## Single Access Point

### *Alias:*

Login Window
One Way In
Guard Door
Validation Screen

### *Motivation:*

A military base provides a prime example of a secure location. Military personnel must be allowed in while spies, saboteurs, and reporters must be kept out. If the base has many entrances, it will be much more difficult and expensive to guard each of them. Security is easier to guarantee when everyone must pass through a single guard station.

It is hard to provide security for an application that communicates with networking, operating systems, databases, and other infrastructure systems. The application will need a way to log a user into the system, to set up what the user can and can not do, and to integrate with other security modules from systems that it will be interacting with. Sometimes a user may need to be authenticated on several systems. Additionally, some of the user-supplied information may need to be kept for later processing. *Single Access Point* solves this by providing a secure place to validate users and collect global information needed about users who need to start using an application.

### *Problem:*

A security model is difficult to validate when it has multiple “front doors,” “back doors,” and “side doors” for entering the application.

### *Forces:*

- Having multiple ways to open an application makes it easier for it to be used in different environments.
- An application may be a composite of several applications that all need to be secure.
- Different login windows or procedures could have duplicate code.
- A single entry point may need to collect all of the user information that is needed for the entire application.
- Multiple entry points to an application can be customized to collect only the information needed at that entry point. This way, a user does not have to enter unnecessary information.

### *Solution:*

*Set up only one way to get into the system, and if necessary, create a mechanism for deciding which sub-applications to launch.*

The typical solution is to create a login screen for collecting basic information about the user, such as username, password, and possibly some configuration settings. This information is passed through some sort of *Check Point* that verifies the information. A *Session* is then created based upon the configuration settings and the user’s access privileges. This *Session* is used to keep track of the global information that deals with the user’s current interaction with the application. When opening up any sub-application, requests are forwarded through the *Check Point* for handling any problems and validations.

UNIX is an example of a system with multiple access points. Each network port can provide access to a separate service. These multiple access points make it difficult to secure a UNIX system as a hole. Individual applications running on a UNIX system can still be secured, however. For example if an application has web, email, and programming interfaces, steps must be taken to ensure that each of these interfaces are forced through a shared *Single Access Point* before accessing the rest of the application.

The "Putting It All Together" section describes the initialization process in more detail. The important idea here is that *Single Access Point* provides a convenient place to encapsulate the initialization process, making it easier to validate that the initial security steps are performed correctly.

## *Example:*

There are many examples of *Single Access Point*. In order to access an NT workstation, there is a single login screen which all users must go through to access the system. This *Single Access Point* validates the user and insures that only valid users access the system and also provides *Roles* for only allowing users to see and do what they have permissions to do. Most UNIX systems also have *a Single Access Point* for getting a console shell. Oracle applications also have many applications such as SQLPlus and the like that provide a *Single Access Point* as the only means for running those applications.

## *Consequences:*

✔ A *Single Access Point* provides a place where everything within the application can be setup properly. This single location can help ensure all values are initialized correctly, application setup is performed correctly, and the application does not reach an invalid state.

✔ Control flow is simpler since everything must go through a single point of responsibility in order for access to be allowed. Note, *Single Access Point* is only as secure as the steps leading up to it.

✕ The application cannot have multiple entry points to make entering an application easier and more flexible.

## *Related Patterns:*

- *Single Access Point* validates the user's login information through a *Check Point* and uses that information to initialize the user's *Roles* and *Session.*
- A *Singleton* [GHJV 95] could be used for the login class especially if you only allow the user to have one login session started or only log into the system once. A Singleton could also be used to keep track of all Sessions and a key could be used to know which session to use

## *Known Uses:*

- UNIX telnet and Windows NT login applications use *Single Access Point* for logging into the system. These systems also create the necessary *Roles* for the current *Session*.
- Most application login screens are a *Single Access Point* into programs because they are the only way to startup and run the given application.
- The Caterpillar/NCSA Financial Model Framework [Yoder] has a FMLogin class, which provides both *Single Access Point* and *Check Point*.
- The PLoP '98 registration program [Yoder & Manolescu 98] provides a *Single Access Point* for logging into the system and entering in credit card information when users registered for PLoP '98.
- Secure web servers, such as Java Developer's Connection appear to have multiple access points for each URL. However, the web server forces each user through a login window before letting them download early access software.

## *Non-security Known Uses:*

- Any application that launches only one way, ensuring a correct initial state.
- Windows 95, also uses a login window which is a *Single Access Point*, but it is not secure because it allows any user to override the login screen.
- Single creational methods provide for only one way to create a class. For example, Points in VisualWorks Smalltalk [OS 95] guides you to creating valid points by providing a couple of creational methods that ensure the Object is initialized correctly. Kent Beck's describes *Constructor Methods* as a single way to create well-formed instances of objects [Beck 97]. These are put into a single "instance creation" protocol. This becomes the *Single Access Point* to create new objects.

- *Constructor Parameter Method* [Beck 97] initializes all instance variables through a single method, which is really a *Single Access Point* for that class to initialize its instance variables.
- Concurrent programs can encapsulate non-concurrent objects inside an object designed for concurrency. Synchronization is enforced through this *Single Access Point*. *Pass-Through Host* design [Lea 97] deals with synchronization by forwarding all appropriate methods to the Helper using unsynchronized methods. This works because the methods are stateless with respect to the Host class.