Secure Access Layer

Alias:

Using Low-level security
Using Non-application security
Only as strong as the weakest link

Motivation:

When secure documents are transferred from one secure area to another in the military base, it is important that the security of the documents is not violated during the transfer. If the document is being transferred via a computer disk, the data could be encrypted and then locked in a briefcase and handcuffed to the arm of the courier during transfer. This will provide an isolation layer to protect the secure information during the transfer.

Most applications tend to be integrated with many other systems. The places where system integration occurs can be the weakest security points and the most susceptible to break-ins. If the developer is forced to put checks into the application wherever the applications communicates with these systems, then the code will be very convoluted and abstraction will be difficult. An application that is built on an insecure foundation will be insecure. In other words, it doesn't do any good to bar your windows when you leave your back door is wide open.

Problem:

Application security will be insecure if it is not properly integrated with the security of the external systems it uses.

Forces:

- Application development should not have to be developed with operating system, networking, and database specifics in mind. These can change over the life of an application.
- Putting low-level security code throughout the whole application makes it difficult to debug, modify, and port to other systems.
- Even if the application is secure, a good hacker could find a way to intercept messages or go under the hood to access sensitive data.
- Interfacing with external security systems is sometimes difficult.
- An external system may not have sufficient security, and implementing the needed security may not be possible or feasible.

Solution:

Build your application security around existing operating system, networking, and database security mechanisms. If they do not exist, then build your own lower-level security mechanism. On top of the lower-level security, build a secure access layer for communicating in and out of the program.

Usually an application communicates with many pre-existing systems. For example, a financial application on a Windows NT client might use an Oracle database on a remote server. Given that most systems already provide a security interface, develop a layer in your application that encapsulates the interfaces for securely accessing these external systems. All communication between the application and the outside world will be routed through this secure layer.

The important point to this pattern is to build a layer to isolate the developer from change. This layer may have many different protocols depending upon the types of communications that need to be done. For example, this layer might have a protocol for accessing secure data in an Oracle database and another protocol for communicating securely with Netscape server through the Secure Sockets Layer (SSL) [Netscape]. The crux of this pattern is to componentize each of these external protocols so they can be more easily secured. The architecture for different *Secure Access Layers* could vary greatly. However, the components' organization and integration is beyond the scope of this pattern.

By creating a *Secure Access Layer* with a standard set of protocols for communicating with the outside world, an application developer can localize these external interfaces and focus primarily on applications development. Communicate in and out of the application will pass through the protocols provided by this layer.

This pattern assumes a convenient abstraction is possible. For example, VisualWorks' LensSession does not support Microsoft Access, so QueryDataManager cannot be used with a Microsoft Access database. *Secure Access Layer*, however, provides a location for a more general database abstraction. Third party drivers have been developed for ODBC that can communicate with Microsoft Access. By using the *Secure Access Layer*, it is easy to extend your application to use the ODBC protocol, thus allowing your application to communicate with any database that supports ODBC.

Example:

The PLoP registration program uses a *Secure Access Layer*. A layer was created where all communications is processed for registering through the web. This communications layer is positioned on top of Apache's Secure Socket Layer. This prevents any information from being sniffed during the entry of data such as credit card numbers. Also, a layer on the database side was also created to provide additional security by encrypting the credit card information in the database. The secure layer uses a key for encrypting and decrypting the data when needed. Thus, even if someone was able to access the database through some back door, the credit card data is still protected.

Consequences:

✔ A Secure Access Layer can help isolate where an application communicates with external security systems. Isolating secure access points make it easier to integrate new security components and upgrade existing ones.

✔ A *Secure Access Layer* can make an application more portable. If the application later needs to communicate with Sybase rather than Oracle, then the access to the database is localized and only needs to be changed in one place. *QueryObjects* [Brant & Yoder 96] uses this approach by having all accesses to the database go through the QueryDataManager, which is built on top of the LensSession [OS 95]. The LensSession can map to either Oracle or Sybase. Therefore the application developer does not need to be concerned with either choice or future changes.

✕ Different systems that your application may need to integrate with use different security protocols and schemes for accessing them. This can make it difficult to develop a *Secure Access Layer* that works for all integrated systems, and it also may cause the developer to keep track of information that many systems do not need.

✕ It can be very hard to retrofit a *Secure Access Layer* into an application which already has security access code spread throughout.

Related Patterns:

- *Secure Access Layer* is part of a layered architecture. *Layers* [BMRSS 96] discusses the details of building layered architectures.
- *Layered Architecture for Information Systems* [Fowlers 97-1] discusses implementation details that can be applied when developing layered systems.

Known Uses:

- Secure Shell [SSH] includes secure protocols for communicating in X11 sessions and can use RSA encryption through TCP/IP connections.
- SSL (Netscape Server) provides a *Secure Access Layer* that web clients can use for insuring secure communication.
- Oracle provides its own *Secure Access Layer* that applications can use for communicating with it.
- CORBA Security Services [OMG] specifies how to authenticate, administer, audit and maintain security throughout a CORBA distributed object system. Any CORBA application's *Secure Access Layer* would communicate with CORBA's Security Service.

- The Caterpillar/NCSA Financial Model Framework [Yoder] uses a *Secure Access Layer* provided by the LensSession in ParcPlace's VisualWorks Smalltalk [OS 95].
- The PLoP '98 registration program [Yoder & Manolescu 98] goes through a *Secure Layer* for access to the system. First the application runs on top of SSL from the Apache Server. Also, all credit card information is stored encrypted when users registered for PLoP '98.