

A large, abstract, light gray graphic on the left side of the page, consisting of several overlapping, curved shapes that create a sense of depth and movement.

SUN JAVA™ WEB INFRASTRUCTURE SUITE
**USING THE JAVA SYSTEM WEB
SERVER AS A REVERSE PROXY
FOR IMPROVED SECURITY**

White Paper
July 2005

Table of Contents

Introduction	3
The Java System Web Server Reverse Proxy Plug-in	4
Server Application Functions (SAFs)	5
Installing the Reverse Proxy Plug-in	9
Configuring the Reverse Proxy Plug-in	11
Resources	15
Appendix A: Tuning the Keep-Alive Subsystem	14

Chapter 1

Introduction

More and more, the HTTP layer is being used for attacks against corporate infrastructures. Since standard security tools fail to analyze the HTTP protocol as an attack vehicle, commercial reverse proxies offer an effective method for protecting Web sites and applications.

By definition, a proxy is a device that stands between two conversing entities. To further clarify proxy functionality, a forward proxy stands between a client and all other servers (for example, outgoing Internet requests from corporate employees). A reverse proxy does exactly the opposite. It stands between a server and all its clients (such as incoming requests for a corporate Web site).

The Sun Java™ System Web Server can be configured as a reverse proxy to protect corporate Web infrastructures. While Apache, with its mod-proxy add-on, can also be used as a reverse-proxy, customers should do so only under caution. Compared to Apache, the Java System Web Server has superior security, with ten times fewer vulnerabilities as reported by CERT (www.cert.org). A secure platform is crucial for functionality sitting in the demilitarized zone (DMZ).

Employing the Java System Web Server as a reverse proxy increases Web security via:

- Single point of access, control, and logging. Since all Web traffic flows through the reverse proxy, it is easy to enforce access control rules (such as IP addresses) for all back-end Web servers. Request logging is also centralized for improved monitoring. Note that without failover, this creates a single point of failure, which should be avoided.
- Hidden network topology. Since reverse proxies map requests to content (for example, www.sun.com/webserver to webserver.sun.com), the corporate network topology is hidden from the outside world. This not only reduces the information available for attacks, but also streamlines changes to the Web infrastructure.
- Improved firewall effectiveness. With the introduction of a reverse proxy, origin Web servers can be isolated behind the corporate firewall. In addition, the reverse proxy creates a new request to the origin Web server instead of passing the initial request.

Chapter 2

Sun Java System Web Server Reverse Proxy Plug-in

The reverse proxy plug-in is a Netscape™ Server Application Programming Interface (NSAPI) plug-in designed for use with the Java System Web Server 6.1 Service Pack 3 (SP3) and later Service Packs. This add-on allows the Web Server to act as a noncaching HTTP reverse proxy for specified uniform resource identifiers (URIs).

A reverse proxy is a proxy that appears to be a Web server (origin server) to clients but in reality, forwards the requests it receives to one or more origin servers. Because a reverse proxy presents itself as an origin server, clients do not need to be configured to use a reverse proxy. By configuring a given reverse proxy to forward requests to multiple similarly configured origin servers, a reverse proxy can operate as an application-level software load balancer. In a typical deployment, one or more reverse proxies are deployed between browsers and origin servers. An example of a reverse proxy deployment is shown in Figure 1:

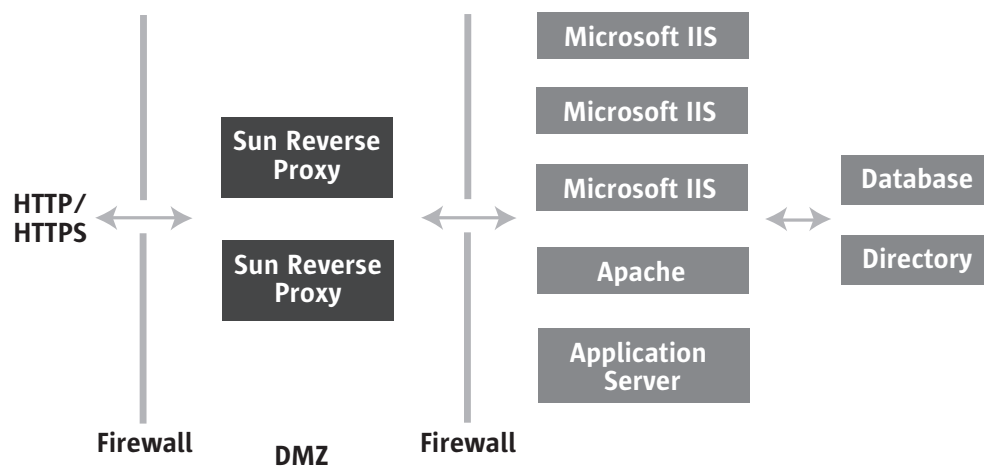


Figure 1. Secure Web Architecture

The reverse proxy plug-in can be used in conjunction with existing Java System Web Server features such as on-the-fly gzip compression, output filters, advanced Access Control Lists, and so on.

The reverse proxy plug-in includes support for the following features:

- HTTP/1.0 and HTTP/1.1 Compliance
- Credential Pass-through
- Authentication to Origin Servers
- Data Encryption
- Session Stickiness
- Simple Load Balancing
- Granular Error Logging

HTTP/1.0 and HTTP/1.1 Compliance

The reverse proxy plug-in issues HTTP/1.1 requests to origin servers, and accepts HTTP/1.0 responses to requests. It does not upgrade incoming HTTP/1.0 requests to HTTP/1.1 in ways that are incompatible with HTTP/1.0 (for example, it does not add a Transfer-encoding: chunked header to a request).

Credential Pass-through

The reverse proxy plug-in passes through Basic-Auth and Digest-Auth credentials presented by the client. It encodes client certificates from the client and presents them in proprietary headers that can be utilized by an appropriately coded application on the origin server.

Authentication to Origin Servers

The reverse proxy plug-in can be configured to present its own credentials to an origin server. The reverse proxy plug-in is capable of presenting Basic-Auth or utilizing a specified certificate nickname.

Data Encryption

The reverse proxy plug-in is able to utilize Secure Sockets Layer (SSL) v2, SSLv3, and Transport Layer Security (TLS) technology when making requests to origin servers.

Session Stickiness

The reverse proxy plug-in is able to be configured to recognize “sticky cookies,” and can configure the name of the sticky cookies.

Simple Load Balancing

The reverse proxy plug-in distributes load to several configured origin servers.

Granular Error Logging

The reverse proxy plug-in takes advantage of the Web Server’s granular error logging capabilities (config, failure, warning, fine, finer, and finest). See the Web Server’s documentation for more detail.

Server Application Functions (SAFs)

The reverse proxy plug-in provides the following SAFs:

- auth-passthrough
- check-passthrough
- service-passthrough

auth-passthrough

The auth-passthrough AuthTrans SAF inspects an incoming HTTP request for client information encoded by a service-passthrough function running on an intermediate server. The client information includes:

- The IP address from which the request originated, as encoded in the Proxy-ip header
- The SSL/TLS session ID of the originating connection, as encoded in the Proxy-ssl-id header
- The SSL/TLS cipher by the originating client, as encoded in the Proxy-cipher, Proxy-keysize, and Proxy-secret-keysize headers
- The SSL/TLS client certificate presented by the originating client, as encoded in the Proxy-issuer-dn, Proxy-user-dn, and Proxy-auth-cert headers

When auth-passthrough detects encoded client information, it instructs the server to treat the request as if it had arrived directly from the originating client, instead of via an intermediate server running service-passthrough.

The auth-passthrough SAF is optional. When used, auth-passthrough is employed on the server instance that receives the request forwarded by service-passthrough.

Since auth-passthrough makes it possible to override information that may be used for authentication (for example, the IP address of the original request), it is important that only trusted clients and servers be allowed to connect to a server running auth-passthrough. As a minimal precaution, only servers behind a corporate firewall should run auth-passthrough; no Internet-accessible server should run auth-passthrough. Further, if this information about the originating client is not required, auth-passthrough should not be used.

The following obj.conf snippet demonstrates the use of auth-passthrough (note that these lines are not indented in a real obj.conf):

```
<Object name="default">
AuthTrans fn="auth-passthrough"
...
</Object>
```

check-passthrough

The check-passthrough ObjectType SAF checks to see if the requested resource (for example, the HTML document or GIF image) is available on the local server. If the requested resource does not exist locally, check-passthrough sets the type to indicate that the request should be passed to another server for processing by service-passthrough.

The check-passthrough SAF accepts the following parameters:

- type — (Optional) The type to use for files that do not exist locally. If not specified, type defaults to magnus-internal/passthrough.

service-passthrough

The service-passthrough Service SAF forwards a request to another server for processing.

The service-passthrough SAF accepts the following parameters:

- servers — A quoted, space-delimited list of servers that receive the forwarded requests. Individual server names may optionally be prefixed with http:// or https:// to indicate the protocol, or suffixed with a colon and integer to indicate the port.
- sticky-cookie — (Optional) The name of a cookie that causes requests from a given client to “stick” to a particular server. Once a request containing a cookie with this name is forwarded to a given server, service-passthrough attempts to forward subsequent requests from that client to the same server by sending a JROUTE header back to the client. If not specified, sticky-cookie defaults to JSESSIONID.
- user — (Optional) The username that service-passthrough uses to authenticate to the remote server via Basic-Auth. Note that ‘user’ requires that ‘password’ also be specified.

- `password` — (Optional) The password that `service-passthrough` uses to authenticate to the remote server via Basic-Auth. Note that `'password'` requires that `'user'` also be specified.
- `client-cert-nickname` — (Optional) Nickname of the client certificate that `service-passthrough` uses to authenticate to the remote server.
- `validate-server-cert` — (Optional) Boolean that indicates whether `service-passthrough` should validate the certificate presented by the remote server. If not specified, `validate-server-cert` defaults to `false`.
- `rewrite-host` — (Optional) Boolean that indicates whether `service-passthrough` should rewrite the Host header sent to remote servers, replacing the local server's hostname with the remote server's hostname. If not specified, `rewrite-host` defaults to `false`.
- `rewrite-location` — (Optional) Boolean that indicates whether `service-passthrough` should rewrite the Location headers returned by a remote server, replacing the remote server's scheme and hostname with the local server's scheme and hostname. If not specified, `rewrite-location` defaults to `true`.
- `ip-header` — (Optional) Name of the header that contains the client's IP address, or `""` if the IP address should not be forwarded. If not specified, `ip-header` defaults to `Proxy-ip`.
- `cipher-header` — (Optional) Name of the header that contains the symmetric cipher used to communicate with the client (when SSL/TLS is used), or `""` if the symmetric cipher name should not be forwarded. If not specified, `cipher-header` defaults to `Proxy-cipher`.
- `keysize-header` — (Optional) Name of the header that contains the symmetric key size used to communicate with the client (when SSL/TLS is used), or `""` if the symmetric key size name should not be forwarded. If not specified, `keysize-header` defaults to `Proxy-keysize`.
- `secret-keysize-header` — (Optional) Name of the header that contains the effective symmetric key size used to communicate with the client (when SSL/TLS is used), or `""` if the effective symmetric key size name should not be forwarded. If not specified, `secret-keysize-header` defaults to `Proxy-secret-keysize`.
- `ssl-id-header` — (Optional) Name of the header that contains the client's SSL/TLS session ID (when SSL/TLS is used), or `""` if the SSL/TLS session ID should not be forwarded. If not specified, `ssl-id-header` defaults to `Proxy-ssl-id`.
- `issuer-dn-header` — (Optional) Name of the header that contains the client certificate issuer DN (when SSL/TLS is used), or `""` if the client certificate issuer DN should not be forwarded. If not specified, `issuer-dn-header` defaults to `Proxy-issuer-dn`.
- `user-dn-header` — (Optional) Name of the header that contains the client certificate user DN (when SSL/TLS is used), or `""` if the client certificate user DN should not be forwarded. If not specified, `user-dn-header` defaults to `Proxy-user-dn`.
- `auth-cert-header` — (Optional) Name of the header that contains the DER-encoded client certificate in Base64 encoding (when SSL/TLS is used), or `""` if the client certificate should not be forwarded. If not specified, `auth-cert-header` defaults to `Proxy-auth-cert`.

When multiple remote servers are configured, `service-passthrough` chooses a single remote server from the list on a request-by-request basis. If a remote server cannot be contacted or returns an invalid response, `service-passthrough` sets the status code to 502 Bad Gateway and returns `REQ_ABORTED`. This returns an error to the browser. This error can be customized in the Web Server by configuring a customized response for the 502 error code.

When `user` and `password` are specified, `service-passthrough` uses these credentials to authenticate to the remote server using HTTP basic authentication. When one or more of the servers in the `servers` parameter are configured with a `https://` prefix, `client-cert-nickname` specifies the nickname of the client certificate `service-passthrough` uses to authenticate to the remote server.

Note that service-passthrough generally uses HTTP/1.1 and persistent connections for outbound requests, with the following exceptions:

- When forwarding a request with a Range header that arrived via HTTP/1.0, service-passthrough issues an HTTP/1.0 request. This is done because the experimental Range semantics expected by Netscape HTTP/1.0 clients differ from the Range semantics defined by the HTTP/1.1 specification.
- When forwarding a request with a request body (e.g. a POST request), service-passthrough does not reuse an existing persistent connection. This is done because the remote server is free to close a persistent connection at any time, and service-passthrough does not retry requests with a request body.

In addition, service-passthrough encodes information about the originating client in the headers named by the ip-header, cipher-header, keysize-header, secret-keysize-header, ssl-id-header, issuer-dn-header, user-dn-header, and auth-cert-header parameters (removing any client-supplied headers with the same name) before forwarding the request. Applications running on the remote server may examine these headers to extract information about the originating client.

Chapter 3

Installing the Reverse Proxy Plug-in

The reverse proxy plug-in is available for use with the Java System Web Server 6.1 SP3 or later Service Packs.

This section includes the following topics:

- Package contents
- Installing on the Solaris™ Operating System, Linux, HP-UX, and AIX using tar packaging
- Installing on Microsoft Windows
- Installing on Solaris using System V, Release 4 (SVr4) Packaging
- Installing on Linux using Red Hat Package Manager (RPM) Packaging

Package Contents

The contents of the platform-specific packages are:

Solaris, Linux, AIX:

- README.txt
- libpassthrough.so (the NSAPI shared object, or “plugin”)

HP-UX:

- README.txt
- libpassthrough.sl (the NSAPI shared object, or “plugin”)

Windows:

- README.txt
- passthrough.dll (the NSAPI shared object, or “plugin”)

Installing on Solaris, Linux, HP-UX and AIX Using Tar Packaging

```
$ gzip -d sun-webserver61-passthrough- {sol|lin|hpux|aix} .tar.gz
    ;; Uncompress the tar archive,
    ;; where {sol|lin|hpux|aix} reflects
    ;; the operating system
    ;; environment the library will be used
$ tar xvf sun-webserver61-passthrough- {sol|lin|hpux|aix} .tar
    ;; Extract the tar archive
```

Please refer to “Configuring the Reverse Proxy Plugin.”

Installing on Windows

```
$ unzip sun-webserver61-passthrough-win.zip
    ;; Uncompress the ZIP archive
```

Please refer to “Configuring the Reverse Proxy Plugin.”

Installing on Solaris Using SVr4 Packaging

```
$ su
    ;; root access is required to install
    ;; SVr4 packages
$ cd <path/to/package>
    ;; Change directory to where the package
    ;; is located
# pkgadd -d .
    ;; Install SUNWwbsvr-passthrough.pkg package
```

NOTE: This places the shared object and README in: /opt/SUNWwbsvr/plugins/passthrough

Please refer to “Configuring the Reverse Proxy Plugin.”

Installing on Linux using RPM Packaging

```
$ su
    ;; root access is required to
    ;; install RPM packages
$ cd <path/to/package>
    ;; Change directory to where
    ;; the package is located
# rpm -iUvh sun-webserver-passthrough.rpm
    ;; Install
    ;; sun-webserver-passthrough.rpm
    ;; package
```

NOTE: This places the shared object and README in /opt/sun/webserver/plugins/passthrough

Please refer to “Configuring the Reverse Proxy Plugin.”

Chapter 4

Configuring the Reverse Proxy Plug-in

The reverse proxy plug-in needs to be initialized in the Java System Web Server `magnus.conf` file and configured in the corresponding `obj.conf` file.

This section includes the following topics:

- `magnus.conf`
- `obj.conf`
- Example 1
- Example 2

`magnus.conf`

`</path/to/sharedobject>` is the path where the shared object was installed, including the shared object itself.

Note that the path elements are `"/` regardless of the operating system.

```
Init fn="load-modules" shlib="</path/to/sharedobject>
```

`obj.conf`

Configuration of the `obj.conf` varies depending on the intended use. See the Java System Web Server documentation for use and syntax of the `obj.conf`.

Example 1

This configuration will proxy the URI `"/example"` if it does not exist locally. A local copy of `"/example"` is preferred to a remote copy:

```
<Object name="default">
# Assign the URI "/example" (and any more specific URIs;
# /example/foo.html, /example/qwe.jsp, etc) the object name
# "server.example.com"
NameTrans fn="assign-name"
    from="/example(|/*) "
    name="server.example.com"
...
</Object>

# Execute these instructions for any resource with the assigned name
# "server.example.com"
<Object name="server.example.com">

# Check to see if a local copy of the requested resource exists. Only
# proxy the request if there is not a local copy.
ObjectType fn="check-passthrough"
```

```

    type="magnus-internal/passthrough"

# Proxy the requested resource to the URL
# "http://server.example.com:8080" only if the "type" has been set to
# "magnus-internal-passthrough"
Service type="magnus-internal/passthrough"

    fn="service-passthrough"
    servers="http://server.example.com:8080"
</Object>

```

Example 2

This configuration will proxy all requests for the URI “/app” without first checking for a local version. The reverse proxy plug-in provide its own credentials via Basic-Auth to the origin server.

```

<Object name="default">
# Assign the URI "/app" (and any more specific URIs;
# /app/foo.html, /app/qwe.jsp, etc) the object name
# "server.example.com"
NameTrans fn="assign-name"
    from="/app(|/*)"
    name="server.example.com"
...
</Object>

# Execute these instructions for any resource with the assigned name
# "server.example.com"
<Object name="server.example.com">

# Proxy the requested resource to the URL
# "http://server.example.com:8080"
Service fn="service-passthrough"

    servers="http://server.example.com:8080"
    user="blues"
    password="j4ke&elw00d"

</Object>

```

Chapter 5

Additional Resources

Sun Java System Web Server: www.sun.com/webserver

- Downloads:
 - Web Server and Reverse Proxy Plug-in: <http://www.sun.com/download/>
- Security and reverse proxy information: http://www.sun.com/software/products/web_srvr/security.html
- Performance benchmarks: http://www.sun.com/software/products/web_srvr/benchmarks.html
- Use cases: http://www.sun.com/software/products/web_srvr/use_cases.html

Sun Fire V20z Server: www.sun.com/servers/entry/v20z/

Chapter 6

Tuning the Keep-Alive Subsystem in Java System Web Server 6.1

NOTE: If you are using Java System Web Server 6.1 SP3 or higher, there is no need to adjust the keep-alive subsystem. Other than the timeout value, Java System Web Server 6.1 SP3 automatically selects the keep-alive threading model that is most efficient for the number of connections it is handling.

For versions prior to 6.1 SP3, the information below focuses on how to tune the keep-alive subsystem. For more in-depth information on tuning other parameters, please see the Sun Java System Web Server Performance Tuning, Sizing, and Scaling Guide at <http://docs.sun.com/db/doc/817-6249-10>.

Keep-Alive Subsystem

Under certain conditions, the performance of Java System Web Server can be slower than that of earlier versions (for example, 6.0 versus 4.1). This behavior is particularly visible when the Java System Web Server serves a page with many inline images to only a few clients. The root cause can be attributed to the scalable keep-alive subsystem. When configured in the standard, out-of-the-box installation, the performance of Java System Web Server 6 can be suboptimal in low-load conditions that rely heavily on keep-alive connections.

The next section explains the functions and settings of the keep-alive performance parameters.

Performance Parameters

The performance parameters reside in the `magnus.conf` file. The following table describes their functions and attributes.

Parameter	Function	Attributes
<code>MaxKeepAliveConnections</code>	Controls the maximum number of keep-alive connections the Web Server can maintain at any time	The default is 256. The range is zero to 32,768.
<code>KeepAliveTimeout</code>	Determines the maximum time (in seconds) that the Web Server holds open an HTTP keep-alive connection or a persistent connection between the client and the server.	The default is 30 seconds. The maximum is 300 seconds (five minutes).
<code>KeepAliveThreads</code>	Determines the number of threads in the keep-alive subsystem. We recommend that you adjust this number to be a small multiple of the number of processors on your system. For example, assign two or four keep-alive threads to a two-CPU system.	The default is one.
<code>KeepAliveQueryMeanTime</code> (in Web Server 6.0, Service Pack 2 or above only)	Specifies the desired keep-alive latency in milliseconds. On lower-load systems, you can lower this number to enhance performance. Doing so can increase CPU usage, however.	The default is 100.
<code>KeepAliveQueryMaxSleepTime</code> (in Web Server 6.0, Service Pack 5 or above only)	Sets an upper limit to the time slept (in milliseconds) after polling keep-alive connections for further requests. On lightly loaded systems that primarily service keep-alive connections, you can lower this number to enhance performance. Doing so can increase CPU usage, however.	The default is 100 milliseconds. The range is zero to 5,000 milliseconds.

`KeepAliveQueryMeanTime` and `KeepAliveQueryMaxSleepTime` control latency. Changing their values affects the Java System Web Server as follows:

- By lowering their settings, you lower the latency on lightly loaded systems. An example is reduced page load times.
- By raising their settings, you raise the aggregate throughput on heavily loaded systems. An example is an increased number of requests the server can handle. However, in the case of too much latency and too few clients, the server remains unnecessarily idle and causes aggregate throughput to deteriorate.

Keep in mind the two general rules for tuning keep-alive subsystems at a particular load:

- In the event of idle CPU time, decrease the setting for `KeepAliveQueryMeanTime` and `KeepAliveQueryMaxSleepTime`.
- In the event of no idle CPU time, increase the setting for `KeepAliveQueryMeanTime` and `KeepAliveQueryMaxSleepTime`.

© 2005 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, CA 95054 USA

All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California.

Sun, Sun Microsystems, the Sun logo, Java, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a). DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS HELD TO BE LEGALLY INVALID.