

In this final section we discuss some of the issues that can arise when considering security. Things don't always go as planned, even during the design phase, and hopefully this will provide a "heads up" on what to look out for earlier in the process.

Non-Indexable Content and Federated Search

There is some conflict between wanting to include more and more data in a search engine's index, while simultaneously increasing the chance that even a trusted employee might accidentally gain access to highly confidential information that he is not supposed see.

In the last two sections we talked about how to satisfy both goals by implementing robust document level security in the main search engine. A core assumption in this design is that there is a central monolithic search engine. The design assumes that the engine will index all of the content into its own centralized search indices, and then perform filtering at search time. We refer to this as the "Über Index" design — one search engine indexing data from all of the repositories in the company, even the ones with sensitive data.


Traditional Monolithic Search
Intranet Web Server
File Server
Database Server
Content Management System
Search Engine Spider / Document Indexer
Search Engine Index
Search Engine and Application
End User
End User
End User
May reside on many physical machines
Company Organizational Boundary
Remote Repositories
Copyright 2006 New Idea Engineering, Inc.

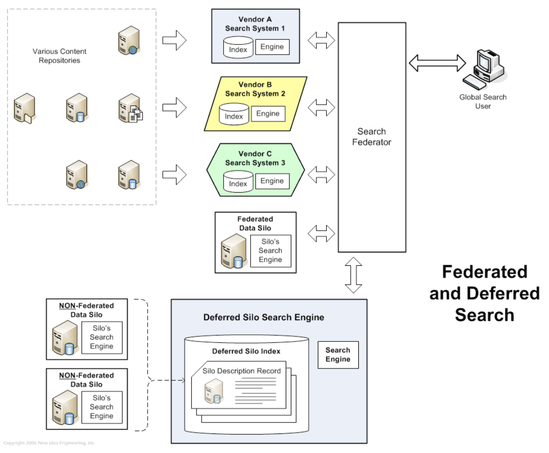
In some organizations this is simply not feasible. There may be technical obstacles such as a repository that no vendor directly supports, or that lacks an export or web interface.

Business-related issues, too, can present intractable roadblocks, such as organizational boundaries or very tight security policies. For example, some groups may not want to provide the spider with a login that can access all of their content, even if they are promised that data will be filtered reliably at search time. Or, since a central search would technically be duplicating their secure data onto another computer in some format (cache files, binary search index, etc), that might violate a security policy that mandates the data remain solely in the original secured place.

For the latter business-related obstacles, a thorough explanation of how search security works might change some minds, but certainly not all. Implementing Federated Search might be a palatable alternative.

Federated Search allows a user to enter a search into a single web form, but get back combined results from multiple separate search engines. When security is a factor, the authentication credentials of the user that are passed into the original system must be forwarded on to the remote search engines, so that they can also enforce document level security on their end. The advantage of this model is that from the remote and highly secure search engine's view, the search can be treated just like a search submitted by a user directly to that engine. If the credentials are wrong, no results are returned; the remote search engine maintains control of its own security, regardless of where the search originated from.

Matching URLs or records from the remote engines can be combined with the results from the central engine into a single list of results. However, there may be technical and design issues involved with doing this. Another option is to keep the results from the various engines in separate parts of the results page, either in separate tiles, different section headings, or on separate tabs.



It is possible that some remote search engines will not accept federated searches for either technical or policy reasons. If there are a number of non-federated sites like this, the sites themselves can at least be listed as suggested sources of data if the description of the site contains matching terms.

As an example, if the finance department doesn't want financial records searched remotely at all, either by über-indexing or by federated search, a description of its site could at least be included. In this example, a user issuing a search for "budgets" would not get any specific finance document back, but they would see a suggestion to visit the finance department's web site because the site's description included terms related to "budget."

Securing All the Links

Note: This section may be a bit terse for some folks; please email us if you need a more detailed explanation.

Search engines interact with many other systems and users over the network, usually via TCP/IP sockets. Hackers often try to monitor these communication channels so highly secure environments will need to encrypt or protect these communication channels.

Some of the typical socket traffic would include:

- From the Spider to the Repository
- From the Client to the Application Server
- From the Application Server to the Search Engine
- From the Search Application or Engine to data logging services, such as when searches are being sent into a Search Analytics Database table

How Are Links Secured?

Detailing this is beyond the scope of this article, but we can at least point you in some specific directions.

A common way to secure sockets is by changing from http to https, using SSL, or employing other encryption techniques.

If it is too difficult to secure all the sockets, machines could be organized into a secured subnetwork or DMZ, protected by a firewall.

Traffic from other sources might be handled via a proxy server or "reverse proxy server." One open source resource that can help with some of these tasks is the Squid caching proxy server from the Apache group (http://www.squid-cache.org/).

Data Stored on Disk

Securing the Search Index

Every search engine stores information about the source documents it has indexed or spidered in some type of local database as a series of large binary files on disk or network storage. These files contain titles, summaries, and a catalog of all the words in the source documents. Some systems even include large snippets of the documents' text, or may have even cached the entire document itself. Even a search engine that *only* contains word instance information is still technically dangerous – it is possible to partially reconstruct source documents using only word instance information. This is not science fiction. It can be and has been done.

We suggest locking down any servers that have access to these disk files. There is talk of creating encrypted search indices or solutions that will encrypt an entire file system, but we worry that this will slow down performance and complicate implementation efforts. This practice is certainly very rare.

Scripts, Logs and Stored Passwords

As with many other software products, the startup scripts, indexing scripts, and spider configuration files can contain passwords. Some vendors do support storing hashed passwords so that at least the passwords are not stored as plain text. However, as with search indices, these disk files should only be available to machines that are tightly secured.

Results List Gotchas

There are a number of things than can thwart your security designs.

Titles and Summaries can be a Leak

As we warned about in the first section, results lists should not reveal anything about documents that a user isn't allowed to see. Securing the document itself, but still showing titles and summaries to a user is a real security breach and it is worth repeating the warning here.

Navigators and Statistics Can Reveal Sensitive Items

As another reminder from the first section, results lists often show how many matching documents there are, or how many documents contain each search term or provide clickable hyperlinks to drill down into the results.

As an example, "layoffs" is a very loaded term; even if an employee doesn't get any documents listed in his results list for that search, a "helpful" navigator that confirms the existence of documents with that term, or worse, how many documents have that term, is still a security breach. An employee might assume that layoffs are, in fact, on the way!

This level of security may not be easy to reach with some vendors. Please check with your vendor carefully if this is a concern.

Highlighted Document URL Linkage = CGI Back Door

There are some very handy and innocent-looking features in results lists that can sometimes be hacked and used to bypass security.

For example, when a user clicks on a document in a results list, many search engines open up the document and show the search term highlighted *within* the document, instead of sending the user to the original document URL. This is referred to as Document Highlighting, which should not to be confused with highlighting search terms in the results list's document summaries.

Related features include the ability of some search engines to convert various document formats into HTML by offering some type of "View as HTML" link. Also, some search engines may fetch matching records from a database, and display them to the user.

In all these examples the search engine is accessing the source document again every time a user wants to view the document, long after indexing is complete. More importantly, these are usually implemented by using clickable URLs that point back into the search engine. These URLs can be edited to access *other* documents, ones the user should not have access to. In other words, even if a user doesn't see a secured document in the results list, she can copy one of these utility URLs to an editor, change the document ID, and then paste the modified URL back into the web browser. If security has not been implemented properly, the search engine will obey and retrieve and display the blocked document.

More modern systems are aware of this type of trick and these secondary links are also validated against the user's credentials. Older systems might use a single "super" login to fetch documents for highlighting, enabling hacking.

In reality, we've never seen a user actually do this, even on the older systems. Search engines seem rather complicated to most users and editing URLs takes a bit of technical skill, but thorough security doesn't rely just on "security through obscurity."

Runtime "Super" Login

Some older search engines had one or more super logins for the runtime search engine which was used at search time, not just at index time. If your system requires this type of login, please re-read the previous section carefully and make sure you understand it.

Admin Gotchas

Due to the default installations of some search products, it is important to double check that administration portion of your system does not introduce security holes.

Secure the Admin!

It seems amazing, but some search engines' default installation brings up an administrative service with no password! In reality, the software is usually on a private network so this practice is slightly less dangerous than it sounds. Still, there is a tendency to forget to correct this, and a year later it may still be unsecured.

Secure the Search Analytics and Business UIs as Well

Also, many search engines have more than one administrative UI. They may have a UI for IT, another UI for business owners, or perhaps even a third UI for running reports. These should also have passwords.

Capturing User Info in Search Logs

From a technical standpoint, it's nice to have information about which user did what search in the search logs. By tracking the ID of each user's search, a Search Analytics package can show trends on a per-user or per-group basis.

However, some sites may have security policies that forbid this type of data gathering, so those sites should make sure to disable this feature. Also, some government jurisdictions may place restrictions on tracking user activity.

A gray area for sites concerned with privacy is that the TCP/IP address of the computer doing a search is often tracked. If the computers have fixed IP addresses or tend to get the same address when they are rebooted, it might still be possible to track searches back to a particular user. From a reporting standpoint this is handy, but again it may violate a company policy or government regulation.

Conversely, Not Capturing User Info in Search Logs

Assuming there is no policy or law forbidding the logging of employee or customer search activity, then it should be properly logged. Not doing so could cause problems later.

For example, suppose the Tech Support manager notices a sudden cluster of searches in the reports, such as "crashing," "software crashing," "software crashes," "core dump," or "your software sucks." It's likely these all came from the same frustrated customer — but which customer? If that info hasn't been logged, then the manager can't proactively provide relief or account management.

Similarly, perhaps an HR manager suddenly notices searches such as "sexual harassment," "sexual harassment policies," or "reporting sexual harassment.” Clearly some employee seems to have some concerns or questions about this subject but has not specifically come forward to report anything. If the HR manager knew who that employee was, he might want to start some preliminary investigations.

Although we are not lawyers, we’ve seen some recent sexual harassment rulings that seem to center on whether or not a victimized employee reported the abuse to management, the implication being that if the employee did *not* report the abuse, then the company should not be held liable for failing to address it. How can an employer fix a problem that it doesn't know exists?

We speculate that at some point in the future a court might decide that sexual harassment-related queries submitted to the HR site were in fact a means of "reporting" the problem. If that were to happen, then a company might become liable if it failed to notice the searches and took action. Being able to trace these searches back to a particular user may become a legal requirement.

Raw Search Logs or Search Reports Could Reveal Sensitive Data

We touched on a related area earlier, but if a casual user of the reporting toolkit were to suddenly see a lot of query activity about a layoff in the search logs, she might infer that a layoff is coming. Seeing search terms doesn't always mean that there was matching content, but it can certainly infer it. Some engines will confirm how many documents matched. If an analytics tool were to report that it was the CFO searching for layoff material, and did he in fact get 150 matches, a report user would have even more angst.

User Info

If an identifiable user has many searches for a particular subject that might be embarrassing, an inconsiderate coworker might let others know. For example, an employee might be looking for information in the HR database about policies related to sexual orientation or substance abuse treatment programs; a co-worker viewing those searches in a report might be very surprised by this and have trouble respecting confidentiality.

Ping / Sanity Checks

As part of ensuring the search engine is running correctly, we suggest that clients run an advanced "ping" script to periodically run a known search and check the actual results.

This is a good idea, but if not properly filtered out it will add a lot of bogus entries to the Search Analytics Reports.

Spider / Indexing Gotchas!

Because a search engine will likely be indexing *all* of your data, it is imperative to fully understand all of the security ramifications.

Spider and Repository "Super" Logins

If the sites your spider has access to require a login, then your spider will also need a login. Unlike typical user logins, the spider's login will have complete access to all of the content. This super login must be treated carefully and should be clearly disclosed to repository owners.

Detecting a Failed Page

The HTTP protocol clearly defines error codes that should be returned when a requested page cannot be accessed. The page may no longer exist or perhaps a login is required. However, web servers do not always use these codes, or do not make it clear to the spider that an error has occurred or that a login is required. Even if your spider has a valid login for the repository it is trying to crawl, you may need to help it understand when that username and password needs to be sent. A symptom of such a problem is noticing that the results list has titles and summaries that talk about logging in or instructions about resetting your password, etc., rather than having the titles and summaries for the real documents.

In the early days of the Internet, when a web server wanted a user to login, it would send an HTTP "challenged response" error code of 401 or 402. For a human operator, a small separate popup dialog box appeared, and they were asked to enter a user name and password, and sometimes also a "realm" or "domain." A 401 style challenged response is easy to recognize because there is a separate popup window in the browser that is clearly not part of a normal HTML page. Spiders generally do understand this type of response and handle it correctly. However, more modern sites often don't use this return code. Instead they want to provide the user with a full web page that explains the problem, and the web page will include the login username and password boxes. This is the type of situation that spiders often have trouble with; they don't understand that his is NOT the page they just requested, and therefore it is treated as a regular document.

False HTTP status 200 Codes

The most annoying version of this problem is when a server returns an HTTP success code of 200. The server returns an HTML page with an error message or a login form, but sends a return code of success.

It is our very strong opinion that this practice violates the RFC protocol for HTTP, or at least its intent. The requested page was not returned, but the server has reported 100% success. If there were an error, the server should be returning a 404 or 500 series error. If the user needed to login, then it should have returned a 401 or 402, or at least redirected to login page by sending a 300 series error code.

If you are trying to spider a site that returns OK/ 200 for failed requests, the administrator of that web site should fix it — if it is within your company or organization, there is some chance he might listen. However, if you have no influence on "false-200" sites, you will need to modify your spider to actually look for the login or error text in the HTML that is returned. Spiders don't usually have an option for this, so you may need to speak with your vendor.

If you will be indexing a lot of public sites, you might want to consider creating some custom spider logic that looks for these patterns by default and takes appropriate action. This type of system could even be designed to read bad phrases from a file or database, so that non-programmers can easily update the lists. This would also be a good way to filter out other types of bad content:

- From above, "login" required / login forms
- "squatter sites" — domains that have not been registered and are "for sale")
- Sites that are "under construction"
- Objectionable or offensive content
- Sites that use frames
- Sites containing no actual text

- Sites that require JavaScript, Java, Flash or some other technology to view the site
- Unsupported user agent errors
- Unsupported "referer" *[sic]*
- Unsupported HTTP or browser versions

Many of these sites will also report a misleading status of 200 to your spider.

Redirects to login pages

A somewhat easier problem to detect and fix is when a site requiring a login redirects the user to a login page. A return code in the 300 range is returned in the HTTP header, along with a "location" header field. Redirects are very common even for pages that don't require a login, so just looking for the HTTP 300 series return codes won't distinguish good pages from bad ones alone, but it's a start.

Some spiders support "forms based login." When these spiders get a redirect, they check the target of the redirect (the new URL is sent back in the location field of the HTTP header). They check this new URL against a list of known login forms. If the new URL points to the login form, the spider understands that it needs to login; if the spider does not recognize the redirect as a login form, it treats it like a normal redirect and attempts to fetch that new page.

Spider Revisiting Orphan Links

An "orphan link" is a web page that still exists, but is no longer linked to by the main site. If a user had bookmarked the URL for the page, then he could still get to it; but a new user starting at the home page would not be able to navigate to it. This typically happens when a webmaster unlinks content on the web site; she decided, for example, that an area of content is obsolete and removes all hyperlinks pointing to that section of the site. In the webmaster's mind, this content has now been effectively removed from the site, even though the specific files have not been deleted.

If a new spider were to crawl the site for the first time after the links had been removed, the spider would never see those pages and would not index them. However, a spider that crawled the site before the content was unlinked would still have a record of those pages and the URLs.

Some spiders will revisit these pages on an individual basis by URL, regardless of any changes to the links to those pages. Since they already have the URLs, and since those URLs still work, the spider will continue to index this orphaned content. A spider that operates in this mode is often referred to as an "Incremental Spider." Generally, revisiting each page individually is an advantage, because the spider can give more attention to pages that have been frequently changing, and only occasionally visit pages that almost never change. This issue of orphaned links is one of the few downsides to these Incremental Spiders.

This is not the case with all spiders. Older spiders tended to start at the top of a site each time they ran, and reindex everything from scratch. Those spiders are generally referred to as "Batch Mode Spiders" or "Non-Incremental Spiders."

To force orphaned content out of your search index, you will need to take one of the following steps:

1. Actually remove the content, page by page, from the web server or repository.
2. Specifically remove each URL from the spider's link database — this may not be possible with some spiders.
3. Start a completely new spider of the site, one which starts with a completely empty links database. Look for options like "Full Reindex" or "Clear Collection."

There is one other related orphaned content problem worth mentioning, though not directly related to security. In rare cases, a webmaster accidentally unlinks content. This also creates orphaned content. If the web site uses an older spider, the number of pages in the search index will drop dramatically; hopefully the site will notice this large drop and fix it right away. However, if a site accidentally unlinks content but is using a newer incremental spider, the spider will mask the mistake because it will continue to access the orphaned content by URL. On the surface this seems like an advantage, but this site now has 3 problems:

1. There is content that users can longer navigate to.
2. The problem is unknown and may not be discovered for some time. Since the spider is incremental, the search engine has no dramatic change to report.
3. At some arbitrary point in the future, when the site is reindexed from scratch, the page count will drop dramatically and there may be difficulty figuring out why. Since the content was accidentally orphaned months or possibly years ago, versus more recent changes which have nothing to do with the problem, the focus may be only on more recent changes and not the real cause.

Spider Used File System Access, Got Unintended Files

Sometimes a company will decide to have the spider crawl a file system, such as a server's hard drive, to find documents to index. These documents are often also available on the web, so the document has both a file name and a URL, and the spider understands how to map one to the other. There are various reasons for wanting to use file system indexing instead of web indexing, including performance, but the details are beyond the scope of this article.

Since the spider can see all the files in every subdirectory, it will want to index all of them, regardless of whether or not those pages are linked to by other pages on the web server side. As an example, an author may have several versions of a document. Only the final copy is linked to on the web server, but when file indexing is done, all 3 versions show up.

Another potential issue is that the web spider may have only been looking for HTML and PDF files, but many file system crawlers will also index Excel spreadsheets, Microsoft Word documents, Access databases, etc. by default. Those files are much more likely to contain sensitive information that was never intended to be published in any format. If you use file system indexing, you should run a report by Mime-Type, to make sure suspicious files have not been accidentally included.

File system indexing can also uncover entire directory trees, and suddenly thousands of forgotten files show up in the search index. If you are using file system access, check for unwanted files.

Spider Activity vs. User Activity

Many other systems within a company also log and track access to the documents they contain. This should not be confused with search analytics logging; here we are talking about the logs that *other* systems maintain to track the documents users are looking at. The spider will appear to those remote systems as a user, and they will likely log the spider's activity as well. Therefore, it is good to have the spider identify itself when requesting pages from other servers so that this activity can be interpreted differently.

One way to flag spider activity in systems that track use accounts, such as a CMS repository, is to give the spider its own special login. Administrators will know that it is normal for the user "speedy-spider" to be reading thousands of documents.

For spiders indexing generic web servers, the easiest way to flag spider activity is to set the "User-Agent" field in the HTTP headings option of your spider configuration. The User-Agent field can even include contact information in case there is a problem. For example:

User-Agent: Internal Search Engine Spider, contact Satish at x4123

A webmaster investigating unusual access patterns will see this in his log files.

Spider "HEAD" Command and Netegrity / Site Minder

In short, the HTTP protocol supports many request types, the two most common being GET and POST. Users will normally only use those two, and therefore in some cases these are the only two request types allowed by Netegrity by default. However, some spiders use the HEAD request type. If your spider uses the HEAD command, and if you use Netegrity or other SSO solutions, you should double check that this has been enabled.

Summary

There are many potential security holes that need to be double checked as you deploy an enterprise search engine. We've listed the most common and important, but we're always happy to hear your thoughts.

New Idea Engineering helps companies make search work right. We focus on search best practices to help companies select, design, and deploy advanced enterprise search applications. Our methodology includes search 2.0 interactivity, periodic review of search activity and ongoing search data quality monitoring to ensure great relevancy and user satisfaction. To contact us, call 1-866-IDEA-ENG or see our website for more information at www.ideaeng.com.