Secure Search in Enterprise Webs: Tradeoffs in Efficient Implementation for Document Level Security

Peter Bailey
CSIRO ICT Centre
GPO Box 664
Canberra ACT 2601 AUSTRALIA
+61 2 6216 7055
Peter.Bailey@csiro.au

David Hawking
CSIRO ICT Centre
GPO Box 664
Canberra ACT 2601 AUSTRALIA
+61 2 6216 7060
David.Hawking@csiro.au

Brett Matson
Funnelback Pty Ltd
401 Clunies Ross St
Acton ACT 2601 AUSTRALIA
+61 2 6229 1710
brett@funnelback.com

ABSTRACT
Document level security (DLS) – enforcing permissions prevailing at the time of search – is specified as a mandatory requirement in many enterprise search applications. Unfortunately, depending upon implementation details and values of key parameters, DLS may come at a high price in increased query processing time, leading to an unacceptably slow search experience. In this paper we present a model and a method for carrying out secure search in the presence of DLS within enterprise webs. We report on two alternative commercial DLS search implementations. Using a 10,000 document experimental DLS environment, we graph the dependence of query processing time on result set size and visibility density for different classes of user. Scaled up to collections of tens of thousands of documents, our results suggest that query times will be unacceptable if exact counts of matching documents are required and also for users who can view only a small proportion of documents. We show that the time to conduct access checks is dramatically increased if requests must be sent off-server, even on a local network, and discuss methods for reducing the cost of security checks. We conclude that enterprises can effectively reduce DLS overheads by organizing documents in such a way that most access checking can be at collection rather than document level, by forgoing accurate match counts, by using caching, batching or hierarchical methods to cut costs of DLS checking and, if applicable, by using a single portal both to access and search documents.

Categories and Subject Descriptors
H.1.0 [**MODELS AND PRINCIPLES – General**]; H.3 [**INFORMATION STORAGE AND RETRIEVAL**]; E.5 [**DATA – Files**]

General Terms
Algorithms, Theory, Security, Design, Experimentation, Measurement, Performance.

Keywords
Document level security; collection level security; access control; enterprise search; caching; performance; scalability; file systems; security models.

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

1. INTRODUCTION
As a number of authors have observed, enterprise search is different to search on the Web at large [4, 6, 9]. One of the demands of real world search in enterprise environments, and increasingly in large scale Web properties (such as Yahoo! 360 [http://360.yahoo.com]), is to respect the access security of individual documents and information when showing results for a search query [8]. "Document level security" is a phrase intended to capture this fine grain level of access control. (In database circles, this concept is referred to as row, column or field level security.) It is commonly distinguished from collection level security in which access is granted or denied to an entire set of documents at once. The latter is commonly applied to a complete repository or some easily defined subset (such as directory hierarchies).

This paper makes four main contributions. First, in Section 3, we describe a model of secure search in a document level security environment. The widely varying security environments that may be encountered make a general purpose solution more complex. However, these break down into two basic architectures, which are illustrated.

Second, although the solution to the problem is not conceptually hard, it appears not to have been documented in any depth previously. We provide a specification of the solution in terms of a pseudo-code algorithm in Section 4.

Third, we describe our experiences in Section 5 with two implementations and the lessons learned from implementing secure search in real world environments. In particular, we detail the interaction between user expectations about search and the mechanics of a solution's implementation.

Fourth, in Section 6 we carry out experiments and analysis to explore the model. These allow us to make a number of conclusions and recommendations on how best to provide efficient secure search in Section 7.

2. RELATED WORK
To be clear about the task: in response to a user's query to a search engine, a page containing search results is shown, such that only those matches from the result set which correspond to the user's rights to view each match are included.

Various papers mention that this is an intrinsic problem for enterprise search systems [2, 4, 6, 9].

One of the earliest and most comprehensive enterprise search systems to support search that respects document level security is Verity's enterprise search engine. The latest version is Verity

K2 Enterprise (K2), and in a white paper their security model is described in some detail [10]; there is also a rather higher level overview in [1]. Effectively, users must authenticate themselves with the K2 system by providing log-in information, and the user is provided back a "ticket". As far as we understand from the white paper, the system then associates this ticket with the user's authentication to the K2 system itself, and to other LDAP, Windows NT, UNIX, or other secure repositories' security systems which Verity is indexing. These repositories are intermediated by a K2 Gateway interface, which permit the K2 Server to cache access control lists (ACLs) for each document provided by the repository. Results that are limited by document level security can then be filtered by checking the user's credentials against each document's cached ACL. No information is provided about how K2 maps user credentials (if the log-in information is not identical across different repositories), represents ACLs for systems that do not provide statically checkable security, or updates ACL caches.

Other commercial enterprise search systems (e.g. Convera's RetrievalWare product [http://www.convera.com/], Coveo Enterprise Search [http://www.coveo.com] and Google Search Appliance [http://www.google.com/enterprise/gsa]) claim to provide support for document level security on their web sites. Coveo's solution is based on various Microsoft technologies (including file shares, IIS, SharePoint, and Exchange).

According to [5], Google's Search Appliance supports a range of authentication facilities, including HTTP basic authentication and/or NTLM authentication, as well as integration with form-based single sign-on systems. These forms of authentication work by having the Search Appliance masquerade as the user. Alternatively, users can provide X.509 client certificates to the Search Appliance to authenticate directly with it. Finally, custom security systems, possibly built on top of LDAP, can be used with a data repository, by developing an Access Connector in compliance with Google's Authentication SPI (which in turn is written using the SAML 2.0 specification). The Access Connector intermediates between the data repository and the Search Appliance's presentation of user credentials. Authentication credentials are cached by the Search Appliance for the duration of a session. At result presentation time, individual documents are checked by the Search Appliance communicating through an Authorization SPI, which verifies whether a user can view the document. The Authorization SPI is provided by an Access Connector system developed for each repository, and is used regardless of authentication method.

Coveo also provide an extensive information article relating to document level security, and describe the basic approach they use [3]. As with Verity's K2, they extract the access lists for each document (including emails etc) that they index, and cache the results. At query time, the user's credentials are then checked against the access control lists, and filtered to remove those that do not match.

In a related example, IBM's WebSphere Portal Engine enables secure search over its DB2 Information Management Software which provides facilities to associate security tokens with individual documents [7]. These remain associated with the document through parsing and indexing processes. However, they provide no details on how the search system works to match the user credentials with the security tokens at query time; it is possible that high level search adapters are provided, in a similar fashion to the Verity Gateway mechanisms. Similarly, Microsoft's SharePoint portal software supports document level security using its Rights Management Services [http://www.microsoft.com/windowsserver2003/technologies/rightsmgmt/default.mspx].

Commercial organizations almost never provide full detail of the mechanics or performance of their secure search implementations. Google's recent documentation is a first for extensively defining the security API and process by which the Search Appliance interacts with the customer-implemented Access Connector. In our review of the literature, we have been unable to find additional information describing how search is integrated with document level security.

2.1 Other environments with document level security

Search engines on the Web at large (e.g. Google, Yahoo!, MSN) typically avoid any attempt to index restricted accessibility documents, since the performance impact in reporting secure results to even tens of thousands of users, let alone hundreds of millions, is too large, and the complexity of managing user authentication too challenging. (Note that some websites present different information to major search engines even though their information is not publicly accessible. This technique is a form of web cloaking. Such content is discoverable through search, but not directly by a user attempting to access the source material.)

Personal search engines (such as Google Desktop Search, Copernic, or Microsoft Windows Desktop Search) work by indexing only content that is accessible to the user. Thus security is implicit since content is not indexed that could not otherwise be found.

3. MODEL

We now introduce a model to capture the properties of secure enterprise search. This model is used to understand how an algorithm and implementations must operate, and forms the basis for analysis of performance.

3.1 Definition of terms

There are some terms which are used consistently throughout the paper; we provide our understanding of them here to ensure clarity.

- *User authentication* – the means by which an individual presents their credentials (for example, login name and password) to gain access to some protected system.
- *Collection* – a set of documents indexed by the search engine.
- *Document* – any information that is indexed as a unit by the search engine. Examples include emails (with or without attachments), individual files in a file share, records within a database, and web pages.
- *Repository* – a collection and an associated information management system that provides additional services, such as automatic metadata management or access control. Examples include content management systems, databases, but also file shares (with associated operating system mechanisms for security and access).
- *Security permissions system* – a computer system which determines whether a person is permitted to access

individual documents. A specific document's access rights are referred to as its *security permissions*.

- *Access control list* – one form of security permissions which may be copied from a document to a copy of the document and otherwise queried by a third party system. Not all security permissions systems support being represented as a set of access control lists.

3.2 Architectural issues

We assume the following environmental parameters for this discussion:

- there is an unknown (and possibly large) number of users;
- there are many documents;
- different security exists over some subset of the documents, which means not all documents are to be visible to all users;
- a single search engine[1] is in use;
- document level security can be determined with respect to a user's credentials;
- changes to security must be respected, within some specified interval of time.

The approach described in this paper is most commonly encountered in enterprise information systems with one or more repositories but with a single mechanism for determining user authentication to the repositories being searched. User authentication credentials are then matched against security permissions on a per-document basis to determine whether a document can be viewed or not by an individual user.

A fundamental premise is that the search engine must be omniscient (with respect to the collections) – that is, it must be able to fetch and index every document available. This premise critically affects security risks with respect to accessing the search engine, as uncontrolled direct access (without authentication) to the search engine must be prevented.

There are two basic architectures which may be adopted:

1. the search engine knows nothing about security as implemented within each repository; or
2. the search engine must know almost everything about security.

The first architecture is useful when existing repositories are managed by corresponding sophisticated information management systems (for example, a content management system, with its own in-built security model and policies). In this case, the repository is responsible for managing document level security, and treats the search engine as a black box which provides high quality ranking of documents irrespective of security – see Figure 1. The repository may subsequently impose additional sorting or re-ranking of results, based on additional information it has about each document.

[1] Of course, multiple search engines may be in place within the enterprise, and individual repositories may have their own inbuilt search engines. However, for this paper, we assume a single search engine (or strictly speaking, search engine interface). It may provide federated search over other search engines where required.

Care must be taken to prevent the repository from accessing any information from the search engine about collections other than those corresponding to its own, and from users accessing the search engine directly (at least for this collection). In other words, the repository acts as the user interface to the search engine. Non-enterprise environments where this architecture is in use include general Web portals with personalized content (for example, Yahoo! 360).

The second architecture works best in environments with single sign-on to all repositories, using directory services (such as LDAP or ActiveDirectory) to provide user authentication and security policy services. In this approach, the search engine holds the user credentials, and either caches the access control lists associated with each document at crawling/indexing time or queries corresponding repositories when preparing a result set to ascertain the rights of the user to view each potential document in the result set – see Figure 2.

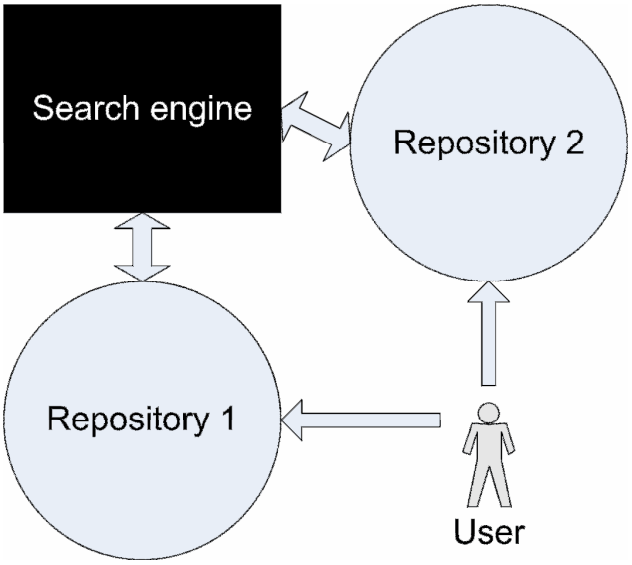

Figure 1 - Architecture 1 : search engine accessed only via repositories, knows nothing about user security

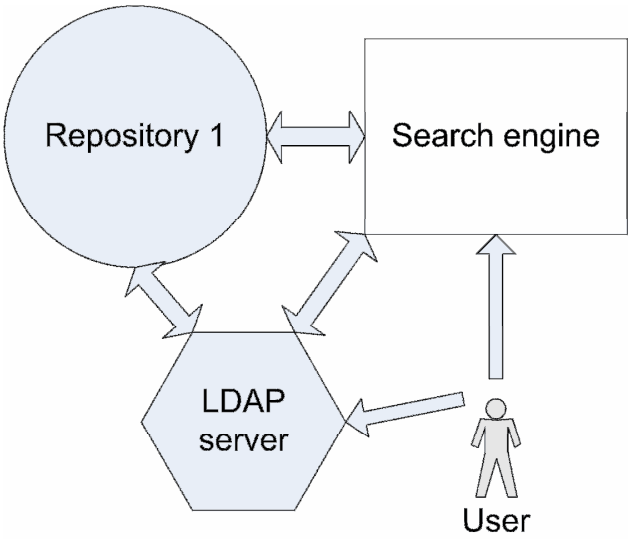

Figure 2 - Architecture 2 : search engine intermediates repositories, knows about user security

In both cases, an algorithm is employed (see Section 4) which filters the results based on the user. The algorithm takes into account paging into results sets beyond the first page, and there is a discussion about the security implications of exposing parameters through URLs.

3.3 Parameters

There are a number of key parameters which affect an implementation of a search solution.

1. *Acceptable time (AT)* - the acceptable time for delivery of a page of results, post the submission of the user's query. The experience of people with Web search engines means that their expectations have increased such that any time longer than a few seconds (5-10 at the utmost limit, and preferably less than 1-2 seconds) will be considered to have failed, and they will start to click the browser refresh button. In certain environments however, it is possible that a fully accurate result (for example in a legal firm retrieving all possible precedents) is critically important, and minutes can elapse to carry out this search correctly.

2. *Number of results (NR)* - the number of displayable results which must be calculated. This number could be only those required to show one page of results. Alternatively, it could be all possible results, in scenarios where an accurate count of the number of matching documents must be reported. Clearly, the more results required, the greater the time taken to produce the page due to the increased number of security checks to be performed.

3. *Average security check time (ASCT)* - the average time to check whether a document's security permissions allows it to be visible to a user.

4. *Visibility density (VD)* - the approximate density of documents visible to a normal user for a representative query (expressed in the range 0..1). In other words, is the information environment one in which most documents are visible to most people (common in our experience in many enterprise intranets) – a *VD* of close to 1, or one in which most documents have highly restrictive access rights (for example, in security agencies) – a *VD* closer to 0.

5. *Overheads (O)* include sending the request to the server (including any user credentials), performing the search, building the page output, and returning the page to the user. It may also be used to model scalability – that is, the need to serve multiple search requests (to other users) within any acceptable time period.

6. *Staleness (S)* - the acceptable level of staleness of the security used when checking access rights for a user. The level can range from none (i.e. security must be up to date and immediate) to a number of days. The greater the degree of staleness, the more that caching techniques can be used. Conversely, if no staleness is acceptable, caching can only be used within the request calculation itself.

A useful secure search engine should satisfy the following inequality:

$$AT \geq \frac{NR \times ASCT}{VD} + O \quad \textbf{(1)}$$

Reformulating this, we get:

$$ASCT \leq \frac{(AT - O) \times VD}{NR} \quad \textbf{(2)}$$

Or:

$$NR \leq \frac{(AT - O) \times VD}{ASCT} \quad \textbf{(3)}$$

For example, if the *acceptable time* is 1 second, *overheads* make up 0.25 seconds, the *number of results* is 10, and the *visibility density* is 80%, the *average security check time* must be 0.06 seconds or less.

Note that this analysis only deals with average case scenarios. It is very possible that worst case security scenarios (e.g. where either the *average security check time* or the *visibility density* for a particular query or a particular user) could cause the *acceptable time* to be exceeded.

Also missing from this analysis is any performance scaling with regard to the number of users accessing the system simultaneously. However, the *overheads* element can be considered to capture part of this issue. In addition, search is intrinsically a scalable problem in that more search servers can be added for meeting more user queries simultaneously.

The level of *staleness* which is tolerated for the environment is essentially an input into the choice of possible algorithms for access checking.

Since many environments have a fairly low level of acceptable staleness (why else have document level security after all), there are hard tradeoffs in the implementation techniques to keep *average security check time* low. Some of these tradeoffs are discussed in more detail in Section 6.

The mechanics of an implementation rely on having an algorithm for producing pages of results, described in the next section.

4. ALGORITHM

We claim no special novelty in the filtering algorithm described below, but present it here for completeness. It is written in a pseudo-code that should be readily rewritten into the syntax of current programming languages. The algorithm returns a set of document ids for the current "page" of results to be displayed by a search engine's Web interface. It assumes the existence of two existing services – a search engine and a security system.

Existing services:

```
searchEngine.query (query: string, number: int, offset: int)
    : documentId list
securitySystem.check (id: documentId, for:
    authenticationTokens) : boolean
```

Method:

```
getResults (
    queryText : string (empty),
    credentials : authenticationTokens (null),
    resultsPerPage : int (NR),
    currentOffset : int (0),
    retrievalSize : int (NR*2/VD) : list
```

Local variables:

```
pageResults : list (empty)
results : list (empty)
result : documentId (null)
maxResultIndex : int (0)
```

Code:

```
repeat
    results = searchEngine.query(queryText, retrievalSize,
        currentOffset);
    if results.size == 0 then return pageResults;
    maxResultIndex = currentOffset + results.size;
    while pageResults.size < resultsPerPage and
        currentOffset < maxResultIndex
    do
        result = results.at(currentOffset);
        if securitySystem.check(result, credentials)
        then pageResults.append(result);
        currentOffset = currentOffset + 1;
until pageResults.size >= resultsPerPage;
return pageResults
```

Notable about the algorithm is that attempts to gain access to secure results by forging URIs cannot work, since the currentOffset parameter (for the starting rank within the raw result set) does not mean security is not checked; it is just a shortcut to avoid first checking currentOffset results before reaching the start of results for a later "page".

5. IMPLEMENTATIONS

Secure search with document level security has been implemented along the lines of the algorithm described above by the first author while at Synop Pty Ltd (using architecture type 1) in its Sytadel content management system, and by the Panoptic team at CSIRO (using architecture type 2). (The Panoptic search engine used for experiments is now available as Funnelback, through Funnelback Pty Ltd.)

5.1 Sytadel implementation

The Sytadel implementation of document level security search uses a search engine adapter framework to plug in any search engine. All content is exposed to the search engine for indexing. When a search is carried out through Sytadel, the query terms are parsed and converted by the adapter to the underlying search engine's query language. Results from the search engine are then filtered according to Sytadel's internal security service and the user's credentials accessing the Sytadel system. This approach means that there is no security staleness – changes in security are reflected immediately in the search results. The search engine must prevent access to the collection from any user or system other than Sytadel.

Sytadel uses the search engine as one of a number of possible retrieval rankings that may be applied over a set of documents, or as a way to obtain a set of documents based on some free text query that can be subsequently filtered by other properties. All content is stored in XML, and the XML content plus content metadata can be used to filter results (in addition to the security restrictions).

5.2 Funnelback implementation

The Funnelback implementation of document level security search has been applied with Windows NTFS fileshares using NTLM authentication. The current implementation crawls a local or remote file collection, augmented with the access control lists. At search time, the user authenticates with the Funnelback search engine via the Windows IIS web server, and the credentials are then used to determine security on a per document basis by checking the access control list for the document (or the chain of directories it resides in) in the local cached copy. This approach means that there is a degree of staleness associated with the security of the search results; based on the crawl frequency.

The current implementation has adopted this approach to avoid the additional complexity of having Funnelback manage security services for the clients. NTLM authentication is based on a client/server model, and to check a third party repository directly requires that the client authenticate with the repository server, not the search engine web server. Such implementation practices are possible, by building in a mechanism whereby Funnelback requires the user to authenticate against each repository server, not the web server, although the web server issues the authentication challenge type 2 message. The Davenport WebDAV-CIFS (SMB) gateway system [http://sourceforge.net/projects/davenport/] is an example application which supports this approach.

5.3 Lessons learned

There have been a number of lessons learned through these implementations.

The visibility density within an organization obviously has a major effect on performance. In our experience, the majority of organizations make the majority of their information available to everyone within the organization, meaning visibility density is typically high. There are usually a small number of exceptions within the collection, such as private HR information (such as medical details, contacts, job applications, etc), which are much more highly restricted. In such environments, document level security is clearly viable and can be made more so if documents are organized into collections such that individual checks are not required for most documents.

To deploy a solution that meets good quality of service (e.g. $AT \leq 1$ second) for all users even in worst case scenarios, using a visibility density (VD) value in the model that is an average for the 10^{th} percentile of users with least access (rather than an average of all users) will help to characterize the maximum average security check time that must be achieved.

In environments where average (not just worst case) visibility density is very low, it may make sense to use a completely different approach. One obvious solution is to deploy personal search engines (such as Google Desktop Enterprise), which search only the information publicly available to an individual. Such per user approaches are viable in small to medium organizations, where the network traffic associated with crawling and indexing information remains acceptably low. In large organizations, having thousands of personal search crawlers attempting to access repositories is likely to use too much of the network bandwidth and impose large loads on the servers.

Contrary to general Web search, where it is understood that content is not immediately indexed, enterprises typically want low levels of delay between new content being created and becoming searchable. In our experience, the same applies to security. When an enterprise changes its security policies, it would like them to apply immediately to search results as well, as people generally have a poor understanding of the costs of having zero security staleness in search results. A discussion is generally required to allow business stakeholders to make an informed decision on the tradeoffs to be made in a search system implementation. Using the concepts expressed in the model, especially relating to staleness and acceptable time may assist.

Security systems tend to be intrinsically complex, and attempt to mirror much of an organization's implicit human-based rules and processes that have been built up over time. Human systems are able to adapt easily to ad hoc changes and exceptions; computer based ones are much more rigid. In our experience, enterprises are using a mixture of repositories, and often each with its own security model. There is considerable overhead matching the security model specifics of a repository into a uniform representation for the search system, even when it is really only read permissions that are required. These complexities contribute to a high degree of difficulty in deploying a search solution which constrains the average security check time to a level that preserves an acceptable time for search responses.

The choice of architecture (type 1 or type 2) also plays an important part in the implementation process. In our experience, the choice comes down to what system is perceived to be the dominant interface to access to information within the enterprise. If it is the search engine, then architecture type 2 should be chosen; if a CMS or other primary repository, then type 1. However, either choice may require making changes to the patterns of user behavior in access to information and/or additional changes to integrate other repositories (and their security systems) with the chosen primary interface.

The exposure of users to Web search interfaces has set a number of expectations about how search "works". For example, users expect to see the number of results available, and they expect to see hyperlinks into successive (and prior) pages of results. Calculating the complete number of results visible to a user is possible, but comes with additional costs as is seen in the next section. Generating next page/previous page hyperlinks on result pages is straightforward and efficient. Generating pages 1 2 3 … hyperlinks is hard (and the implementation will be slow). For example, to have a page 4 link requires that the system searches linearly through the result set checking (and discarding) results 1-39 to find results 40-49 that match the security of the user, and it must also calculate the total number of results visible to the user. Of course, both of these features can be dropped for performance reasons. Even on Web search engines, the numbers are nearly always just estimates, and there is a sharp dropoff in people clicking past the first page of results.

Lastly, the search engine must have "global" security rights – the ability to view all content anywhere in the organization. Administrative access to the search engine's "global" collection (when not mediated through the security system) must be highly protected to the same or higher degree that administrators with access to other secure repositories are restricted.

6. EXPERIMENTAL TIMES

While the implementation experiences are illuminating, hard experimental analysis of our model provides even more insight.

6.1 Setup

Basic setup of the experiment involved indexing and searching a 10 000 document collection from the Wikipedia's simple dump (http://download.wikipedia.org/wikipedia/simple/) using the CSIRO's Panoptic search engine. The test server was a Dell Optiplex 3GHz Pentium 4, with 512MB of RAM and 150GB of disk storage, running Windows 2003 SP1. Various patterns of enterprise security were used for splitting this collection and investigated. Times were reported to a log file, with 5 runs per query per user, which were then averaged. All times reported are in milliseconds. In practice, splitting the collection according to different security patterns reveals relatively little of interest. Instead, using different queries and different users for a single security pattern demonstrates everything that is salient.

The times reported use the following security pattern split, which is approximately representative of one particular way that access control might be applied within an enterprise. Note we make no claim that this is a real enterprise security scenario, which we expect would be considerably more complex. An organization size of about 50-100 might be representative.

Security is applied primarily to groups of documents:

- 1% of docs visible to Private group only – i.e. 100 documents, representing one user
- 6% of docs visible to HR group only
- 24% of docs visible to anyone (Public group)
- 69% of docs not visible (Other Private group docs) representing other individuals

Visibility by users when conducting searches:

- Public person can view Public group (24% of docs)
- Private person can view Private and Public groups (25% of docs)
- HR person can view Public and HR groups (30% of docs)
- Contractor person can view only 1 document per query.

The baseline in each test query is Panoptic with no document level security applied. This is approximately equivalent to a case where collection level security is applied, since there is a single check made to decide if the user is permitted to view the collection.

Table 1 – Query terms and number of documents which match within the collection, irrespective of security

Term	Number of docs
aloof	1
verse	10
triangle	20
sugar	50
available	100
china	249
had	499
may	743
like	1008
are	2511
of	5216
wikipedia	10000

A set of one word queries was determined from the collection's lexicon, to approximate progressively larger numbers of potentially matching documents from the collection. The queries and number of potentially matching documents are reported in Table 1. Again, these are not representative of a real enterprise's

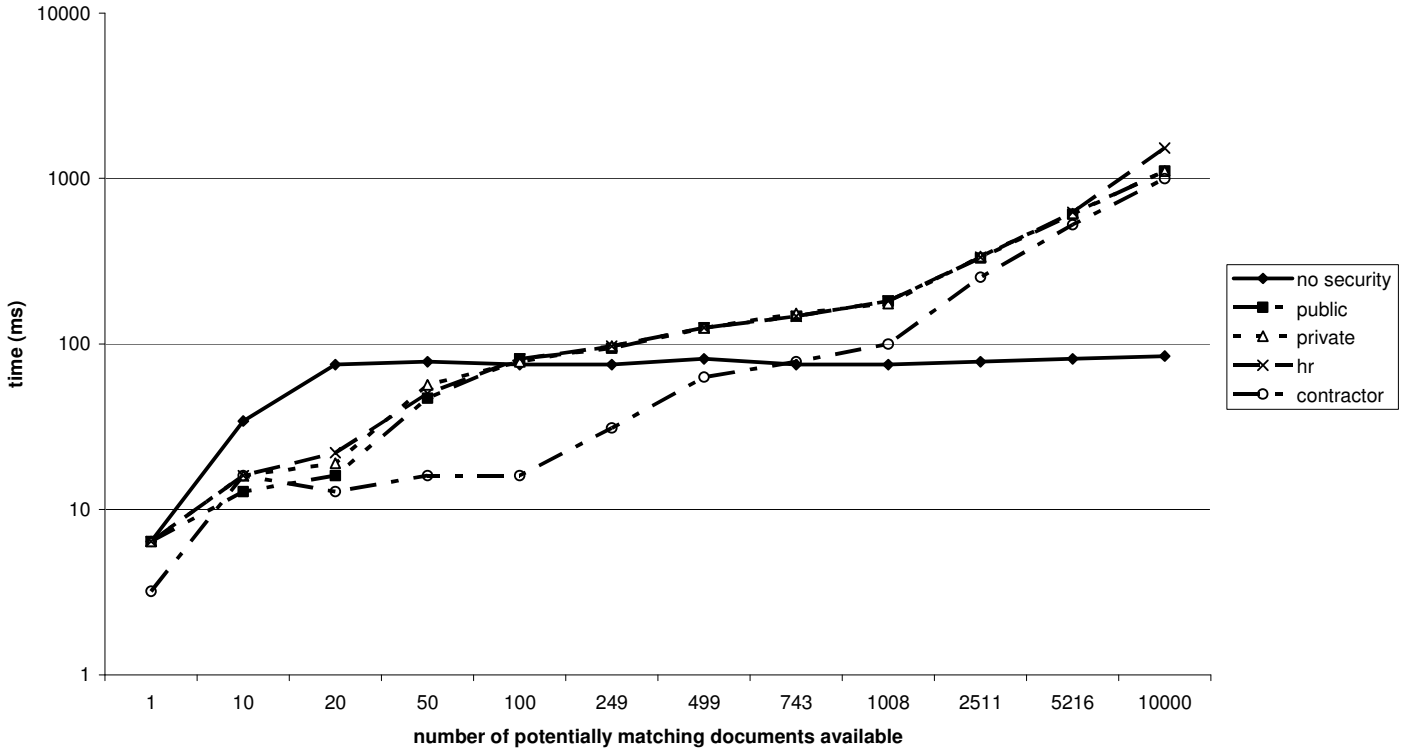

Figure 3 - Times for queries with different users and on-server access checks

search queries; they are used to examine the behavior of the system with respect to properties of the collection.

6.2 Reporting the exact counts of matching documents

Panoptic is operated first in a mode whereby all possible matching documents are found before reporting the first page of results (*NR* = number of docs). This enables an exact count of the total number of visible matches to be reported. It means that all potentially matching documents must be checked against a user's security credentials before deciding whether to include the result or not.

Results in Figure 3 demonstrate that as the number of documents that are potentially visible to the user increases, the overall time to process the request converges regardless of the visibility density, and becomes proportional to the number of potential matches. A log scale in the y-axis is used to more effectively show the correlation between time and the number of potentially matching documents (in the x-axis). In comparison, the processing time with no security checking remains roughly constant at about 100ms, from about 20 documents, with a minimum time of around 6ms for just 1 document. These results are in line with our model (1) of the parameters, remembering that the dominant factor is the *number of results* times the *average security check time*. The *overheads* appear to increase as the number of results increases, but stabilize at 100ms for no security check times at 20 documents. Note that timing inaccuracy renders these values only approximate, not precise.

A peculiarity of interest is that the no security times are greater than secure times for small numbers of potentially matching documents. This apparent anomaly is explained by the additional overhead in producing additional result information such as query biased summaries. The no security time for 20 potentially matching documents must show 20 actually matching results, but the various secure results are showing from 1 to 4 results (according to visibility density), there is considerably less per result presentation overhead required. We verified this by re-running the "triangle" query (with 20 potentially matching documents) for all users with a number of different result presentation modes. When only a single result had to be shown the query processing time is identical. By 100 potentially matching documents, the Public, Private and HR people's searches must also show 20 results, and the timing reflects this. The Contractor takes substantially less time than the other secure people, since they only ever produce 1 result in the results page.

From an implementation perspective, with up to 1000 potentially matching documents, the query time to calculate security correctly over all matching documents remains close to the no security check time. Beyond that level, and certainly by 10 000 potential matches, the query time grows by up to an order of magnitude, and appears to be growing linearly with the number of potentially matching documents. Note that large numbers of potential matches are a distinct possibility, depending on the query and the size of the organization. For example, in [4] the authors report they found 4.6 million non-duplicate pages within IBM's intranet.

6.3 Security checking times

In Table 2, we calculate the incremental cost per document of checking security on all potentially matching documents versus the no security check computation time. At low values of potentially matching documents, since the no security check time is greater due to the overhead in producing more actual results (up to 20 for a complete page) than for the secure results which may have less than 20, the times are negative. However, from the query *may* (with 499 matching documents), the times start to stabilize for the Public, Private and HR users. The

exception is for the Contractor. This exception demonstrates that the code path is distinctly different when a match is found to be visible compared to when a match is not visible. An average of the times for the last 5 queries is shown in the final row. The average security check time for the Public, Private and HR users is approximately 0.11 ms per document. These costs are being hidden relative to the overall request processing overheads at small numbers of potentially matching documents.

Table 2 – Average security check time (in ms) per document calculated against the no security check time baseline

Number of docs	Public	Private	Hr	Contractor
1	0.000	0.000	0.000	-3.200
10	-2.140	-1.820	-1.820	-1.820
20	-2.950	-2.800	-2.650	-3.110
50	-0.620	-0.428	-0.556	-1.240
100	0.062	0.030	0.062	-0.590
249	0.076	0.088	0.088	-0.177
499	0.110	0.110	0.110	-0.046
743	0.097	0.105	0.097	0.004
1008	0.106	0.099	0.106	0.025
2511	0.211	0.103	0.102	0.070
5216	0.048	0.102	0.104	0.085
10000	0.102	0.102	0.144	0.091
average of last 5	0.113	0.111	0.111	0.055

Considering equation (3) from the model in section 3.3, assuming the same example values of 1 second *AT*, 0.25 second *O*, and 80% *VD*, we see that the maximum number of results which can be checked for security at this performance level of *ASCT*, while maintaining the quality of service (of a 1 second acceptable time), is 5454.

6.4 Caching and batching

In the test scenario, all documents were located on the same server as the search server. The implication of this is that checking security is an efficient activity, since no inter-server requests need to be performed. Such a scenario is also representative of one where some security staleness is acceptable, since the access control information for all documents may be cached on the search server at collection indexing time for efficiency.

Some simple inter-server LDAP search tests were performed to determine what the cost of calculating security in a distributed repository environment where the acceptable security staleness is zero. These involved querying an LDAP (ActiveDirectory) server running on similar hardware to the search server from another server in the same local network. The average system time of an LDAP query in these circumstances was 6.4 ms.

Reconsidering equation (3), again assuming the same example values, we see that the maximum number of results which can be checked for security at this performance level of *ASCT*, while maintaining the quality of service is 93.

A straightforward implementation to check access means that at best 157 potentially matching documents can be checked in a second. With 10 000 potentially matching documents, it would take over a minute to check security for all documents. Obviously inter-server access control requests require considerably more time than inter-process requests on the same server, but this is a necessary tradeoff if security staleness must be zero and repositories are distributed across multiple servers.

One way to dramatically improve performance in this situation would be to batch requests for access control checks. Thus instead of checking each result for a security match, a single request could list 100 to 1000 document URIs to check for a user, and the response would list the ones which are accessible. There would be some increased overhead in building the larger request and reply packets and transmitting them, but the bulk of the time is due to network latency, not network bandwidth. At present, the standard Microsoft file protocol (SMB/CIFS) does not support such batching of requests, but according to Andrew Tridgell (creator of Samba) this could be added through an extension to the *trans2 findfirst/findnext* calls [Tridgell, personal communication].

The conclusion to be drawn from this set of results is that keeping the acceptable time at or below 1 second is going to be increasingly difficult if a complete count of matching results must be reported with any of the following: zero security staleness and repositories distributed across servers; larger collections; or more users conducting searches. In the hosted search services operated by CSIRO for various organizations, query request frequency often exceeds 1 query per second.

6.5 Prioritizing acceptable time

In the second set of tests, Panoptic is operated in a mode whereby only sufficient security matched documents (numbering 20) to show a page of results are found. This corresponds to setting *NR* in our model to 20. In this mode, if there are more than 20 matching results, it is not possible for Panoptic to report the total number of matches. The same query terms as before are used, and times are shown in Figure 4.

The Contractor person is not shown in this graph, since their results are the same as in the earlier graph (as they have only one matching document). Note that performance in worst case scenarios for either unusual queries (i.e. large numbers of potential matches, but few visible ones to an average user) or people with very low visibility density, will become progressively poorer as the collection size increases.

For our sample people (Public, Private, and HR), as they have plenty of available matches for each query, performance is similar to the collection level/no security scenario by the time approximately 100 potential matches are available. The same behavior regarding number of results actually shown is observed, with the no security time more expensive until this level. Query processing time slowly increases past this level, which is most likely due to slow increases in the memory overheads for handling larger potential sets of matches. The query processing time stabilizes at around 75-90ms, which allows acceptable time to be kept well below 1 second even with larger collections, and increases the scalability of the system with respect to increasing query load. There are not significant time differences between the different people, which is to be expected as they have roughly similar visibility densities.

Additional performance wins can be obtained by examining security as applied to hierarchical directory structures. Assume that all human resources documents for the organization are located in the directory /HR. When calculating the result set for

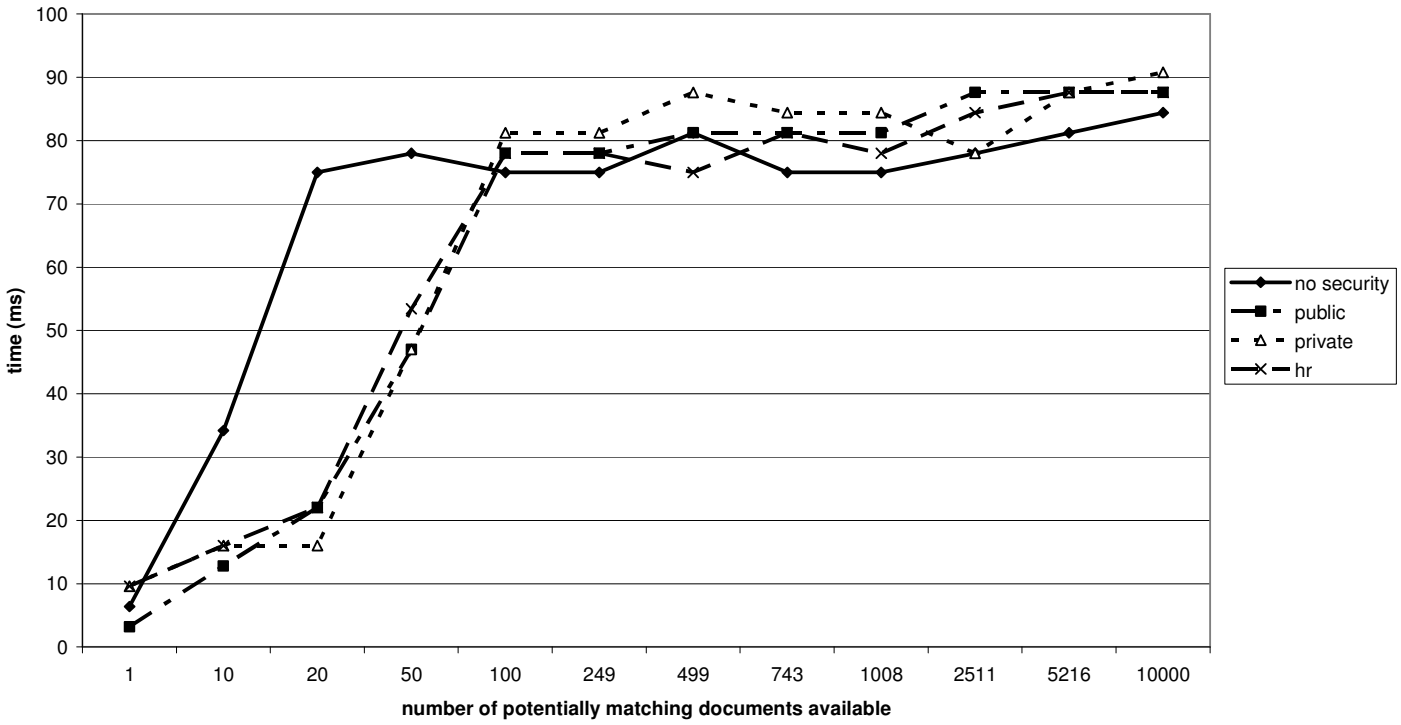

Figure 4 - Times for queries with different users to produce at most 20 results

Fred, who is not a member of the HR group, the directory can be checked for directory traversal rights for Fred and found to be blocked for access. Subsequently, any additional results located in /HR must also be inaccessible, and the more costly access control check can be replaced by simple string comparisons on the URI path. This principle applies under both Windows SMB/CIFS protocols and UNIX filesystems. (The reverse, finding a document to be accessible and assuming all documents in the directory are accessible, is obviously not safe. Note also that these techniques only apply to collections consisting of files in directories, and not general URI accessed documents, since web servers may rewrite URI paths on an arbitrary basis).

7. CONCLUSIONS

By far the most efficient models for implementing security of search results are: (a) an integrated portal and (b) collection-level security. In the case of the former, currency of the security model applied during search is guaranteed because the portal controls both the search and the access control of documents. Security checks can be quick because they are made internally, rather than to a third-party system (necessitating inter-process and sometimes inter-server communication). In the case of collection-level security, only one access check need be made per search rather than one for each potentially matching document. Collection level security implementations can respond quickly to changes in the permission rights of individuals to access a collection, although not to changes in which documents are part of the collection.

When document level security rights are enforced by a system external to the search engine, or worse, on a separate server, the costs of conducting access checks increase dramatically. In the case where the search engine must report an accurate count of matching documents, we found that query processing time was increased by an order of magnitude for a query where there were 10 000 potentially matching documents. In a simple-minded implementation the cost of security checks is linear with the number of results to check, leading to query times of 100 seconds when the number of candidate results reaches 1 million documents (possible in the largest enterprises), even using caching of access control lists on the search server and thereby sacrificing zero staleness.

When an accurate match count is not required, average costs reduce substantially, and become close to those with collection level security or without security at all. The Panoptic security result mode of finding just 20 matching documents works effectively in this way. A pragmatic compromise (used in the Sytadel implementation) is to report match counts accurately when there are fewer than some limited number of potentially matching documents – typically between 100 and 1000. However in worst case scenarios, e.g. searchers who have access to only a tiny proportion of the available documents (low visibility density), the query time may still end up being just as high as with full security checking.

Costs of external security checks may be reduced by three techniques: batching, caching, and exploitation of hierarchically structured security permissions. Caching has tradeoffs in terms of increasing staleness of security, and thus may not be applicable in all enterprise environments. Batching requires the development of extensions to existing file system protocols to allow efficient access control lookups. Entities responsible for the improvement of existing file system protocols should look to provide such extensions in future. Hierarchically structured security permissions may only apply effectively in file collections.

Where possible, organisations should structure their document security model to allow for collection level security to apply. Additional collections for which document level security must apply should be minimised. Search engines can then provide

search over multiple collections, and perform result list combining to merge results, while still respecting per user access control over all documents.

When implementing a search solution within an enterprise, our model which relates *acceptable time* and other search parameters to *average security check time* or *number of results* should be used as a guide to ensure the search experience remains effective and sufficiently fast to users. In our experience, an acceptable time of 1 second should be seen as an upper bound within which results should be returned for most search tasks.

8. ACKNOWLEDGEMENTS

The authors thank Andrew Tridgell for sharing his specialist expertise about the SMB/CIFS protocol. We also thank the anonymous reviewers for helpful suggestions for improvement.

9. REFERENCES

[1] Abrol, M., Latarche, N., Mahadevan, U., Mao, J., Mukherjee, R., Raghavan, P., Tourn, M., Wang, J., and Zhang, G. Navigating large-scale semi-structured data in business portals. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases* (San Francisco, CA, USA, 2001), pp. 663-666. Morgan Kaufmann Publishers Inc.

[2] Broder, A.Z. and Ciccolo, A. Towards the next generation of enterprise search technology. *IBM Systems Journal 43,* 3, 451-454.

[3] Coveo, Inc. CES040406-1: Understanding document level security. http://www.coveo.com/support/articles/Information%20-%20CES350-040406-1%20-%20Understanding%20Document%20Level%20Security.htm. 2004

[4] Fagin, R., Kumar, R., McCurley, K.S., Novak, J., Sivakumar, D., Tomlin, J.A., and Williamson, D.P. Searching the workplace web. In *WWW '03: Proceedings of the 12th international conference on World Wide Web* (New York, NY, USA, 2003), pp. 366-375. ACM Press.

[5] Google, Inc. Google Authentication/Authorization for Enterprise SPI Guide http://code.google.com/enterprise/documentation/authn_authz_spi.html

[6] Hawking, D. Challenges in enterprise search. In *CRPIT '04: Proceedings of the fifteenth conference on Australasian databases* (Darlinghurst, Sydney, Australia, 2004), pp. 15-24. Australian Computer Society, Inc.

[7] IBM, Inc. DB2 Information Management Software – Information center. Enterprise search security. http://publib.boulder.ibm.com/infocenter/db2help/index.jsp?topic=/com.ibm.db2.ii.of.doc/admin/iiysasecure.htm. Version 8.2.2. 2005.

[8] Lesk, M., Cutting, D., Pedersen, J., Noreault, T., and Koll, M. Real life information retrieval (panel): commercial search engines. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on research and development in information retrieval* (New York, NY, USA, 1997), pp. 333. ACM Press.

[9] Mukherjee, R. and Mao, J. Enterprise search: Tough stuff. *ACM Queue 2*, 2, pp. 36-46.

[10] Verity® K2 Architecture White Paper. 2002. Verity, Inc. http://www.verity.com/pdf/products/ics/k2_enterprise/whitepapers/MK0366_K2Arch_WP.pdf