13.7 Front Door

Web applications and services often need to identify a user and keep track of a user's session. Integrating several such services allows a single log-in and session context to be provided. A reverse proxy is an ideal point to implement authentication and authorization, by implementing a Web entry server for your back-ends. A sophisticated reverse proxy can even access external back-ends, providing the user's id and password automatically from a 'password wallet.'

Also Known As

Web Entry Server, Web Single Sign On.

Example

Let us continue with the Myshop.com example. Soon after the INTEGRATION REVERSE PROXY (465) was deployed, users complained that they had to re-enter their identity several times on the Web site. Myshop.com's IT personnel recognized that each Web application carried its own user database. Adding an application that required user authentication only meant adding another user data base. Providing support services to their customers and resellers via the Web required more sophisticated authentication, and they wanted to allow access only to those users who paid for the service. See figure on page 474.

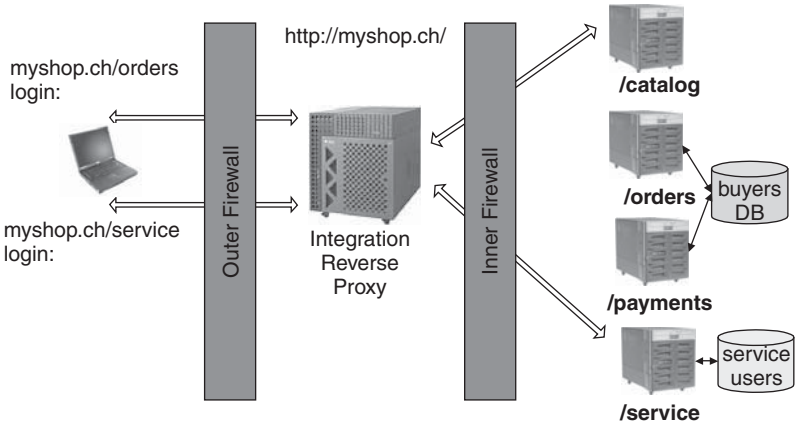
How can Myshop.com provide access control to their Web applications easily, without requiring users to sign on several times, and with support for extensibility?

In addition, the CIO recognizes that new means of user authentication can become popular in the future, so doesn't want the different applications to depend on a single authentication schema. For example, Myshop.com might give security tokens that generate one-time passwords to their resellers, to add a more secure authentication for users who place bulk orders.

Context

A Web site consisting of multiple Web applications that require user authentication.

An INTEGRATION REVERSE PROXY (465) where applications need to authenticate users, and where only authenticated users are authorized to access a defined subset of applications, a PROTECTION REVERSE PROXY (457) where user authentication is required, and where only authenticated users will get access to the underlying Web application, or a combination of both.



Supporting multiple databases through an INTEGRATION REVERSE PROXY

Problem

How do you provide a single sign on for several Web applications or services?

The solution to this problem must resolve the following forces:

- You want a single user identity for all applications, even when existing Web applications already carry their own user data base.
- You do not want users to have to provide their password for each application separately, depending on your security policy.
- You might want to force users to identify themselves several times, to avoid misuse of a user's session that is left alone for some time.
- You want your applications to be independent of the authentication schema used. Depending on your security policy, you might even require different schema. For example, strong authentication with a one-time password from a security token for payment service, or weak authentication using a regular id-password combination for service access.
- Different users have different access rights to your systems. You want to be able to handle these differences with a single solution.
- You want new applications to easily integrate into your authentication– authorization schema.
- You want both a single sign-on and a single log-off. That means that a user should keep his session as long as he is active, regardless of the concrete back-end he interacts with. On the other hand, when a user logs off, his session should be terminated, so that even when the browser is left open, nobody else can connect to the back-end servers without re-authentication.

Solution

Implement a FRONT DOOR (473) server as a specialization of the INTEGRATION REVERSE PROXY (465) that identifies users and keeps track of user sessions. This server passes user identity and session identification to all of the back-ends. The FRONT DOOR (473) can log all user activity in a central log. Depending on the nature of the complete solution, some back-end servers might be accessible by everyone, and the FRONT DOOR (473) only protects some back-end servers from unauthenticated users. Nevertheless, remember that it can also act as a PROTECTION REVERSE PROXY (457) for the public part of the Web site.

You need to consolidate user identities held in existing back-end applications. Store the resulting user profiles by combining a user's identities and access rights in a single user directory. Currently an LDAP directory server is the popular solution for that, but another kind of data base might also be appropriate.

A system for managing user identities and access rights is beyond the scope of this pattern, but is often required. In large solutions that use a vendor's solution for access rights, management can be effective, or you might be able to extend an Active Directory when you are using Windows.

Structure

You end up with the following structure if you apply FRONT DOOR (473) to the scenario from our example. The usual place for the machine hosting the FRONT DOOR (473) service is the DEMILITARIZED ZONE (449). See figure on page 476.

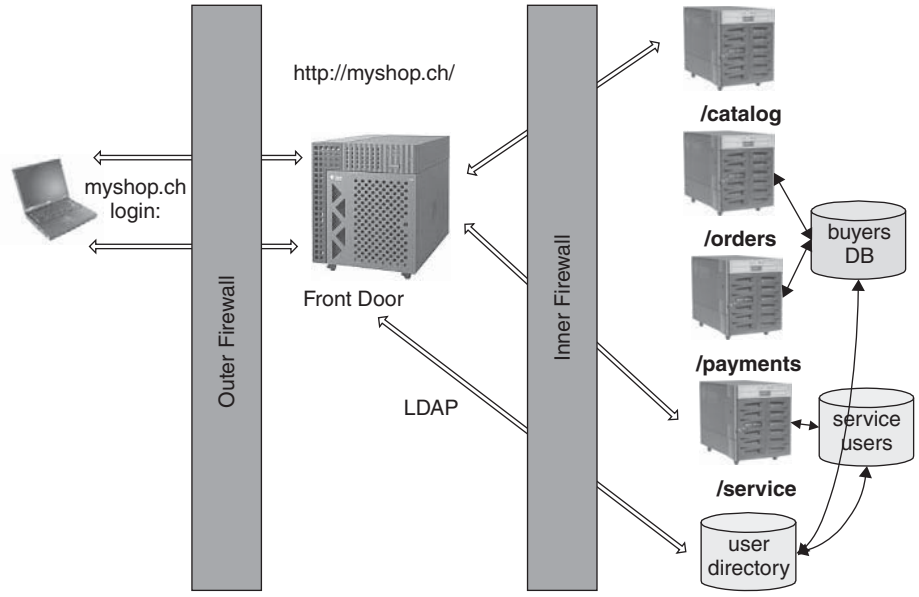
Dynamics

In addition to INTEGRATION REVERSE PROXY (465)s mapping to back-end servers, FRONT DOOR (473) adds another pre-processing step by first checking a user's permissions. Depending on the result of this check, the request is either routed to the desired back-end server, to a log-in page for a user to authenticate themselves, or, in the case of access denial, to an error page. As with PROTECTION REVERSE PROXY (457), FRONT DOOR (473) might chose to silently drop unauthorized requests and just log the access attempts.

Implementation

To implement a FRONT DOOR (473) reverse proxy the following must be considered in addition to the issues given in the preceding patterns:

1. *Unify user representations and data base*. This is easiest if you have a clean start, or if only one user database exists. An LDAP directory server is a popular



Adding a FRONT DOOR

means for storing user identities, passwords and access rights. If you want to integrate existing back-ends that you cannot change, you might need to add an identity and password wallet to each user object in the directory, to enable automatic replay of id and password when accessing such a back-end.

2. *Define authentication mechanism(s).* Popular mechanisms are id-password, one-time passwords, one-time token based password, challenge-response with token, biometrics, certificates, or any useful combination of these. Since now only the FRONT DOOR (473) needs to implement user authentication, it is easy to change or extend authentication mechanisms later without any impact on existing applications. See Chapter 7, *Identification and Authentication (I&A)* for more information.
3. *Define access rights schema if needed.* Different approaches exist for representing access rights and the mapping of users to the set of allowed services. For the purpose of the FRONT DOOR (473), a coarse-grained model is sufficient, but individual applications might need fine-grained control to internal functionality. A sophisticated implementation will provide a complete model applicable not only for the FRONT DOOR (473), but also for all an application's needs. Chapter 8, *Access Control Models* provides some insights into these issues.
4. *Design user and session representation* as passed to back-ends via header fields. This can be a specifically-named header field, or you can use HTTP's

basic authentication mechanism to pass on the user identity. If there is no single user representation, FRONT DOOR (473) might need to map the user id to the one specific to the back-end. This mapping needs to be stored in the user data base as designed in step 1. Optionally define additional header fields for inter-back-end communication. Those header fields are analyzed by FRONT DOOR (473), kept in its session store and automatically passed to all interested back-ends.

This and the following three steps correspond roughly to activities described in SECURITY SESSION (297)'s Implementation section.

5. *Design and implement how FRONT DOOR (473) keeps track of user sessions.* Some solutions that rely on SSL for browser-FRONT DOOR (473) communication use the SSL session id for that purpose. Using a session cookie is also popular. Rewriting all URLs to add a session id in the content of back-end replies, if cookies are disabled, seems too great an overhead and too complex. Using cookies, it is even possible to be able to keep the session context when switching between HTTP and HTTPS, either for performance or security reasons. FRONT DOOR (473)'s session cookies should be encrypted and cryptographically signed to ensure that they cannot be manipulated. If FRONT DOOR (473)'s cookies are secured like this, and they also contain some identification of their source, FRONT DOOR (473) can even accept such a cookie as a valid user identification after a crash without the user being aware of the session's re-authentication.

6. *Design and implement FRONT DOOR (473)'s session context.* The session cookie can be the means to store all session context. However, because of a cookie's size limitation and security issues, it might be better to keep the session context on the server side. One solution is to keep a session list with all session contexts in memory. This is the most efficient solution, especially if the access rights of a user are also cached there, but it carries the risk of losing session state on a crash. Another option is to use persistent storage in a data base for session context. However, this tends to be an order of magnitude slower, but allows for several FRONT DOOR (473) instances to share session context. Which solution for keeping session context is best depends on the concrete requirements. For more explanations for keeping session state, refer to the chapter *Session State Patterns* in [Fow03].

7. *Implement a cookie jar.* If back-end servers use their own session cookies, FRONT DOOR (473) can keep those session cookies in its own session context and not pass them to the user's browser. This ensures single log-off. If this is not done, a browser might send an old application session cookie after a new user logged into FRONT DOOR (473), confusing the back-end server.

8. *Design and implement log-in and portal pages.* FRONT DOOR (473) can delegate user identification to a special back-end server, or it can implement its

own log-in page. As with INTEGRATION REVERSE PROXY (465), a portal page that consists of a menu of all services available to the logged-in user is a possible poor-man's portal solution. In addition, a special service link (for example, `/logoff`) should be implemented by FRONT DOOR (473) to allow applications to give the user the ability to consciously terminate their session.

Apart from its visual nature, this corresponds to CHECK POINT (287) in a FRONT DOOR (473) user's experience.

Variants

As with INTEGRATION REVERSE PROXY (465) you can deploy two FRONT DOOR (473)s that share back-end servers, one for the Internet (effectively making it an extranet) and one for intranet users.

Known Uses

Peter Sommerlad's former company's Frontdoor solution for Telekurs Financial Services Ltd. implements most of the issues given here, in addition to being able to be configured as a PROTECTION REVERSE PROXY (457). The underlying application framework in C++ should be available as open source software when this book is in print.

Bull Evidian PortalXpert [Evidian] implements a Web Entry Service.

IBM Tivoli Access Manager [Tivoli] provides FRONT DOOR (473) reverse proxy functionality with its Web Seal product.

Consequences

In addition to the consequences of PROTECTION REVERSE PROXY (457) and INTEGRATION REVERSE PROXY (465), this pattern implies the following **benefits**:

- *Single sign-on and single log-off*, because FRONT DOOR (473) keeps track of a user's session, and back-ends automatically obtain the user id from FRONT DOOR (473) instead of asking the user for it again.
- *One user profile is possible across back-end applications*. This is not necessarily the case, for example if you start with several existing Web applications and integrate them, but FRONT DOOR (473) facilitates the mechanisms that enable you to move implement a solution that uses one user profile and one administration application.
- *Applications are relieved from implementing access control and user authentication*. This gives you the opportunity to deploy Web applications quickly that readily integrate with FRONT DOOR (473)'s access control. Experience shows

that such an architectural guidance for Web applications can be a great benefit, especially on an intranet.

- *Centralized logging* allows user tracking and reporting. Marketing departments might die for such logs, which keep a detailed track of user activity.

However, in combination with the previous patterns' liabilities, FRONT DOOR (473) carries the following additional **liabilities:**

- Applications might enforce their own user database, increasing the risk of inconsistencies. For example, RSA's ACE/Server has its own user database for managing tokens for its strong authentication. If you implement FRONT DOOR (473) using both RSA SecureID and another user authentication schema, you end up with two user databases you need to synchronize.
- A central management application for user identities and access rights is needed. Without a single sign-on, this need can exist already, but might not be recognized. Deploying FRONT DOOR (473) makes this need prominent. Also a lack of corresponding organizational processes is more easily shown up.
- Password aging policies across back-end applications can conflict. You then need to auto-generate new passwords when they expire, or let the user worry about changing their password on the back-end application and in their FRONT DOOR (473) profile.
- Conflicting session time-outs of FRONT DOOR (473) and applications can confuse users.

See Also

You can view FRONT DOOR (473) as adding CHECK POINT (287) and SECURITY SESSION (297) to an INTEGRATION REVERSE PROXY (465) or PROTECTION REVERSE PROXY (457), and thus also providing a SINGLE ACCESS POINT (279) to a company's Web applications and services.