An architecture to search private data using arbitrary assessment algorithms without disclosing their content

A. Yannopoulos, Y. Stavroulas, N. Papadakis, D. Halkos, T. Varvarigou
National Technical University of Athens,
Department of Electrical & Computer Engineering,
Division of Computer Science, Telecommunications Laboratory
Zografou 157 73, Greece

Abstract: - This paper describes an architecture which allows an un-trusted search algorithm supplied by a User to be granted access to private, non-disclosable data of a Data Provider. The result is that the User can receive the output of the algorithm, a relevance rating or other useful statistic, but cannot ever access the data itself. Some support is required by the Data Provider, who actively sets up the system. This means the Data Provider knows exactly how security is achieved and can therefore trust the system fully. Java security management is utilised and the system is made efficient through the hosting of a Mobile Agent platform. Each component of the system is given an appropriate security profile so that no unauthorised access to information is ever possible. Two example applications are described, which allow a user to assess (but not access) the full-text of published books, in the first case, and the content of private images, in the other.

Key-words: Full-Text Search, Security, Data Access.

1. Introduction

A platform is proposed in this paper to allow the secure cooperation of two parties: 1) A Data Provider, which possesses a number of “documents” that must not become publicly available at any time, but about which certain queries should be answered as informatively as possible. The documents can be any sort of data, e.g. text, images, audio files, etc. 2) A User, who wants to assess these “documents” using a custom-built algorithm. This algorithm is not in the possession of the Data Provider and may only be given away in binary form.

The Data Provider would be happy if the User could have the output of the algorithm run on the documents, but cannot allow free access to the User because then unauthorised copies of private data could be made. Is there a way to run an un-trusted algorithm provided by the User on private documents of the Data Provider and offer the User useful feedback but no opportunity to keep records of sensitive data? Furthermore, can this task be carried out efficiently, especially avoiding large User waiting times, CPU overload and large bandwidth consumption between the two parties, since a large database of large documents might be regularly searched by many Users?

The concept of remote execution of code, as generally supported by Java and especially Java’s security framework [13], [11], and as furthered by the more specialised cases of Mobile Agent platforms [2]-[7], provides a basis for a single approach which naturally satisfies both of the above requirements. The User’s code is loaded onto the Data Provider’s site, supplied with the necessary “documents” and executed in a completely restricted environment. No communication back to the User is allowed; the code’s output is captured by the Data Provider, thus assuring that it only contains acceptable information, and only then returned to the User. In counterpoint to this heavy restriction, we assume that each document is characterised by some publicly available information which could include a title, a summary (for text data) or a description (for images, audio etc.), a keyword list, statistics such as date and size, etc.: this non-sensitive data is offered for processing by code again written by the User, running once more at the Data Provider’s site, but restricted this time only to avoid damage to the hosting site and without the need for data protection. The extra information allows the user to coordinate the search and considerably improve overall efficiency, making the system viable for practical implementation in many environments.

An example of a generally useful application is that of a publisher who could easily make the full text of books available electronically. It would be entirely unacceptable for unauthorised electronic copies to be made. On the other hand, it would be very useful if people contemplating the purchase of a book could determine how relevant the content actually is to their needs. In a bookstore people browse through books, but electronically this is not done since anything downloaded for evaluation by a user can easily be copied and retained. The architecture we propose allows arbitrary search methods to be
employed on such full-text documents in order to assess their relevance to any criteria the user specifies.

A more specialised example involves assessing the content of classified image databases. This could become valuable in a case such as that of international police cooperation. Police departments of different countries would like to make their classified data searchable for other departments. The full data collections could not be made openly available to scrutiny by human experts, but if a foreign department could mechanically determine a small number of specific items which would be valuable to it, their disclosure could then be considered, or, at least, useful feedback could be provided. Such databases would be made openly available to searches in the sense that they would be online and queries would be answered automatically and without the need for supervision. However, operation would be under strict quotas, as described below, so the service would have to be regarded as a valuable resource and not to be used without good need.

Results from test implementations of both these examples are presented after the theoretical description of the architecture. Keyword searches are used for the text documents, while face recognition/detection is performed on the image documents.

2. System Architecture

The overall system architecture is shown in Figure 1. In brief, the proposed system functions as follows: The User can run arbitrary code within the Mobile Agent platform, but only public information can be accessed there (with CPU and bandwidth quotas imposed, if desired). The Search Filter Object runs within a Mobile Agent and chooses which documents are worth full evaluation, based on the related public information. Its results are passed into the Sandbox, but not directly. Rather, a method of the Connecting API is called, which simply passes its parameter into the Sandbox. From the set of private documents available, the requested subset is selected. The Search Algorithm Object is capable of assessing a private document for the user. It is loaded onto the Mobile Agent platform but has no access to documents there. It is passed as a parameter to the Connecting API, which runs it within the Sandbox. The algorithm can only respond back to the Connecting API, since it has access to absolutely no resources save for the single document which it is given and the sole communication option of returning a value upon termination. Finally, the Connecting API passes the return value into the Mobile Agent platform for the User’s code to exploit. Since the Connecting API only offers a method to feed back this single value, communication out of the Sandbox is strictly controlled. Note that the Search Filter Object may produce a Search Result Object by itself if the summary information which it accesses is sufficient on some occasion. Note also that any other code may be run within the Mobile Agent platform to coordinate the search(es) and to handle the result(s).

The Data Provider has a number of tasks to fulfil. All of these, however, are “standard procedure” and hence easy to set up. It is the combination of processes which lends the system its special functionality. Specifically, the Data Provider must do the following (at its own site):

1. Set up a “Sandbox” environment. This is as easy and safe as running a Java applet with maximum security, of which the equivalent in the current situation is an application run-time environment based on the Java SecurityManager set up in the most restrictive manner [11], [10]. The Sandbox is given Java objects by the “Connecting API” which is described below; it provides these objects with private data as input and passes their output back to the Connecting API. The objects are never allowed to communicate with anything but the Connecting API but, internally, they may implement arbitrary functions to assess their input. Note, finally, that in setting up security restrictions, denial-of-service attacks are easily avoided by specifying CPU and bandwidth quotas.

2. Host a Mobile Agent platform [2]-[7]. This is in practice a less-trusted component to deal with. However, although it does not contribute to the black-box-view functionality of the system, it is important for its efficiency. Because it may not be fully trusted, it is never given access to any private data. It is set up to communicate freely with the User remotely and the Connecting API. It is also provided with public Document information whenever it requires it. Otherwise, it should be entirely cut off from the Data Provider’s system (i.e. it may certainly not access the file system of the Data Provider, or any other sensitive resources).

3. Implement the Connecting API. This component is at the heart of the system. Correct (secure) functioning hinges on its unflawed and fully trusted operation. The owners of a data source are likely to be very cautious when it comes to granting access to their resources. Therefore, by design: 1) except for the (routine) setting up of access privileges described above, the Connecting API carries the full responsibility for the security of the system and 2) the Connecting API is very simple to implement. Thus, in order
to install the proposed system, the Data Provider can choose to use a ready version of this API, examining it at will to verify its correctness, or alternatively the API can be re-implemented with a small amount of effort by systems programmers of the Data Provider itself.

4. Implement a document format converter. This converter is required to transform documents from their native format, in which they are stored, to the standard XML format acceptable by the search algorithm interface. Essentially, it converts documents from their provider-specific data format to the format readable by the search algorithm. In practice the need to implement a converter will rarely occur, as converters for popular text formats (plain text, HTML, Microsoft Word, RTF, PDF) are already provided, while other types of data will generally be encoded in standard formats.

![Figure 1 - Overall System Architecture](image)

To examine the details of the proposed system’s operation, a good starting point is to focus on the possible attacks which it must withstand. We will concentrate our description on text data initially, and compare to other data (basically images and audio) in conclusion.

Firstly, a malicious User could attempt to bypass the specified access restrictions. Within the Mobile Agent platform, communication back to the User’s site is free, so it would be enough for code executing therein to manage to access a private document. From within the Sandbox, access to private documents is granted, therefore it would suffice to find a hole allowing communication back to the user’s site. Whether the proposed system can withstand such attacks is simply a question of whether Java security management can be compromised or not [9], [10], [11], [13]. This paper’s assumption is simply that Java security works. If a significant risk is seen in the small probability that a Java implementation bug could disrupt the system, then the proposed system, and just about any other Java-based application implemented by a third party, should not be used on-line by the Data Provider.

Secondly, a malicious User could attempt to leech private data through the channel which is intentionally available. Any feedback from within the Sandbox back to the User could be taken advantage of. The intention is that such feedback should indicate whether the documents viewed are interesting to the User, but the User could instead encode small sections of the document itself into the feedback data, and attempt to slowly steal the entire document. The only solution possible is to massively restrict the amount of data allowed for feedback from the Sandbox. Each time the User’s code assesses a private document, it is allowed N bytes of feedback, where suggested values for N are as low as 1 or 2. Furthermore, the User is simply not allowed to perform more than M searches per day which access the full-text of a given private document, where M is once more suggested to take values as low as 1 or 2.

Consider the more lenient case where N = 3 and M = 7. The User is allowed to write arbitrary code to access a private document, so choosing which bytes to steal can be organised well. N*M = 21 bytes can be transmitted each day, which amounts to 7665 bytes per year. Even with 3:1 data compression, 22195 bytes stolen in a year from a full-text document is probably not a significant threat. With N = 1 and M = 2, the number of bytes per year reduces to 730.

Since the number User queries which can access private documents is so restricted, it is important for the Search Filter Object to function as well as possible in order to avoid wasting this allowance. It is often hard to accurately assess a potentially interesting data item, but it is often much easier to detect others which are definitely not interesting. Significant savings can be made by performing such assessments outside the Sandbox. It should also be stressed that the considerations described previously make it necessary for each private document to be viewed in isolation within the Sandbox, with a new Search Algorithm instance created for each one (thus avoiding data from one document being included in a false result intended for another). This also makes it important for the User’s code within the Mobile Agent platform to efficiently coordinate the requests made through the Connecting API.
In order to control the number of queries performed by each User, the system must include an authentication mechanism which will not be covered in detail here. Possible solutions would include cryptographic authentication of any User desiring to utilise the system, or an account-based front end which Users must log on to using a safe protocol before being allowed to submit a search query. Further attention will be given to the issue of how non-technical people will search for textual information in the manner presented in section 4.

Note that the carefully controlled information which is allowed to flow from the sandbox into the Mobile Agent platform can be post-processed at will before being returned to the user. The relevance statistic derived from the full text can be connected to any public information related to the private document, such as title, summary, etc. The whole bundle can then, of course, be converted to a reader-friendly format, and sent back.

Finally, in the case of non-textual data, there are significant advantages to be gained. Document sizes will generally be large, which means either that quotas can be increased to maintain the same level of security, or that higher security is achieved with the same quotas. Also, in many cases data sizes can be inflated arbitrarily, resulting in larger computational costs but also arbitrarily greater security: an image can be converted to a higher resolution or merged with other, randomly selected non-security-sensitive ones. Audio can be up-sampled and could also be placed within irrelevant random data. The use of random data may well require the search algorithm to be adapted, but in many cases this will in fact turn out not to be a problem: in our example, which aims to detect whether a certain person is shown anywhere within a complicated image, the same algorithm can work on the inflated images with negligible performance loss.

3. Implementation Outline

The arbitrary functionality which the User can implement in order to locate documents of interest must be wrapped in a predefined format in order for the Data Provider to be able to run any code which the User produces. The following Java classes and interfaces show how this can be achieved.

This code deviates in one detail from Figure 1. The figure shows the Data Provider offering documents, public and private, to the user through some means other than the Connecting API, which has not been specified in the discussion above. In practice, there is no reason to implement two APIs, and all documents can pass through the Connecting API. This has not been depicted in the figure for clarity, since it is purely an implementation issue, and the code below is given merely as an indication of how the system described might be implemented.

The code which is presented is a slightly simplified version taken from tests which have been performed on the system, using text documents as data. It is intended mainly to show how simple the wrapping and interfacing necessary to implement the architecture is. By using such code, User and Data Provider can coordinate and ensure that compatibility of incoming code with the available platform is achieved.

```java
import java.util.List;
import java.util.Date;

public class Document {
    public long id;
    public List fulltext; // list of Strings, so it fits in memory.
    public List keywords; // list of Strings, one per keyword.
    public String title;
    public String main_topic;
    public String summary;
    public String author;
    public String publisher;
    public long size;
    public Date date;
} // note that fields may be null!

public class SearchResults {
    static final byte N = 1;
    public Document document; // for reference, check that it is valid
    public byte[] results = new byte[N];
} // yes, it is so simple, and should be!
// Let the user interpret meaning of the bytes.
```
abstract class SearchFilter {
  private ConnectingAPI theConnectingAPI;
} // this class is run by the MA. All it needs is a reference to the ConnectingAPI and custom running method.

abstract class SearchAlgorithm {
  abstract public SearchResults assessFullDoc(Document fullDoc);
} // this class is given to the ConnectingAPI, which must know how to get it running.

public interface ConnectingAPI {
  public SearchResults[] performFullDocSearch(Document[] docs, SearchAlgorithm algorithm);
  public SearchResults[] performFullDocSearch(Document[] docs, SearchAlgorithm[] algorithms);
  public void newSession();
  public Document getNextPubDoc();
} // simplest implementation is to call these methods from the SearchFilter

4. The full-text document search service

The system presented in this paper offers two opportunities to developers. Firstly, it should be noted that, until now, it has been shown how a custom search algorithm can be run on private data even if un-trusted by the owner of this data. Clearly there will be no large number of users actually writing Java code to be executed.

The first possibility for successful use of the system would involve the penetration of a generic and probably relatively simple search method into data sources which were previously entirely withheld from public scrutiny. For example, a publisher could allow clients to run keyword and phrase searches on the full-text of books before they decide which to purchase. Clients could then determine the detail in which a main topic of interest is covered within a book which actually concerns a number of topics, test reference to specialised sub-topics not mentioned in published summaries, etc.

The system's two key advantages, that is the non-disclosure of information and the acceptance of any search algorithm, can thus power a new component to Internet search engines. Today, most search engines crawl the Internet and index locally the public information they find (e.g., amongst many others, [8]). Thus searches are performed locally and information which is not publicly available is a priori excluded. The extra component can work in parallel to this classical structure to overcome the availability constraint. It would submit its search engine to each co-operating site in advance and, whenever a search was asked for by a user, it would distribute the parameters to the various sites. The searches would then be executed in parallel and the results sent back to the search engine. Note that the Data Provider would require proof that each search request submitted has originated from a specific, registered and authenticated end user. This could be easily achieved by integrating the search engine with the user’s web browser or some other programme, and letting the software submit the search directly to the provider. This could still be achieved while presenting the user with an environment identical to what would be experienced if submitting queries to a site hosting a search engine. An important advantage of such a situation is that search algorithms and interfaces to end users could be developed by third parties and would be immediately runnable on all cooperating Data Providers, which offers a huge advantage over today’s situation, where a Data Provider wishing to offer special search capabilities to users must implement and maintain them itself.

The second possibility is that this architecture could offer a new opportunity for the application of much more intelligent search algorithms than the evaluation of Boolean expressions as implemented by the standard search engines which are popular today. Applying artificial intelligence methods is especially difficult when the available data is unstructured, but developing and testing such methods is also much harder when proprietary data is used under strict licensing policies which confine research work to be performed only in a confined environment. Although development work could not take advantage of the proposed system (since development generally requires full access to data), it would be very useful if existing methods, which currently produce inadequately useful results when running on public data, could be tested as a search method on highly structured proprietary data, upon which success rates would likely be noticeably

1 This search, although still simple algorithmically, would need to be modified from its most common form, as will be discussed two paragraphs below.
higher. In this case, the developer of the search algorithm would also want to be protected from the Data Provider: uploading a piece of code which has been specialised to assess a full document based on predefined and hence hard-coded criteria would be easy, whereas having made a general-purpose algorithm available to the Data Provider to run with search criteria defined by its own customers would have amounted to giving the algorithm away.

What is achieved by the system presented in this paper is the separation of the ownership of interesting data and the implementor of the search algorithm. Let us revisit the example of users running keyword searches before buying a book. It would be easy for a publisher to design their own standard keyword search, accessing books in full-text, and make it available through their web page. However, an entirely conventional keyword search would be inadequate for documents of the size considered. Once the search could deal with the proximity between terms sought, count term occurrences and measure the dispersion of these occurrences, etc., it would become very useful. No single publisher is adequately motivated to develop such a method, but with the suggested architecture a third party possessing the necessary expertise could do so instead and publishers could safely allow registered clients to use it.

4.1 Relation to the MEMPHIS project – Application example

The idea presented here has been developed tangentially to the IST project “MEMPHIS” which all authors are involved in [1]. This project is a very ambitious one and work is organised by developing a demo service, which, once functional in a basic way, will be expanded and improved as much as possible. This paper provides one path along which such improvement can be sought.

MEMPHIS goes far beyond the acquisition of interesting content. In full, the project involves the definition, development and validation of a new system which will enable the operation of commercially significant multilingual premium services, focused on wearable thin clients such as mobile phones, PDAs, etc. MEMPHIS will extract multilingual information from various content sources in different formats, e.g. from the Internet. Content acquisition will be based on dynamic user profiles and tailored to users’ specific interests and requirements. Acquired content will be transformed into a meta-representation which will enable content storage independent both of source and destination format. Meta-processing steps will include recognition of the source language and automatic translation of the source document into a meta-language. The utilisation of information extraction will provide compressed documents, e.g. summaries. On user request, a translation choice from meta-language to destination language will be provided. Destination format conversion will be related to the thin client output format requirements. This will include conversion into documents of different length but extracted from the same source. [1]

Very powerful legacy software is being used in the project for Natural Language Processing tasks including, most relevant to the current discussion, powerful Topic Identification algorithms. However, the natural limitation is that this intelligence can only be applied to publicly available information. The main demo service being implemented at the time of this writing is called “media alert” and informs users about new books being published. The only way to decide whether a new book is interesting for a user – given this user’s detailed preferences – is to visit the publisher’s web site and apply the considerable machine intelligence possessed by the system to publisher-provided summaries. Since the output required is merely a relevance rating, so that users can be informed about new publications most relevant to their predefined interests, the algorithms applied to the summaries could just as well be applied to the documents’ full text. More computational power would be needed, but the result would be a much more discerning system.

5. Face recognition/detection in private image databases

The other application which we have considered deals with the determination of whether a specific person is shown in some given image. Face recognition technology currently under development at NTUA (the National Technical University of Athens) was wrapped in interfaces such as those specified in section 3. The documents were simple bitmapped images, each one complemented with public information giving a purposefully vague description, the date of its creation and its size, plus an arbitrary integer code to identify it. We chose to allow 8 bytes of feedback, for a double-precision floating point probability assessment to be returned.
The recognition task is carried out with neural-networks techniques. Specifically, the Probabilistic Neural Network is used. Background information on this network will be provided upon request by email to the primary author of this paper. For each person to be recognised, the neural network is trained with a number of “views” showing him or her under different conditions. Thus, the identity of the subjects to be detected is built into the search algorithm used. The feature extraction performed on the input images is an active research topic at NTUA and will not be presented here. This, perhaps, only better illustrates the generality of our approach. The result of the algorithm is a probability for each subject sought. How exactly this is produced is immaterial with respect to our desire for mobility and security.

The Search Filter chooses images created within a specified range of dates. The Connecting API takes the Search Algorithm Objects and provides all images in the database within this range to an equal number of object instances. The match probabilities are then returned to the Mobile Agent platform.

### 6. Content of Search Filter and Algorithm – Current testing

The code which can be executed within the framework described can be of arbitrary design. It is strictly limited in terms of communication with and access to the outside world, but algorithmically there is no restriction beyond that of computational power available. The natural language processing algorithms used in MEMPHIS\(^1\), for example, could be loaded onto the platform without any trouble, given, of course, that they have been implemented in Java.

Testing the system in the case of text data has been based on a mere keyword and phrase search on large ASCII documents, as the most important issue was to see whether data could be stolen. The architecture’s safety has such concision, however, that all attempts to compromise it are pitched directly against the Java framework, the security of which is not in dispute. Leeching 1 to 9 bytes per day does not seem a realistic threat, while the security method implemented is too strict to allow this rate to be increased. Identical results hold in the case of images.

Test output is different, at two levels. Internally, on the Mobile Agent platform, a result object is received for each document (text or image). We could tabulate the contents of these objects for display:

<table>
<thead>
<tr>
<th>id</th>
<th>text title</th>
<th>short description</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MEMPHIS technical annex</td>
<td>Contains the initial specification of the system to be built...</td>
<td>0.22</td>
</tr>
<tr>
<td>2</td>
<td>MEMPHIS deliverable D7</td>
<td>Official document describing final specs after user needs analysis...</td>
<td>0.70</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

After post-processing, the User can be given results in any format desirable, e.g.:

```
Your search found 3 items with relevance above 75%:
1. Image 14, probability 93% (for subject 1). “Shot in the countryside, by a river”.
2. Image 9, probability 79% (for subject 2). “A group playing cards”.
3. Image 33, probability 76% (for subject 2). “–there is no description available–”.
```

### 7. Conclusions

This paper has offered a solution to an important accessibility problem on the Internet, by making it possible for non-privileged users to assess information which is not directly available to them at the time, without disclosing this information either to the user or to the search engine.

Information retrieval on the Internet is currently based on the assumption that the information is always available for free. Public search engines require unrestricted access to information sources. This assumption excludes information sources that offer premium services, thus seriously impairing their accessibility from the public. Corporate search engines have unlimited access to the information sources but are essentially developed independently of one another, allowing no room for experimentation by expert users or third-party search engine developers, thus greatly impeding the development of truly intelligent search algorithms.

We propose an architecture that allows a data provider to receive and execute a search request on its own premises, and return only the results of the search to the user. The results can be in a format that is acceptable according to the provider’s information disclosure policy (e.g. only document names and scores, or short excerpts etc). A search request comprises both the search parameters and the search algorithm itself. It should be stressed once more that the search algorithm does not have to be either known or trusted from the data provider. Furthermore, this task is carried out efficiently.

---

\(^1\) Described in section 4.1.
through the use of Mobile Agent technology (avoiding large user waiting times, CPU overload and large bandwidth consumption), making the system applicable to real-world-size search applications. The capability to search and report sites and documents of interest to potential customers could open a tremendous potential for companies whose business models are based on premium services. Specialist applications are also possible, where private data is available for assessment, but not normal viewing, by any experts participating in the system.

References:


