Reverse Proxy Patterns

Author Peter Sommerlad
Erlenstrasse 79, CH-8832 Wollerau, Switzerland, +41-79-432 23 32
psommerlad@hispeed.ch

Overview Implementing an application-level server-side proxy[1] [GHJV95][POSA96] can result in a large number of positive consequences. However, the aspects of network security, single sign on and integration imply different forces upon such a reverse proxy. Attaching the surrounding infrastructure can show additional roadblocks for a successful deployment.

The following patterns try to structure the forces, regarding the different aspects into three patterns, that can be studied to understand reverse proxy solutions and applied to design reverse proxy architectures.

The most popular reverse proxies implement the hypertext transfer protocol (HTTP), therefore the rest of the paper just refers to such HTTP reverse proxies. Nevertheless, the underlying patterns are also applicable for any other Internet protocols, for example FTP.

The Protection Reverse Proxy pattern shows how to protect your servers on the application protocol level at the network perimeter. An Integration Reverse Proxy allows to integrate a collection of servers under a common entry point, thus hiding the network and host internals. The Front Door pattern gives guidance for single sign on and access control to a set of web applications.

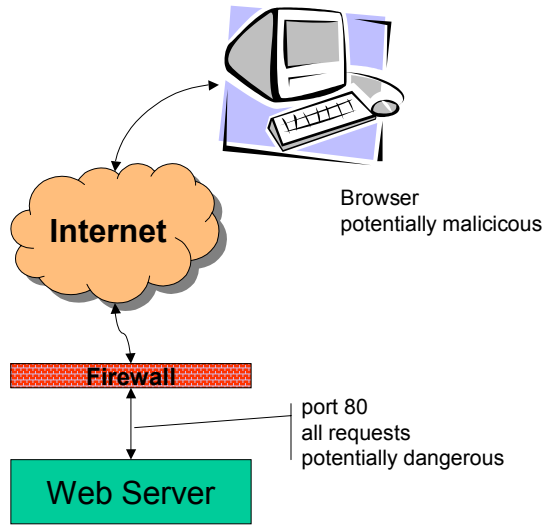
---

1. In contrast to a “regular” proxy configured within a user’s browser, such a transparent server-side proxy is called a “reverse proxy”.

Protection Reverse Proxy

Putting a web server or an application server directly on the Internet gives attackers direct access to any vulnerabilities of the underlying platform (application, web server, libraries, operating system). However, to provide a useful service to Internet users, access to your server is required. A packet filter firewall shields your server from attacks on the network level. In addition a Protection Reverse Proxy protects the server software on the level of the application protocol.

Example You are running your web site using a major software vendor's web server software. Your web site uses this vendor's proprietary extensions to implement dynamic content for your visitors and you have invested heavily in your website's software. Your server is protected by a packet filter firewall.



You must open this firewall to allow access to the public port (80) of your web server. Attacks from the Internet exploiting vulnerabilities of your server software burden your system administrator with installing patches frequently. Switching to another vendor's web

server is not possible because of the existing investment in the web server platform, its content and your own software extensions. In addition, with every new patch you install, you run the risk of destabilizing your configuration so that your system extensions cease to work, that your software extensions cease to work. How can you escape the dilemma to keeping your web site up without compromising its security?

Context Any kind of service accessible through the Internet or a through another potentially hostile network environment. Usually the access protocol is HTTP or HTTPS.

Problem Even if you install a simple packet filter firewall [add xref] your web server can remain vulnerable to attacks exploiting weaknesses in its protocol implementation. How can you protect your web server infrastructure in the light of its potential vulnerability to attacks using its protocol?

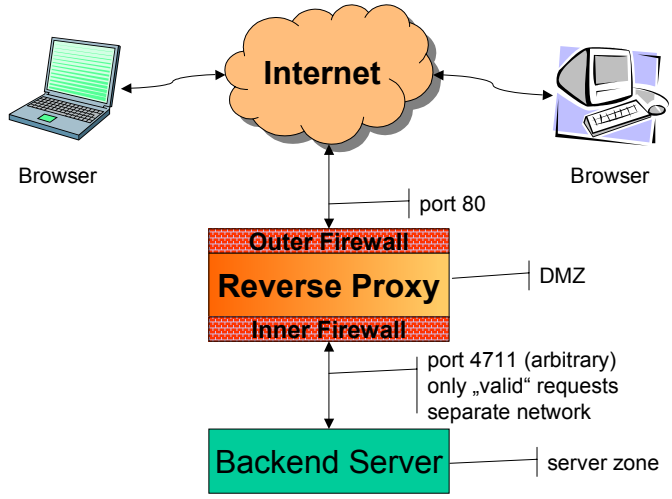
In particular you want to address the following forces:

- A simple packet filter firewall is not enough to protect your web server, since access to its protocol (e.g. port 80) must be provided to the Internet.
- Attack scenarios often employ extra long, or extra crafted request parameters to exploit buffer overflows. Most firewalls work on the network packet level and cannot prohibit attacks using such invalid requests.
- Hardening your web server can be beyond your capabilities. For example, because it comes as a black box from your vendor or because it is too complex.
- Installing patches to your web server platform helps to avoid exploitation of known vulnerabilities. But with each patch you risk that your system extensions cease to work. You need to rerun your integration tests at each patch level and might need to keep your extensions up to date with each patch level. It might even be impossible to upgrade your web server in a timely manner, because the extensions aren't ready.
- Switching to another web server software by a different source is expensive, risky and time consuming, too. A new web server might

have fewer vulnerabilities, but you are less familiar with it. In addition it might also require to adapt your own system extensions.

- You cannot know about vulnerabilities detected in the future.

Solution Change your network topology to use a protection reverse proxy that shields your real web server. Configure this reverse proxy to filter all requests, so that only (mostly) harmless requests will reach the real web server. Two packet filter firewalls ensure that no external network traffic reaches the real web server. The resulting network topology provides a demilitarized zone (DMZ) containing only the reverse proxy machine and a secured server zone containing the web server.



Also this solution talks only about web servers, it applies to other protocols like ftp, imap, smtp as well. A Protection Reverse Proxy for ftp, for example, might scan file content for viruses or executable and prohibit upload of such files, or it can limit the available ftp commands and prohibit third party host data connections, which are allowed by the ftp standard.

Component Browser	***Collaborators*** Outer Firewall Reverse Proxy Backend Server
Responsibility • issues potentially malicious requests to backend server via firewall and reverse proxy • retrieves backend reply via reverse proxy and firewalls	

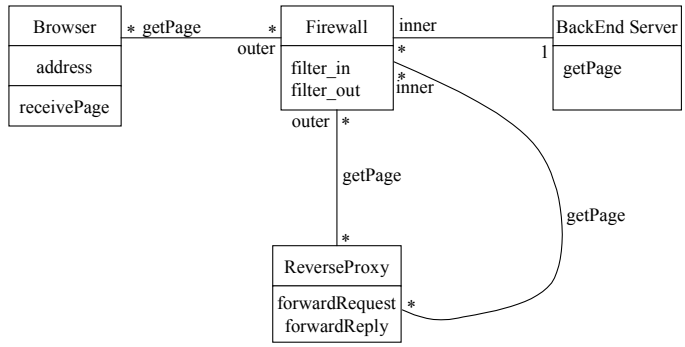
Component Outer Firewall	***Collaborators*** Browser Reverse Proxy
Responsibility • filters incoming network traffic and allows only HTTP port access to the Reverse Proxy • can deny outbound connection from Reverse Proxy	

Component Reverse Proxy	***Collaborators*** Outer Firewall Inner Firewall
Responsibility • accepts requests from browsers and forwards only valid requests to backend server. • passes reply from backend server back to originating browser.	

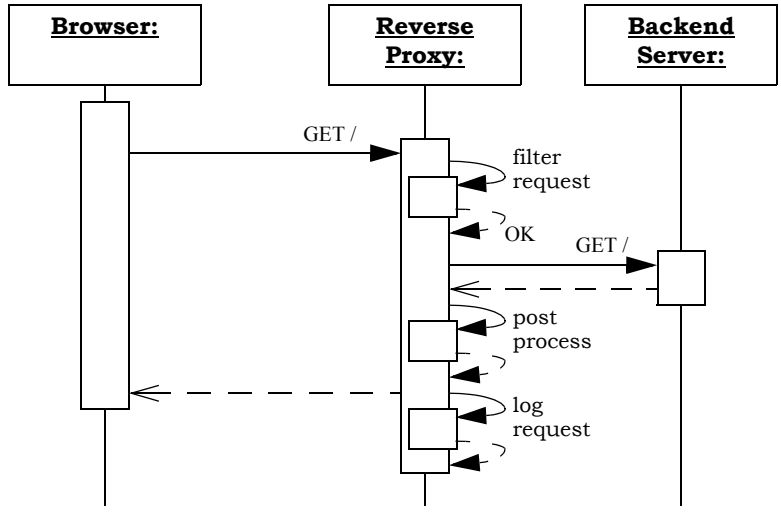
Component Inner Firewall	***Collaborators*** Backend Server Reverse Proxy
Responsibility • separates server zone from DMZ • denies inbound connections except from Reverse Proxy • denies outbound connection from backend servers	

Component Backend Server	***Collaborators*** Inner Firewall
Responsibility • provides the real web service. • accepts requests from reverse proxy and returns reply back.	

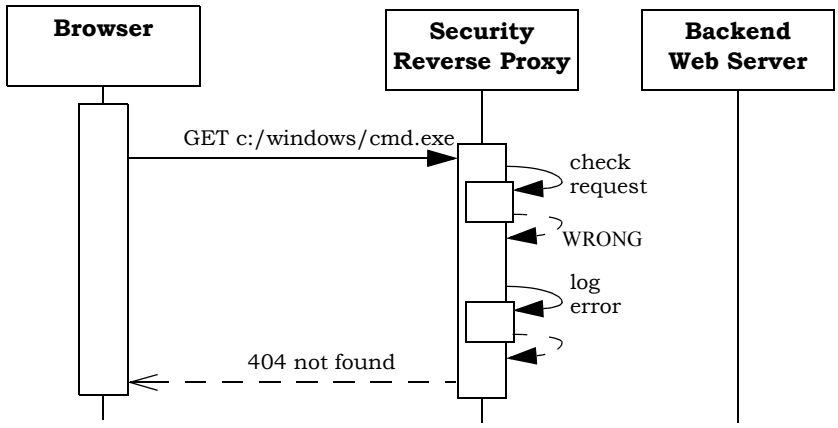
Structure



Dynamics The first scenario shows how a valid request is checked and passed on by the protection reverse proxy. The inner and outer firewall components are assumed to be transparent in that case and thus are not shown. Post processing a backend server's reply is optional, but can be used to adjust protocol header fields, for example. Note that the access log is only written after the reply was sent to improve responsiveness of the system.



The second scenario demonstrates the blocking mechanism of the protection reverse proxy by ignoring an invalid and thus potentially malicious request. Nevertheless, even if the browser will not get an answer the attempt will be logged. According to the official hypertext transfer protocol, the reverse proxy should return an error code (typically “403 forbidden” or “404 not found”). It depends on your security policy in that case, to either give an error reply or silently ignore the attempt and close the connection at the protection reverse proxy.



Implementation

To implement the Protection Reverse Proxy several tasks need to be done:

1 *Plan your firewall and network configuration.* Even if the firewall update is done after every other part is in place, it is good to start with a plan, so that configuration of the other components can rely on the firewall plan. Often the concrete configuration needs to consider more than just one protocol and some explicit “holes” in your firewall may be needed. Find out what protocol your reverse proxy solution needs to support. Typically only HTTP (port 80) is needed, but you might want to allow other protocols as well through your reverse proxy.

2 *Select a Reverse Proxy platform.* You might create your own reverse proxy, for example by configuring the Apache web server with mod_rewrite and mod_proxy modules, several vendors offer professional reverse proxy solutions, or you might need to implement your own reverse proxy, for example, because you are using a special protocol not supported by other solutions.

   Showing the details of implementing your own reverse proxy server software is beyond the scope of this pattern. Nevertheless, there are cases, where you might not trust a solution provider or not have one and only rely on your own skills.

   When selecting a vendor or source for your protection reverse proxy you should opt for a simple and proven solution. For example, using Apache you risk all Apache web server vulnerabilities to be present in your protection reverse proxy. On the other hand, the Apache web server is deployed so often, that most vulnerabilities and countermeasures are known.

3 *Configure your backend web server(s).* The web content should rely on relative path names and not use its internal name or IP address to refer to itself. Otherwise, links might not work, because the browser can no longer directly access the machine it is running on.

4 *Configure your Protection Reverse Proxy.* For the security to work, you need to define what requests should be mapped towards your backend web server, and define what to happen if invalid requests occur. For example, you might want to log what request were denied by the reverse proxy. For request filtering there exists two approaches: black lists and white lists.

   - A *black list* filter only blocks requests that its list of malicious requests knows of, but passes on all others. Black list filters are easier to deploy but riskier. They are often used by "higher-level" firewalls.
   - A *white list* filter is more restrictive and only lists allowed requests. It needs to be configured with detailed knowledge of the backend server and allowed URLs. A white list filter needs to be adapted every time your backend server changes significantly in its URL space. Nevertheless, it is the better choice for a Protection Reverse Proxy.

If your backend server relies on redirects or other mechanisms using its host address and you cannot change that, you need to configure your reverse proxy to modify server responses accordingly.

5 *Deploy everything*. Initial deployment with setting up firewalls, network and routers, host IP addresses and so on requires good planning. If you have something up and running already, this re-configuration might mean some service interruption. Nevertheless, later changes to the topology need only consider the reverse proxy and eventually the inner firewall.

Example Resolved Following these implementation guidelines we are able to protect our vulnerable web server with a Protection Reverse Proxy.

Known Uses Security Reverse Proxies are popular. Some organizations in the financial industry have the guideline to use a reverse proxy for every protocol provided over the Internet (with some exceptions, like DNS). Thus they can ensure that a vulnerable server is never directly accessible from the "wild".

Vendors of security infrastructure provide Security Reverse Proxies as part of broader infrastructure. Examples of such infrastructures at the time of writing are Bull Evidian Access Master and PortalXpert, IBM Tivoli Access Manager, SYNLOGIC Frontdoor and FTP Frontdoor.

Variants The patterns Integration Reverse Proxy and Front Door can (and should) be combined in their function with Protection Reverse Proxy and thus vary this pattern by adding functionality.

See also The Protection Reverse Proxy is a special implementation of Single Access Point [SecPat].

In conjunction with the regular firewalls, the Protection Reverse Proxy builds a "defense in depth" [BEAH][Schn03].

Consequences The pattern implies the following **benefits**:

- Attackers can no longer directly exploit vulnerabilities of the backend server. Even when the backend server gets compromised, the firewalls hinder further spreading of Internet worms, etc., by blocking outgoing requests from the backend server.

- Even with known vulnerabilities, you might be able to keep your web server configuration stable, because the Protection Reverse Proxy with its request filtering capability can prohibit exploitation of the web server's vulnerabilities.

- Easier patch administration. Only one machine remains connected to the Internet directly and needs to be monitored for potential vulnerabilities and existing patches to be applied. However, you cannot blindly trust your Protection Reverse Proxy. A backend server still needs to be configured with your brain on, to avoid exploitation of vulnerabilities with "allowed" requests.

- More benefits apply, when combined with more functionality. See the other patterns Integration Reverse Proxy and Front Door in this collection.

However, the Protection Reverse Proxy pattern also has its **liabilities**:

- Black list filtering can give you a false sense of security. Like patches, black lists can only be constructed after a vulnerability is known.

- White list filtering can be fragile, when backend servers change. Adding functionality, or re-arranging content structure on the backend web server, can imply additional work to re-configure the white list filter of the Protection Reverse Proxy.

- Latency. A reverse proxy adds latency to the communication, not only because of the additional network traffic, but also for the filtering and validation of requests.

- Some loss of transparency. Some restrictions are imposed on the backend servers. However, these are typically good practice anyway, like relative paths in URLs. Nevertheless, the backend servers no longer see the communication end partner directly on the network level. So the protocol may need to provide a means to identify the original communication end point (which HTTP allows).

- Additional point of failure. If the reverse proxy stops working, any access to your web site is impossible. Any additional component that can fail increases the overall risk of system failure. To reduce this

risk, you can provide a hot or cold stand by installation with hardware or software fail-over switches.

- Hardware, software and configuration overhead. The Protection Reverse Proxy requires to configure an additional packet filter firewall as well as another machine to run the reverse proxy on.

Integration Reverse Proxy

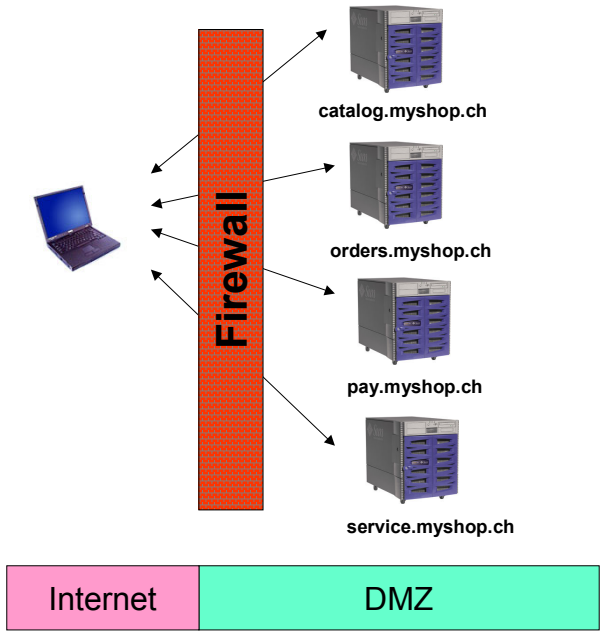
A web site constructed from applications from different sources might require several different servers, because of heterogeneous operating requirement of the different applications. Because of the Internet addressing schema, this distribution across several hosts is visible to the end user. Any change of the distribution or switch of parts of the site to a different host can invalidate URLs used so far, either cross-links of the web site or bookmarks set up by users. An Integration Reverse Proxy alleviates this situation by providing a homogenous view to a bunch of servers, without leaking the physical distribution of several machines to end users.

Example

Consider a typical web site of a company (myshop.ch) selling some goods and services. Their on-line presence was established with an interface to their support group, giving users access to static documentation, like an FAQ and a simple e-mail interface to contact support personnel. This web server runs on a machine name support.myshop.ch. Then the marketing department purchased an on-line catalog software showing their offerings on the server catalog.myshop.ch.

Later on they implemented a simple on-line ordering system with a small development company, since orders needed to be automatically routed to their home-grown ERP system. Because they used a different platform for development for cost reasons, this order taking system again needed to run on a separate server as order.myshop.ch.

To avoid problems with late paying customers and ease operation of their on-line business they added a credit card on-line payment software from another vendor. Again an additional machine was needed with its name pay.myshop.ch. They end with a structure shown in the diagram.



The business flourishes and now, their original infrastructure reaches some limits. But the practice of having every single server known on the Internet makes shifting applications to another server or running an application on two different systems hard. The complexity of the infrastructure and the cross-linking of the different application servers make every change a complex endeavor with the risk of many broken links. How can the IT organization shield end users and also servers from changes in the infrastructure? How can they extend functionality or processing power without breaking links or invalidating bookmarks of users?

Context A web site consisting of several web servers or web applications.

Problem You want to implement your web site using different servers or use different vendors solutions for your web site. How do you provide everything under a consistent web application space without showing your server topology to users? How do you gain flexibility in network topology, e.g. by adding or removing servers without surprising

users? How do you provide fail-over switching or load balancing, if an application server gets overloaded?

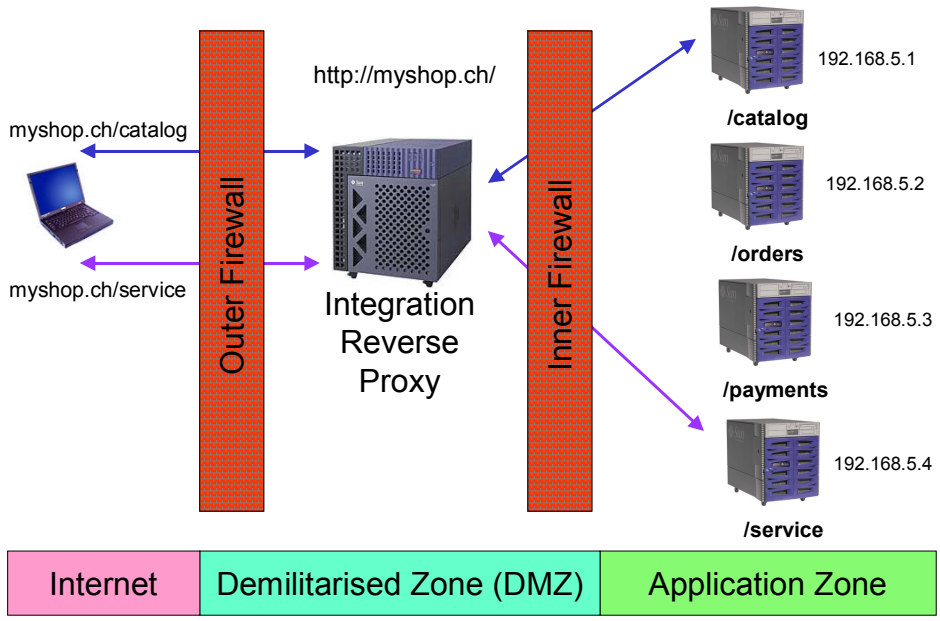
In particular you want to address the following *forces*:

- You cannot implement your complete web site with just a single server and platform, because of complexity, performance, robustness or reuse reasons.
- You want to hide network topology from your users, so that changes in machine configuration do not break their bookmarks or links to your web site.
- In addition cross (backend) server links should continue to work regardless of network topology. This ensures individual backend applications continue to work unchanged even when one backend application is moved to some other machine.
- You want to be able to exchange parts of the web site's implementation without breaking links.
- You want to easily add new elements and functionality to your web site.
- You want to be able to switch a request for an application between hosts, either for fail-over or load balancing.
- You want only a single SSL certificate, because certificates are expensive, especially coordinating their renewal.

Solution Use a reverse proxy for integrating all your web servers as backend servers with a common host address (that of the reverse proxy).

Map URL paths below the common host address to individual backend server functions, so that any modification of the association of a function to a specific backend host, can easily be changed at the reverse proxy. Optionally provide your Integration Reverse Proxy with a SSL certificate for your web site domain.

You end up with the following structure.



Implementation

The implementation of an Integration Reverse Proxy follows most of the steps explained for the Protection Reverse Proxy. Additional steps to be considered are:

1 *Design your web site's name space.* This is the step requiring some planning to allow for future extensibility. In our example, a path prefix maps to a specific server implementing the functionality. Several prefixes might also map to the same machine. Nevertheless, try to keep the mapping simple. There is one special case regarding the entry point '/': One backend server can handle this or the reverse proxy itself can show a navigation page to the user consisting of a menu of configured backend services. This can change automatically with a changed configuration of the reverse proxy.

An alternative to the path prefix mapping is to use virtual hosts for the reverse proxy, where a host name still designates a backend service. This allows our example company to continue to provide their original host names, even after they switched to a reverse proxy architecture.

A combination of prefixes and virtual hosts allows a service provider to host similar functionality for several clients without the need to duplicate all infrastructure and with easy extension of infrastructure if the need for it arises. Our example company can use this combination to provide a shop service (catalog, order, payment) for resellers under their reseller's domain address with myshop.ch's servers.

2 *Configure backend web servers*. In addition to the issues mentioned in the Protection Reverse Proxy pattern, you want links from one backend service to another. For example, the /catalog backend server will want to link to /orders and vice versa. Following your name space schema, adapt your web pages and applications to create correct links without referring to the internal host addresses.

3 *Implement backend server fail-over*. If your web site should be operational in case of hardware or software failures, or when a new version of some backend needs to be installed, you can provide a fail-over switch to a different backend server machine implementing the same functionality. Such a switch can be automatic, when the reverse proxy cannot connect to the primary backend server or manually configured by operating personnel.

4 *Implement backend server load balancing*. Similarly to fail-over you can also implement some load balancing for backends if you need it. There are several strategies possible. The simplest one is passing requests in a round robin fashion among several backend servers implementing identical functionality. More sophisticated strategies can make use of statistics collected at the reverse proxy like response times of backends or special queries to the backends collecting their respective loads.

   In the case of web applications on the backends that carry a user's session context, load balancing gets more complicated, because a session's request need to be passed to the same backend, when more than one is available (session stickiness). See the Front Door pattern for ideas on how to resolve these issues.

Known Uses Pound (http://www.apsis.ch/pound/index.html) is an integration reverse proxy providing SSL wrapping and load balancing with a simple form of session stickiness.

Variants Integration Protection Reverse Proxy. As already said, it is easy and wise to combine the Integration Reverse Proxy with the Protection Reverse Proxy and gain both benefits.

You can use Integration Reverse Proxy also for an Intranet integration scenario. Simple Intranet applications (e.g. using PHP or Perl) can be deployed quickly behind the reverse proxy, without the need to publicize the servers address explicitly to all users. In addition external web applications can be similarly integrated into the workspace without users recognizing the external nature. Combined with a menu of available backend services generated on the reverse proxy, deployment of tools can so be instantaneous. This style of integration just relying on HTTP is much easier than using a full-fledged portal server platform. In addition you can gain security, because the backend servers can operate in a separated network zone not accessible directly from the also potentially hostile corporate Intranet (e.g. consider e-mail worms).

Even a combination of two Integration Reverse Proxies, one facing the Internet and one for the Intranet sharing the backend servers is possible. This reduces cost, if the same functionality must be available on both networks.

Consequences The pattern implies the following **benefits**:

- Only one externally known host. Both only one name and one IP address for the reverse proxy need to be known and accessible outside, except when you make use of virtual hosts. You also increase security, because fewer machines need to operate in the DMZ.

- The network topology of backend servers is hidden. You can move backend web servers from one machine to another without invalidating external URLs or cross application links.

- Ease of integration and extension. Mix and match of web applications and technology becomes feasible for backend web servers and is transparent for end users.

- Bookmarks and cross-backend links continue to work, even when a backend is moved to another host.

- Load balancing of backend servers by the reverse proxy is possible. However, if your backend servers carry session, the reverse proxy must take stickiness into account. See Front Door for more optional features.

- Centralized logging. The Integration Reverse Proxy provides a good hook to implement access and error logging. Ideally backend servers no longer need to perform logging. A single log is easier to evaluate, for example, a user's navigation path can be followed easily, even if more than one backend server is used.

- You can save money and effort on SSL certificates and may be also on IP-addresses or host names, because only one host is connected to the Internet. The virtual host to service mapping is infeasible with a single SSL certificate. You then need to configure multiple IP addresses for the reverse proxy to make it possible to use valid SSL certificates or you need to use an expensive wild card SSL certificate.

  However, the Integration Reverse Proxy pattern also has its **liabilities**. It shares the last three liabilities with the Protection Reverse Proxy: Latency, some loss of transparency for backends, and it is also an additional point of failure.

- Potential single point of failure. If everything runs through your reverse proxy, this becomes a single point of failure. Additional redundancy is required for risk minimization. Without the reverse proxy, a single server outage can reduce available functionality, but might not bring everything down completely. Using a redundant hardware load balancing switch and a redundant reverse proxy configuration can alleviate that problem.

- Number of concurrent connections is limited. IP gives a hard limit to the number of usable ports and thus the number of concurrent

connections possible. On really heavy loaded sites with relatively slow backends this might imply you need additional means like multiple reverse proxies with DNS round robin to stretch these limits.

- Complexity. There can be simpler means to gain one or the other benefit. For example you can use a hardware load balancing switch.

- Session stickiness with load balancing can be problematic, when backend servers rely on sessions. See Front Door pattern for more details and resolutions of this problem.

- Testing individual applications can be harder. You may need to set up also a "dummy" Integration Reverse Proxy to be able to test new applications. There can even be the need for a complete testing environment consisting of all backends to validate all possible cross-links.

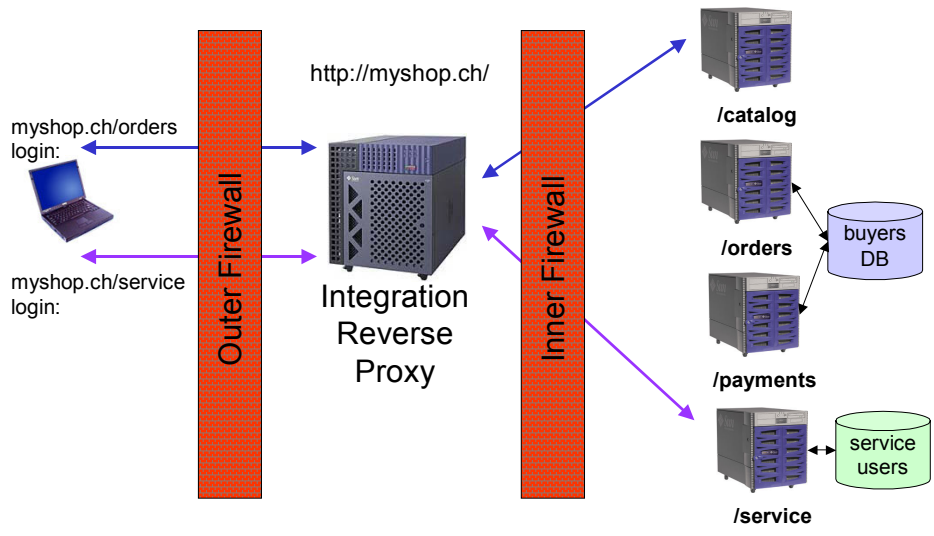
See Also The Front Door pattern for an even more sophisticated application of Integration Reverse Proxy using user authentication, authorization and session management.

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 to provide a single log-in and session context. A reverse proxy is an ideal point to implement authentication and authorization. You end up with a web entry server for your backends. A sophisticated one can even access external backends 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 our myshop.ch example. Soon after the Integration Reverse Proxy was deployed, users complained that they had to re-enter their identity several times on the web site. myshop.ch's IT personnel recognized that each web application carried its own user database. Adding an application requiring user authentication meant to add just 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 paying for the service.



How can myshop.ch easily provide access control to their web applications, without having users sign on several times and with an easy extensibility?

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

Context A web site consisting of multiple web applications that require user authentication.

An Integration Reverse Proxy where applications need to authenticate users and only authenticated users are authorized to access a defined subset of applications, a Protection Reverse Proxy where user authentication is required and only authenticated users will get access to the underlying web application, or a combination of both.

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

In particular you want to address the following *forces*:

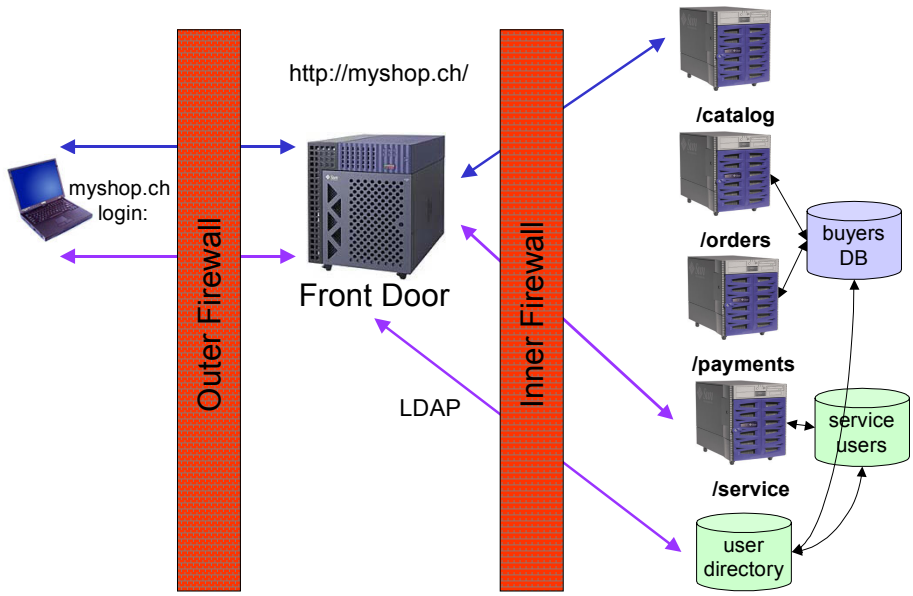
- You want a single user identity for all applications, even when existing web applications already carry their own user data base.
- You do not want users to provide their password for each application separately. This is depending on your security policy.
- You might even want to force users to identify several times to avoid misuse of a user's session left alone for some time.
- You want your applications to be independent of the authentication schema used. Depending on your policy you might even require different ones. For example, strong authentication with a one time password from a security token for payment service, or weak authentication using 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 by a single solution.

- You want new applications to easily integrate into your authentication, authorization schema.
- You want not only a single sign on but also a single log off. That means a user should keep his session as long as he is active, regardless of the concrete backend 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 backend servers without re-authentication.

Solution Implement a Front Door server as a specialization of the Security Integration Reverse Proxy that identifies users and keeps track of user sessions. It passes user identity and session identification to all of the backends. The Front Door can log all user activity in a central log. Depending on the nature of the complete solution, some backend servers might be accessible by everyone and the Front Door only protects some backend servers from unauthenticated users. Nevertheless, remember it can also act as a Protection Reverse Proxy for the public part of the web site.

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

Beyond the scope of this pattern, but often required is a system for managing user identities and access rights. In large solutions using 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.



Implementation To implement a Front Door reverse proxy in addition to the issues given in the preceding patterns the following must be considered:

1 *Unify user representation 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 means to store user identities, passwords and access rights. If you want to integrate existing backends, that you can not change, you might need to add a identity and password wallet to each user object in the directory for automatic replay of id and password when accessing such a backend.

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. Since now only the Front Door needs to implement user authentication it is easy to change or extend authentication mechanisms later without any impact on existing applications.

3 *Define access rights schema if needed.* There exist different approaches to represent access rights and the mapping of users to the set of allowed services. For the purpose of the Front Door 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 Front Door's but also for all application's needs.

4 *Design user and session representation* as passed to backends via header fields. This can be a specifically named header field or you can use HTTP basic authentication mechanism to pass on the user identity. If there is not a single user representation, Front Door might need to map the user id to the one specific to the backend. This mapping needs to be stored in the user data base as designed in step 1. Optionally define additional header fields for inter-backend communication. Those header fields are analyzed by Front Door, kept in its session store and automatically passed to all interested backends.

5 *Design and implement how Front Door keeps track of user sessions.* Some solutions that rely on SSL for browser-Front Door communication use the SSL session id for that purpose. Using a session cookie is also popular. Rewriting all URLs in the content of backend replies to add a session id, if cookies are disabled, seems to be too much overhead and too complicated. Using cookies it is even possible to be able to keep the session context when switching between HTTP and HTTPS for performance and security reasons. Front Door's session cookies should be encrypted and cryptographically signed to ensure that they cannot be manipulated. If Front Door's cookies are secured in such a way and they also contain some identification of their source, Front Door can even accept such a cookie as a valid user identification after a crash, without the user recognizing.

6 *Design and implement Front Door'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 most efficient, especially if also access rights of a user are cached there, but 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 it allows for several Front Door instances to share session context. It depends on the concrete requirements which solution for keeping session context is best. For more explanations for keeping session state, refer to the chapter 'Session State Patterns' in [Fow03].

7 *Implement a cookie jar*. If backend servers use their own session cookies Front Door can keep those session cookies in its own session context and not pass them to the user's browser. This way you can ensure single log off. Otherwise a browser might send an "old" application session cookie, after a new user logged into Front Door confusing the backend server.

8 *Design and implement login and portal page*. Front Door can delegate user identification to a special backend server or can implement its own login page. As with Integration Reverse Proxy, a "portal" page consisting 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 (e.g., /logoff) should be implemented by Front Door to allow applications to give the user the ability to consciously terminate their session.

Variants As with the Integration Reverse Proxy you can deploy two Front Doors sharing backend servers, one for the Internet (effectively making it an Extranet) and one for Intranet users.

Known Uses SYNLOGIC's Frontdoor implements most of the issues given here, in addition to being able to be configured as a Security or Integration Reverse Proxy.

Bull EVIDIAN PortalXpert implements a Web Entry Service.

IBM Tivoli Access Manager provides with its Web Seal reverse proxy functionality of a Front Door.

Consequences In addition to the consequences of Secure Reverse Proxy and Integration Reverse Proxy this pattern implies the following **benefits**:

- Single sign on and single log off, because Front Door keeps track of a user's session and backends automatically obtain the user id from Front Door instead of asking the user again.

- One user profile possible. It is not necessarily so, when you already start with a bunch of web applications and integrate them, but Front Door facilitates the mechanisms to end up with one user profile and one administration application.

- Applications are relieved from implementing access control and user authentication. This gives you the opportunity to quickly deploy web applications that readily integrate with Front Door's access control. Experience shows, that such an architectural guidance for web applications can be a great benefit, especially in an Intranet.

- Centralized logging allows user tracking and reporting. Marketing departments might die for such logs that keep exact track of user activity.

  However, in combination with the previos patterns' liabilities Front Door carries the following additional **liabilities**:

- Applications might enforce their own user database thus increasing the risk of inconsistencies. For example, RSA ACE/Server has its own user database for managing tokens for its strong authentication. If you implement Front Door 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 already exist but it might not be recognized. Deploying Front Door makes this need prominent. Also a lack in corresponding organizational processes more easily shows up.

- Password aging policies across backend applications can conflict. You then need to auto-generate new passwords when they expire, or let the user worry to change its password on the backend application and in its Frontdoor profile.

- Conflicting session time-outs of Front Door and applications can confuse users.

Credits Many thanks to my EuroPLoP 2003 shepherd Kevlin Henney and the writers' workshop participants in Irsee. The work presented is based on work of my current and former colleagues at SYNLOGIC and itopia, that have implemented our Frontdoor solutions: Andreas Birrer, Bruno Büchel, Marcel Huber, Ulf Leonhardt, Alessio Montorfano, Markus Pfister, Jürgen Wothke. Thanks to Lara Beraha, Lukas Buzzi and Felix Gähler of Telekurs Financial Information Ltd, that have let us implement Frontdoors and learned with us about the issues, benefits and drawbacks of operating reverse proxies. Thanks also to the co-editors (Markus, Frank, Ed, Duane) of the upcoming book "Security Patterns" for their support and patience.

References

[BEAH] Ben Elsinga, Aaldert Hofmann: *Security Paradigms*, Cap Gemini Ernst & Young, Nederland B.V.

[Evidian] see http://www.evidian.com for product description and white papers

[Fow03] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley 2003

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995

[POSA96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *Pattern-oriented Software Architecture–A System of Patterns*, J. Wiley & Sons, 1996

[POSA2000] D. Schmidt, F. Buschmann, H. Rohnert, M. Stal: *Pattern-oriented Software Architecture Volume 2–Patterns for Distrubuted and Concurrent Systems*, J. Wiley & Sons, 2000

[Schn03] Bruce Schneier, *Beyond Fear*, 2003

[SecPat] M. Schumacher, E. Fernandes, D. Hybertson, F. Buschmann, P. Sommerlad: *Security Patterns*, upcoming Wiley 2004

[SYNLOGIC] see http://www. synlogic.ch

[TivoliAM] IBM: *Enterprise Security Architecture using IBM Tivoli Security Solutions*, available from http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg246014.pdf