

# **Sentences Security White Paper**

**Lazysoft**  
technology

[www.lazysoft.com](http://www.lazysoft.com)

Copyright © 2003 Answerbrisk Ltd

## Introduction

Security is a vital aspect of every computing environment today, and one which Lazysoft takes very seriously indeed. This document summarises the issues surrounding the security of Sentences Enterprise Edition.

More detailed information may be found in the Lazysoft document "Sentences Security: How to Provide for Secure Access to a Sentences Application" and in the Sentences Users' Guide.

If you have questions concerning specific issues and operating environments, our Technical Support team has a wealth of experience and will be pleased to assist.

---

<b>How are the Data Files Secured? .....</b>	<b>2</b>
<b>How is Data Secured From Unauthorised Use? .....</b>	<b>3</b>
<b>How is Data Secured To and From the Database? .....</b>	<b>5</b>
<b>How are Individual Transactions Secured? .....</b>	<b>6</b>

---

## How are the Data Files Secured?

It is important to ensure that all Sentences data files are secure from unauthorised external access (ie. other than via Sentences). This section discusses the issues.

### File Ownership

All server operating systems used by Sentences support the concept of a file owner, which is a user account having privileged access to the file. The most secure solution for Sentences is to create a user account specially for Sentences, which is used to run both the servlet container and all Sentences batch functions, and to make that account the owner of all Sentences data files, denying access to all other users.

### File Encryption

Sentences data files may be readily encrypted. On a Windows 2000 Server, the Encrypted File System (EFS) supports encryption of all files within a selected folder. When requests for data are made, the required data is decrypted and handed to the requesting application. On update the un-encrypted data is handed back from the application to the operating system, which encrypts it prior to storage. The encryption key is stored in the file header, encrypted with a user's public key, which effectively restricts access to that data to a single user account.

There are a number of ways to encrypt UNIX and Linux file systems, but there are some issues to be aware of. Many UNIX variants ship with an encryption application called "crypt": however this is easily broken by several cracking programs available on the web. Also, many UNIX systems support the concept of a "super-user" who may access any file, including encryption keys.

### Preventing Physical Access

A malicious person with physical access to a server can usually gain access to its data, often simply by re-booting it using their own boot disk. On UNIX or Linux systems, this would allow them to make themselves a super-user, and to read any data on the system. On a Windows system, they could install a defective disk driver, and obtain access to the data on the hard disk. Preventing physical access to the server is therefore essential.

### Inherent Security

If all else fails and Sentences data files are maliciously accessed, they are nevertheless inherently far more secure than relational database files. The reason lies in the nature of associative data.

Relational tables store data as values in table rows, so if someone obtained illicit access to a relational table containing, for example, customer names, addresses and credit card details, it would be clearly apparent whose name was associated with which address and which credit card number, because the pieces of data would be physically contiguous in the database volume.

This would not be the case in an associative database such as Sentences. Whilst a malicious person may be able to see entities representing names, addresses and credit card details, they would not be physically contiguous (or even in the same volume or on the same server) and there would be no way, without an operative version of Sentences properly configured, to correlate the entities and extract the meaning of the data. The associations that link a name, an address and a credit card number would not be contiguous to the entities, and moreover they refer to the entities by indirect and meaningless binary surrogates that are impossible to decipher without Sentences itself.

## How is Data Secured From Unauthorised Use?

Having ensured that your data is physically secure, you will then probably wish to control who within your organisation may gain access to what data, and for which purposes. Sentences provides both global features, independent of the identity of individual users, and also mechanisms that require individual users to identify and authenticate themselves, subsequently controlling their access according to their identity or role.

### Global Features

#### **Password Protection**

A Sentences database comprises any number of data files, or "chapters". Each chapter may be independently password protected. A password-protected chapter cannot be used by Sentences until the correct password has been supplied.

As a safeguard, Sentences Personal Edition cannot open a password-protected chapter under any circumstances.

#### **Sentences Permissions**

Sentences provides the ability globally to enable or disable various forms of access to a database by setting certain properties for the entire installation. These include client access, API access, XML access, on-line backup and access to various types of statistics.

If a particular type of access, eg. API access, is not routinely used in an installation, then it should be disabled. The Sentences server will not then respond to any requests for this type of access.

This capability is independent of individual chapters, which may also be password-protected.

### User Authentication

To provide more fine-grained control, users may be required to authenticate themselves by providing a username and password, and the authentication information may then be used in several ways to restrict access to data.

The authentication system maintains usernames and passwords, together with a list of what each user is allowed to access. This data may be held in Sentences or externally. In either case, it is a good idea to use HTTPS to ensure that the password data entered by the user is encrypted. Once the web server has accepted the authorization string, all subsequent requests from that browser session to that server will carry it.

Sample code to achieve this, and use it in the various ways described below, is included with the Sentences Application Suite.

#### **Interaction with the Application**

Having authenticated a user, the next step is to allow or restrict access to the application and its data, depending on the settings for this particular user. This can be done in several ways, which provide increasingly detailed control at the expense of correspondingly greater effort.

#### **Using Profiles**

The simplest method is to control access using Sentences profiles. Each username has a list of profile names associated with it. After logging in, they must choose a profile, after which an HTML page is constructed dynamically specifying that profile and using the Sentences 'Fixed Profile' applet parameter to prevent them from switching profiles.

Detailed control over what data they can see and change is done by holding it in chapters that are or are not included in the corresponding profile. Sample code using a JSP which reads a configuration file holding the lists of available profiles is included with Sentences. With this technique, all users can be given access to the Sentences Explorer, and no further application programming is required.

### **Using Roles**

For more detailed control, each username may be linked to a set of authorities, or roles. A set of JSPs defines the actions that the application provides, and each page includes a menu which presents only the actions that the current user is allowed to access. The menu is built by a header JSP, which is called from each of the other JSP pages, which uses the information retrieved by a login servlet and stored in an authorization bean.

The authorization information, in the form of the bean, is held by the JSP session, and is accessible from a JSP. The information is also available to Sentences queries by including user-defined parameters in the HTML that embeds each applet. The user information can then be used in subset queries to dynamically restrict the data that any user sees.

## How is Data Secured To and From the Database?

Sentences runs in an intranet, extranet or internet environment, using HTTP as its basic transport protocol. There are various considerations involved in ensuring that data is secure on its way to and from Sentences.

### Sentences Servlets

Sentences provides its various services through servlets, which run within a web server. Securing a Sentences database thus begins with the web server itself. Access to the services provided by the Sentences servlets is achieved by accessing URLs, with a different URL for each different type of access: client access, API access, XML export, XML import, on-line backup, access to OLAP cubes and access to various types of statistics. The features of the web server should be used to limit access to these URLs.

### Sentences Applets

The Sentences user interface is implemented as a set of applets: the Explorer applet, the Dataform applet, and so on. To run an applet, it must be embedded within an HTML page. The HTML page can specify parameters to the Sentences applet to restrict access to the database. Therefore, to allow a user to have only limited access to the database, one would control the HTML pages to which they have access, and make sure that these HTML pages include Sentences applet parameters to further restrict access as required.

Any Sentences applet will refuse to run unless the HTML page in which it is embedded came from the same host. This check is needed to prevent a malicious user from making a local copy of a Sentences HTML page and altering the applet parameters to give themselves more access than they should be allowed.

The Sentences applets communicate with the Sentences server using a binary data format, which cannot be easily read by an eavesdropper.

### HTTPS

The HTTPS (HTTP over secure sockets) is a secure communications protocol supported in Java by means of the Java Secure Socket Extension. It may be used to protect sensitive network traffic from eavesdropping via encryption.

HTTPS may also be used to protect the data traffic passing between the Sentences applets and the Sentences servlets. The URLs which are used to access Sentences servlets may be readily configured to use HTTPS.

Sentences applets, by default, use the same protocol that was used to load the HTML page that the applet is embedded in. In other words, if you access the page via HTTP, the applet will connect via HTTP; if you access it via HTTPS, the applet will connect via HTTPS.

### Restricting Access by IP Address

Some web servers allow access to particular URLs to be restricted to specified IP addresses. This may be used to limit access to various Sentences services to specific ranges of client machines.

If the web server does not provide this feature, you may use a security servlet to monitor access to Sentences.

## How are Individual Transactions Secured?

A database reflects the state of some part of the real world. When something in the real world changes, the database must change too to ensure that it still accurately reflects the real world. The thing that causes the database to change in response to a change in the real world is called a transaction.

Every database transaction must be **atomic, consistent, isolated** and **durable**. Atomicity and durability are preserved by database recovery mechanisms, which handle various types of hardware, software and other system failures. Consistency is preserved through constraint mechanisms. Isolation is preserved by database concurrency mechanisms.

The associative model of data used by Sentences implements an “add only” update philosophy, which has implications for atomicity and isolation that are discussed below. Under the associative model, all updates, including changes and deletions, are accomplished by adding items and links to the database. Each transaction is also itself represented by an item, which has links to other items and links to record its own attributes.

In Sentences, the atomic action that commits the transaction to the database is the addition of the link between the item representing the transaction and the item representing the timestamp that records the date and time at which the transaction was committed. We call this the “commit link”, and it is the last link in the following example:

```
XYZ Ltd address 125 Canary Wharf, London E6 5TT  
... deleted by Transaction 97756392  
XYZ Ltd address Montague Court, London EC3 4RR  
... created by Transaction 97756392  
Transaction 97756392 committed at 2002/03/21 15:57:01.51
```

Sentences incorporates the concept of a visibility layer, which operates close to the physical storage medium. The job of the visibility layer is to filter out all items and links that are not currently logically visible for any reason. The visibility layer applies, amongst others, the following rules:

- Any transaction item whose commit link is not visible is itself not visible;
- Any item or link created by a transaction that is not visible is itself not visible;
- Any item or link logically deleted by a stop link created by a transaction that is not visible is itself visible.

Thus, during a transaction, the items and links that the transaction creates, including stop links, are added to the database but remain invisible because the transaction itself is invisible. At the commit point, the commit link is added and the transaction is thereafter visible. If the transaction fails to commit for any reason, including system failure, the commit link will not be added and the transaction is thus never committed. Items and links created by uncommitted transactions remain in the database, but are filtered out by the visibility layer, and may be physically deleted at any time by housekeeping routines.

### Atomicity

Atomicity means that a transaction must be “all-or-nothing”: either the entire transaction is effective, or the transaction has no effect. If the transaction succeeds, then its entire effect is reflected in the database. If a transaction fails, then it has no effect on the database.

The associative model preserves atomicity through the commit mechanism described above. A transaction is rendered visible by the atomic action of adding its commit link to the database. Once this has happened, the entire transaction is visible. If it fails to happen, none of the transaction is visible.

## Consistency

Consistency says that a transaction must not effect changes that cause any of the rules and integrity constraints expressed about data in the database to be violated: it must leave the database in a consistent state.

Sentences incorporates data type, cardinality, referential integrity and other constraint checks to ensure that only valid transactions are processed.

## Isolation

Isolation requires that a transaction must execute independently of any other transactions that may be executing at the same time ("concurrently"). It must appear to each transaction that every other transaction happened either before or after itself. The effect on the database of executing a number of transactions concurrently must be the same as if they were executed one after the other ("serially").

In the associative model, the commit mechanism described above also serves to preserve isolation. From the standpoint of the visibility layer, every transaction occurs at the precise instant recorded by its commit link, and a transaction is rendered visible by the atomic action of adding its commit link to the database. Thus no transaction that is in progress and has not yet committed may ever see any part of any other transaction that may also be in progress.

## Durability

Durability requires that, once a transaction has succeeded, the changes that it has made to the state of the database must be permanent. Once atomicity and consistency have been assured, the principal mechanism to ensure durability is the ability to make and restore backups of the database. Sentences provides the capability to take backups whilst the database is in use.

*(Database transaction processing, concurrency and recovery are all major subjects in their own right. The fundamental principles of each types of mechanism are relevant to all database architectures, including the associative model, although much of the literature naturally focuses on their application to the relational model. Most introductions to database concepts will provide a good overview. We recommend:*

- *R. Elmasri and S. B. Navathe: "Fundamentals of Database Systems, Second Edition", The Benjamin/Cummings Publishing Company, Inc, 1994.*
- *C. J. Date: "An Introduction to Database Systems, Six Edition", Addison-Wesley, 1994.*

*For greater depth, we also recommend:*

- *J. Gray, A. Reuter: "Transaction Processing: Concepts and Techniques", Morgan Kaufmann, 1993.*
- *Vijay Kumar, Meichun Hsu: "Recovery Mechanisms in Database Systems", Prentice Hall PTR, 1998.)*