webproxy

administrator's guide

FEBRUARY 2004

Legal Notices

The information contained in this document is subject to change without notice.

Warranty Disclaimer

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS INFORMATION, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Hewlett-Packard shall not be liable for errors contained herein or for direct, indirect, special, incidental or consequential damages in connection with the furnishing or use of this material.

Restricted Rights Legend

Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013. Rights for non-DOD U.S. Government Departments and Agencies are set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

Trademark Notices

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

Intel® Itanium™ Processor Family is a trademark of Intel Corporation in the U.S. and other countries and is used under license.

Acknowledgements

This product includes software developed by the Apache Software Foundation. This documentation is based on information from the Apache Software Foundation (http://www.apache.org).

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org).

This product includes cryptographic software written by Eric Young (eay@cryptsoft.com).

More information of the HP-UX Apache-based Web Server Suite can be found at http://www.hp.com/go/webserver.

Contents

Contents

Contents

Contents

1 • Webproxy Overview

This chapter provides a brief overview of the features, architecture, and limitations of Webproxy. Subsequent chapters show you how to install and configure the proxy solution.

Webproxy is a *secure reverse proxy solution* that can enhance the security of any application service delivered through a web portal. It mediates between clients on the Internet and application servers on the intranet, providing a safe passage for application data between authenticated users and authorized resources. Webproxy is based on and bundled with the HP-UX Apache-based Web Server. Webproxy functionality was previously available as the Virtualvault Web Proxy, but is now incorporated into the HP-UX Apache-based Web Server. Webproxy has the same functionality as Virtualvault Web Proxy, so previous Virtualvault Web Proxy users will continue to have the same type and level of security. For more information about Webproxy, see the HP-UX Apache-based Web Server bundled documentation.

What is a Webproxy?

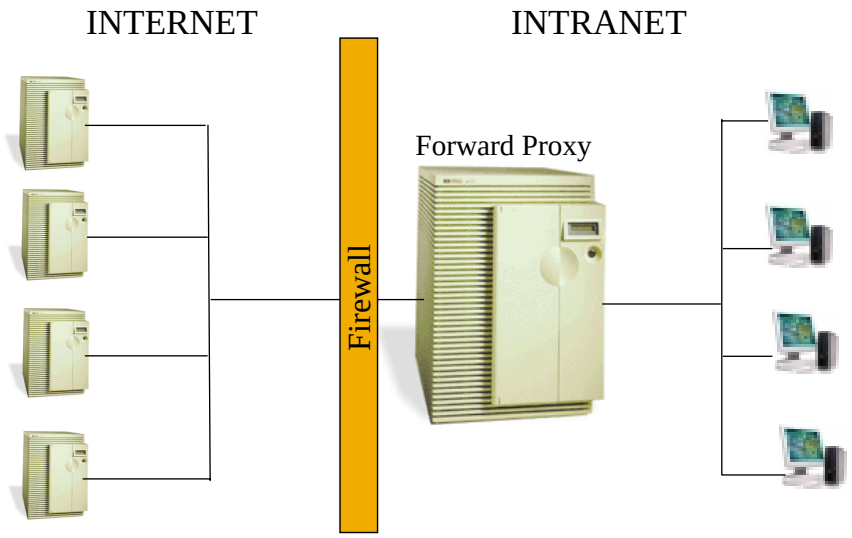
The term *proxy* originated in the legal community to indicate an entity empowered to perform actions on behalf of another. In the Internet context, a proxy is a server that acts on behalf of another. A *proxy server* is a system that resides between clients and the servers to which they want access. When the proxy server receives a request from a client, it forwards the request to a remote server, reads the server response, and then relays the response back to the client.

The term *reverse proxy* is used in so many different ways that you can easily feel lost. To help you find a clear path through the jungle of opinions, this section explains what we mean by the terms *proxy*, *forward proxy*, and *reverse proxy.*

A proxy server is more than a relay. The proxy server is a convenient site for recording user access information because all requests are channeled through this system. Moreover, the proxy server is ideally positioned to filter incoming requests or outgoing responses, providing some important security features.

Forward Proxy

A *forward proxy* server usually sits behind an enterprise's firewall, mediating access between clients on the intranet and servers on the Internet. When a client requests a resource from the Internet, the proxy server fetches the resource from the web server, caches it locally, and then forwards it to the client. If a client requests the same resource again, the proxy server returns the local cached version as long as the data has not expired. Caching eases network bottlenecks and improves the apparent response time.

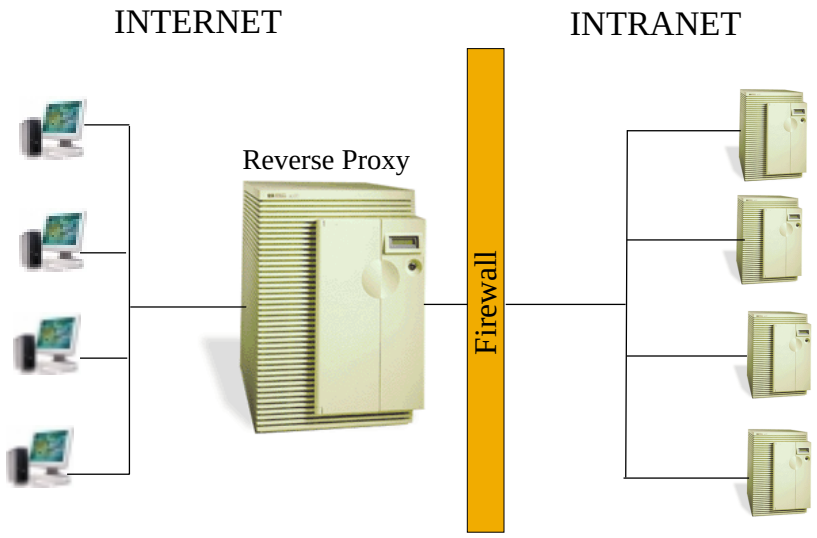


What distinguishes a proxy server deployed as a forward proxy? As a forward proxy, the proxy server is transparent to end-users. In general, the URL that users request is the URL they see in their browser's location display. Nevertheless, administrators can configure a forward proxy to filter incoming requests and send the browser to a different location.

For example, the administrator can configure the proxy server to deny access to certain web sites that are prohibited by the company's Internet usage policy. The proxy server then examines requested URLs for specified keywords and, upon a match, denies the request or serves an alternative page to the browser.

Reverse Proxy

A *reverse proxy* server mediates between clients on the Internet and servers on the intranet. When a client requests a resource, the proxy fetches it from the back-end server and relays it to the client.



What distinguishes a proxy server deployed as a reverse proxy? As a reverse proxy, the proxy server makes the back-end servers transparent to end-users. Here, *reverse proxy* refers to the deployment in which a proxy server stands-in for back-end servers.

For example, if a user requests the URL **http://www.company.com/foo.html**, the proxy intercepts the request and sends it to a back-end server, for example, **foo1.company.com**. The user's browser then displays the document location as **http://www.company.com/foo.html** even though the document is really located at **http://foo1.company.com/index.html**. Yet, a reverse proxy is more than the opposite of a forward proxy.

Reverse proxy also refers to the operation in which the proxy alters the location redirect header in server responses. For example, if the back-end server redirects the browser, the reverse proxy operation alters the location redirect header so that the browser requests **http://www.company.com/new_index.html** instead of **http://foo1.company.com/new_index.html**.

Webproxy Features

Webproxy reduces the risk of malicious attacks against your application and minimizes the damage that such attacks can cause. Combining the security of the HP-UX Apache-based Web Server and the capabilities of a reverse proxy deployment, Webproxy provides the following features.

- **User Authentication**
  Webproxy ensures that users authenticate themselves before they gain access to application resources.
- **Content Filtering**
  Webproxy can restrict access to back-end resources based on any aspect of the incoming request. Administrators can configure Webproxy to scan incoming HTTP requests and deny access from specific clients to specific back-end

servers, files, or file types. Webproxy can also route incoming requests from specific clients to alternative content instead of simply denying access.

- **Reverse Proxy**
  Webproxy hides hostnames and IP addresses of back-end servers on the intranet. Administrators can configure Webproxy so that browsers display the URL relative to the hostname of Webproxy, instead of the back-end server that actually processed the client's request.
- **Load Balancing Support**
  Webproxy can forward incoming HTTP requests to a group of servers with identical content, known as a replicated server set, so that the workload is balanced among all the servers in the replicated server set. If one back-end server becomes temporarily unavailable, Webproxy can route requests to other servers until the faulty server is repaired.
- **Portal Support**
  Webproxy can forward incoming HTTP requests to multiple applications running on disparate operating systems, such as IIS/Microsoft Platforms, HP-UX Apache-based Web Server/HP-UX, SunOne on Solaris/Solaris, each hosted on an independent, replicated server set. Webproxy maintains maximum separation between application server sets, ensuring the privacy of individual vendors even though they use a common infrastructure.
- **Self-authentication of Webproxy to Back-End Servers**
  This feature allows Webproxy to authenticate itself to back-end servers on the intranet if client authentication is required.
- **POST Method Data Filtering**
  This feature enables Webproxy to filter POST method data sent to back-end servers, based on the presence of a configured string pattern.
- **Webproxy in a Chrooted Environment**
  This feature allows the Webproxy to be run in a chrooted environment to enhance the security of the system. Webproxy server instances are chrooted by default.

Reasons for Deploying Webproxy

Webproxy can protect any web-based application that provides users on the Internet with access to resources on the intranet. Without Webproxy, applications running on the intranet are exposed to a variety of malicious attacks. An intruder may obtain unauthorized data from the back-end server or compromise the security of all hosts on the intranet. An attacker may tamper with the configuration of the proxy server, making the back-end servers susceptible to damage.

With Webproxy, the risk of such attacks is far less. Webproxy provides a blanket of security around the applications running on back-end servers, such as:

- **Privacy**
  To clients on the Internet, Webproxy appears to be a single web server because the identity of the various back-end servers is hidden from their view. As a stand-in for back-end servers, Webproxy prevents direct exposure of internal resources.
- **Containment**
  In the unlikely event of a successful attack on the web server, perpetrators have difficulty taking advantage of their break-in. They have little to gain. There are no files they could modify that would damage the application. Nor can the perpetrators gain access to the intranet because only Webproxy can communicate with the intranet hosts.
- **Logging**
  Administrators can log all application traffic at a single location because all HTTP requests and responses are routed through Webproxy. Traffic log files provide valuable insights into current and future network requirements and aid intranet expansion strategies.

How Webproxy Works

Webproxy runs on HP-UX, taking advantage of HP-UX's security features to provide a secure reverse proxy solution. When an Internet browser sends an HTTP request, the web server receives it first. After the client is authenticated, the web server sends the request through a secure interprocess communication (IPC) channel to the proxy server. The proxy server forwards the request to a back-end server on the intranet, and then returns the server response back to the browser.

Several HP-UX features and products work to make the Webproxy secure. HP-UX Apache-based Web Server instances achieve file system isolation from each other by using separate chroot compartments. Network traffic to and from the system can be limited to permitted HTTP/SSH and other secured communications via IPFilter/9000. Exploits that gain control of the system through buffer overflow attacks can be thwarted with HP-UX's built-in stack overflow protection. These characteristic not only protect applications from outside attacks, but also mitigates any damage the applications themselves could cause under the control of a malicious attacker. The configuration file that determines the proxy server's behavior is safe from tampering because it resides in one chroot compartment while the proxy server executes in another.

The web server listens for customer requests on the Internet interface only, whereas the proxy server that communicates with the back-end resources listens on *localhost* only, so the two cannot ordinarily communicate with each other on pre-established communication paths.

While Webproxy can be made reasonably secure on the HP-UX platform, Webproxy cannot protect an application against its own misconfiguration. The application running on the back-end server may contain flaws in its security design or implementation that remain unrecognized as application data passes through the Internet-boundary system. Yet, the advantage of using Webproxy is that applications need not be ported to, or execute on the Internet-boundary system where they would be at greater risk of becoming compromised.

Webproxy can be quickly integrated into any existing Internet application deployment in order to provide adequate security. You can use Webproxy to not only maintain the security of your current application, but also to gain familiarity with HP-UX security features and solutions.

HP-UX Security Overview

HP-UX, when hardened for a secure web platform such as the HP-UX Apache-based Web Server, allows you to conduct business safely on the World Wide Web. Hardened HP-UX will securely connect internal enterprise applications with clients on an external, untrusted network.

The following are a few of HP-UX's most important security features.

- Containment of programs and data files within chroot compartment
- Limited access to the system through the use of IPFilter
- Integrity tools to detect and correct configuration changes
- Application-level logging to log events with security implications
- Out-of-the-box secure configuration
- Simplified administrative interface to reduce administrative errors

Documentation

This guide is intended to provide instructions on installing and configuring Webproxy. If you are new to HP-UX security, refer to the extensive HP-UX documentation to derive the maximum benefit from its security features. If you are new to proxy servers, refer to the additional sources listed in this section.

Web Server Documentation

Webproxy is integrated into the HP-UX Apache-based Web Server, which is based of software developed by the Apache Software Foundation (**http://www.apache.org/**). The HP-UX Apache-based Web Server comes bundled with a complete set of documentation, however, the bundled documents do not cover security enhancements specific to Webproxy. Those features are covered in this document. The documentation for the HP-UX Apache-based Web Server can be found at **http://www.hp.com/go/webserver.**

Additional Documentation

We recommend that you refer to the following sources for guidance on configuring proxy servers for content filtering and load balancing.

- Apache HTTPD Server Documentation located at **http://httpd.apache.org/docs-2.0**
- Apache module mod_rewrite URL Rewriting Engine, Apache HTTP Server v. 2.x located at **http://httpd.apache.org/docs-2.0/misc/rewriteguide.html**
- Miscellaneous mod_ssl documentation located at **http://www.modssl.org/docs/**
- A Users Guide to URL Rewriting with the Apache Web Server, Ralf S. Engelschall located at **http://www.engelschall.com/pw/apache/rewriteguide**
- Apache module mod_proxy, Apache HTTP Server v. 2.x located at **http://httpd.apache.org/docs-2.0/mod/mod-proxy.html**
- Load Balancing Your Web Site: Practical Approaches for Distributing HTTP Traffic, Ralf S. Engelschall located at **http://www.webtechniques.com/archives/1998/05/engelschall**

To learn more about regular expressions (used for configuring URL Rewriting), we recommend the following:

The *regex*(3) manual page.

Mastering Regular Expressions, by Jeffrey E. F. Friedl, O'Reilly & Associates, 1997.

Technical Support

Security Updates

Since Webproxy is designed to protect your Internet application from malicious attacks, it is important that you remain current with the security issues related to your system. To that end, Hewlett-Packard provides access to the HP Security Bulletin service free of charge. Please see the following Web Server support page for information on subscribing to the electronic HP Security Bulletin, **http://www.hp.com/products1/unix/webservers/apache/support/index.html**. Updates for all security vulnerabilities found in the HP-UX Apache-based Web Server product (including Webproxy), are promptly announced in the HP Security Bulletin. At the time of announcement, a secure HP-UX Apache-based Web Server replacement version will be available for download on HP Software Depot, **http://software.hp.com.**

2 • Installing Webproxy

This chapter shows you how to install Webproxy.

Hardware Requirements

Minimum hardware requirements for running Webproxy are based on the needs of the HP-UX Apache-based Web Server with the addition of the following:

- Special needs of any application software
- Two supported network interface controllers per system

Software Requirements

Webproxy is part of HP-UX Apache-based Web Server B2.0.48.00 and later which can be installed on HP-UX 11.0 or 11i for PA-RISC and Itanium systems. The HP-UX Web Server Suite contains command-line tools and utilities to support Webproxy. Tools are available to create SSL keys, generate certificate requests, and add/remove/change private key passwords. For more information about these tools and utilities, please refer to the HP-UX Web Server Suite's bundled documentation.

Installing Webproxy

Webproxy is installed when the HP-UX Apache-based Web Server product is installed as it is part of the product. Visit HP Software Depot at **http://software.hp.com** for download and installation of the HP-UX Apache-based Web Server.

Overview of Installed Files

Webproxy, part of the HP-UX Apache-based Web Server product, installs as the "webproxy" subdirectory in the default HP-UX Apache-based Web Server directory. Below the webproxy subdirectory are several webproxy-related subdirectories.

- **/opt/hpws/apache -** This HP-UX Apache-based standard directory contains a typical Apache installation. Instead of directly executing the Apache in a single directory, Webproxy uses files in this directory as the base from which the created server instance finds its binaries, libraries, tools and documentation.

Note: The default directory for 32-bit HP-UX Apache-based Web Servers on IPF systems is **/opt/hpws/apache32**.

- **/opt/hpws/apache/webproxy/bin -** This subdirectory contains tools specific to the Webproxy installation. These tools are used to create execution and runtime environments for individual Web server instances. These tools include:
  - *mkchroot* - A tool to generate a new chroot compartment environment.
  - *rmchroot* - A tool to remove a chroot compartment environment.
  - *wp-create* - A tool to create a new server instance.
  - *wp-remove* - A tool to remove a web server instance.
  - *wp-modify* - A tool to modify an existing server instance.

  - *web_proxy_config* - A convenience tool for establishing rudimentary proxing between two servers for the purpose of connecting internet browsers to intranet hosts.

- **/opt/hpws/apache/webproxy/conf -** This subdirectory contains Webproxy configuration templates for the server(s). Each server instance, along with specific configuration, is contained in a separate subdirectory.
- **/opt/hpws/apache/webproxy/run -** This subdirectory is used by Webproxy server instances to create ssl session cache and other runtime files.
- **/opt/hpws/apache/webproxy/servers -** This subdirectory contains a subdirectory for each Webproxy server instance. The server-specific directories instance contain their own bin, conf, logs subdirectories which contain server-specific start scripts, configuration, log files and encryption certificates. The parameters make up the server instance's runtime configuration.

3 • Securing Internet Connections

When an Internet browser sends an HTTP request to your enterprise server, the web server running in the internet chroot compartment receives the request first. As the first line of defense against malicious attacks, the internet web server plays an important role in securing Internet connections. The site administrator can configure the web server to authenticate users and enable encrypted communication between the server and the clients.

Internet commerce may require both encryption and authentication services between Internet clients and webserver hosts, and encryption is supported for Webproxy servers. As a site administrator, you can configure Webproxy to support the highest level of encryption allowed by law. The following sections show you how to enable encryption on the Webproxy server.

You can have as many web servers as system resources will support on the HP-UX system, but you must have at least one internet web server and one intranet web server created to configure Webproxy. If you do not already have a web server created, refer to the following section for instructions for creating one.

Web Server Creation and Removal in Webproxy

This section provides detailed instructions for creating and deleting Webproxy servers. Two tools are provided for Webproxy server creation and removal:

wp_create - This program generates a set of configuration files and runtime directories in which a Webproxy server instance may execute. The command takes seven arguments.

Usage:

```
wp_create <serverid> <servername> <http_port> <ssl_port> <user>
<group> <network/chroot>
```

Example:

```
wp_create server1 server1 80 443 owww other internet
```

- <*serverid*> identifies the name of the file system instance of the server. The <*serverid*> string argument is appended to **/opt/hpws/apache/webproxy/servers/wp-** to form the directory name of the location where the server-specific files are to be maintained, **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>.**
- <*servername*> is a string that is used to modify the prospective server's name. <*servername*> is edited into the httpd.conf's ServerName directive, and becomes the base name that the server will assume when started.
- <*http_port*> is the default port number on which the server will listen. The <*http_port*> string argument will be edited into the prospective server's Listen directive.
- <*ssl_port*> is the default secure port on which the server will listen. The <*ssl_port*> string argument will be edited into the prospective server's Listen directive and VirtualHost declaration.
- <*user*> is a system account/username under which the server will execute. The <*user*> string argument will be edited into the prospective server's User directive.
- <group> is a system group under which the server will execute. The <group> string argument will be edited into the prospective server's Group directive.
- <network> indicates the Chroot in which the server will execute. The <network> string argument will be edited into the prospective server's Chroot directive. The reason for referring to this as <network> is the loose association between the Chroot compartment and the network that the server listens to. While there is no enforcement that a server specified as either internet or intranet actually listen to the internet or propagate requests to the intranet, if a system administrator or integrator follows the default conventions for server creation and configuration, then servers created using these defaults will operate as one might intuitively suspect they would. It is simply important to remember that the <network> argument is substituted into the Chroot directive, and that

this specified chroot compartment must exist and be populated (see the section on Chrooting the server) in order for the server to correctly chroot and execute.

wp_remove - This program removes the server instance directory at **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>**, and the cloned server module that might exist in the Webmin GUI interface.

Usage:

```
wp_remove <serverid> <servername>
```

Example:

```
wp_remove server1 server1
```

Enabling Encryption on the Webproxy Server

This section provides detailed instructions for requesting and installing server certificates, enabling encryption on the Webproxy server, and maintaining server certificate and key pair files used by the Webproxy server when encryption is enabled. The key elements to enabling encrypted communication between Internet clients and the Webproxy server are summarized as follows:

- Generating and Submitting a Certificate Request - A certificate request consists of your private key, your distinguished name (because it must be distinguishable from everyone else's), and some indisputable proof of your identity. For example, a business license. When you submit a certificate request to a CA, the CA will verify your authenticity and issue you a certificate. If you choose to request a Global Server ID, you must submit your request to Verisign Inc., currently the only CA that provides Global Server IDs.
- Installing the Certificate on Your Webproxy Server - When you install the server certificate on your Webproxy server, it allows the server to encrypt and decrypt data transmissions. If you requested a Global Server ID from Verisign

Inc., you will receive two certificates upon approval. You must install both certificates.

- Enabling encryption on the Webproxy server.
- Setting the Webproxy server encryption preferences.
- Starting the Webproxy server to apply encryption settings.
- Maintaining Certificate and Key Pair Files - The public and private keys, the server certificate, and the cryptographic modules used by the Webproxy server are stored in individual files on the secure Webproxy server. The browser and server use the encryption keys to agree upon a *session key* securely. The session key is then used to encrypt and decrypt traffic between the two entities.

Webproxy Server Certificates

A server certificate is a digital document from the certificate authority (CA) who is a trusted individual or organization that confirms the server's authenticity. In effect, the CA certifies your identity and issues you a certificate stating that the server's public key is unique and can be used to authenticate the server when it communicates with browsers.

Different CAs may use different names for the same document. In addition, some CAs may offer special server certificates for which the eligibility requirements are more stringent. The following procedures, however, apply to all cases unless otherwise noted.

In addition to maintaining authenticity, the server certificate also determines the strength of the encryption used in connections with browsers. The length of the key used to encrypt a message is a good indication of the amount of effort needed to decrypt that message.

Configuring a Webproxy server certificate involves two steps:

1. Requesting the server certificate for the Webproxy server
2. Installing the server certificate on the Webproxy server

The following sections provide step-by-step instructions for these procedures.

Generating Keys and Server Certificate

Tools to generate keys and certificates can be found in the HP-UX Apache Web Server Suite. This is available as a free download from HP Software Depot at **http://software.hp.com**. You may either use the HP-UX Web Server Suite mkcert.sh utility (which has built-in usage and help documentation), the openssl binary bundled with the suite, or the Webmin GUI to generate keys and certificates. To create a RSA private key for your Apache server (Triple-DES encrypted and PEM formatted):

```
# cd /opt/hpws/apache
# bin/openssl genrsa -des3 -rand <filename> -out \ webproxy/serv-
ers/serverid/conf/ssl.key/server.key 1024
```

<*filename*> is a large file or files containing random data used to seed the random number generator, or an EGD (Entropy Gathering Daemon) socket. Multiple files can be specified separated by a ":",(i.e. file1:file2:...:file5).

Backup this server.key file and remember the passphrase you had to enter at a secure location. You can see the details of this RSA private key via the command:

```
# bin/openssl rsa -noout -text -in \
webproxy/servers/serverid/conf/ssl.key/server.key
```

You can create a decrypted PEM version of this RSA private key using:

```
# bin/openssl rsa -in \
webproxy/servers/serverid/conf/ssl.key/server.key -out \
webproxy/servers/serverid/conf/ssl.key/server.key.unsecure
```

Create a Certificate Signing Request (CSR) with the server RSA private key (output will be PEM formatted). During this step you must answer a series of questions in order to build your distinquished name. It is very important that you answer all questions accurately.

```
# bin/openssl req -new -key \
webproxy/servers/serverid/conf/ssl.key/server.key -out \
webproxy/servers/serverid/conf/ssl.csr/server.csr
```

Answer the following questions as appropriate:

- country

- state
- locality
- organization
- organizational unit
- common name (enter the domain name or web server alias)
- email address (enter your email address)
- When prompted, enter a challenge password
- Optional company name

Make sure you enter the FQDN ("Fully Qualified Domain Name") of the server when OpenSSL prompts you for the "CommonName", i.e. when you generate a CSR for a website which will be later accessed via **https://www.foo.dom/**, enter "www.foo.dom". You can see the details of this CSR via the command:

```
# bin/openssl req -noout -text -in \
webproxy/servers/serverid/conf/ssl.csr/server.csr
```

Add "NEW" to server.csr so that:

```
-----BEGIN CERTIFICATE REQUEST-----
-----END CERTIFICATE REQUEST-----
```

becomes:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
-----END NEW CERTIFICATE REQUEST-----
```

Copy the csr file, created above, and go to a trusted CA. They will generate your CA-signed certificate.

Note:

- If you are requesting a Global Server ID from VeriSign Inc., you must copy-and-paste the certificate request into the enrollment form at VeriSign's web site.
- The approval of your certificate request may take a significant period of time, even weeks, because it involves a separate authority and requires positive proof of identity.

Installing a Server Certificate

When you receive the signed server certificate from the CA, you need to install the certificate and the private key into the Webproxy server file system hierarchy. The certificate is encrypted with your public key and can only be decrypted with your Webproxy server's private key. The server decrypts it using the private key when the server is started.

Use your browser's Edit menu to copy-and-paste the certificate and install it on the server, since incoming email is disabled on the typical HP-UX Webproxy system. Alternatively, the certificate can be installed from a file, provided the certificate file has been transferred to the HP-UX Webproxy from a floppy disk, tape, or via *ftp*.

Note: You should be logged on as **root** to access the appropriate directories for installing the certificate.

To install the server certificate and private key on the Webproxy server, follow these steps.

1. On the server that is running Webproxy, go to the **/opt/hpws/apache/webproxy/servers/ws-<*serverid*>/conf**
2. Move the private key file from to the Webproxy server instance's **/opt/hpws/apache/webproxy/servers/ws-<*serverid*>/conf/ssl.key/server.key** file.
3. When you receive your certificate from the CA, create a file named **server.crt** in theWebproxy server's **/opt/hpws/apache/webproxy/servers/ws-<*serverid*>/conf/ssl.crt** directory and put the new certificate into the **server.crt** file. To do this, use your browser's Edit menu to copy and paste the new certificate into the **server.crt** file, or *ftp* the new certificate onto the HP-UX Webproxy system from an unsecure system.
   Note:
   - If there is already a certificate file in the **/opt/hpws/apache/webproxy/servers/ws-<*serverid*>/conf/ssl.crt** directory, you must rename it or it will be overwritten. If you use a different filename other than **server.crt**, you must change the filename specified in the server's **ssl.conf** file so that they match.
   - Key and certificate files should be readable (but not writable) by the user

account specified by the server's configuration file "User" directive. This is often either iwww (intranet) or owww (internet).

4. Check the server's **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/ssl.conf** file to ensure the pathnames for the key and certificate are correct. The ssl.conf file directives should be similar to the following:

   ```
   SSLCertificateFile
   /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.cr
   t/server.crt
   SSLCertificateKeyFile
   /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.ke
   y/server.key
   ```

   Note: If you are installing a Global Server ID, you must also install the Intermediate CA Certificate that is provided with it.

After you install the certificate on the server, you can activate encryption on the Webproxy server as described in the following section.

Enabling Encryption

SSL gives Webproxy the ability to encrypt SSL traffic from the outside web servers to the inside back-end applications servers.

Additionally, Webproxy now supports rewriting to inside and back-end web servers with HTTPS to establish an SSL connection between the back-end web servers and Webproxy.

Enabling encryption on the Webproxy server is optional. You may choose to leave it turned off, as is the default. You can enable and disable it by modifying the Webproxy server's **ssl.conf** file. This file is located in the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf** directory.

Note: You must be logged on as **root** to access the appropriate directories for turning encryption on or off.

To turn encryption on for the Webproxy server, change the `SSLEngine` directive in the Webproxy server's **ssl.conf** file from `off` (default) to `on`. For example:

```
SSLEngine on
```

To turn encryption off for the Webproxy server, change the `SSLEngine` directive in the Webproxy server's **ssl.conf** file from `on` to `off`.

Setting Encryption Preferences

When a browser initiates a connection with the Webproxy server, the server indicates to the browser a number of ciphers it can use to encrypt information. The browser, in turn, will choose one of the cipher options indicated by the server. In any two-way encryption process, both parties must use the same ciphers. Since there are a number of ciphers available, your choice of cipher can be guided by the popularity of the cipher, but it should be reasoned on the basis of the key size.

If you installed a Global Server ID on the Webproxy server, then you must also consider a special case in choosing ciphers. The "step up" mechanism involved in

the 128-bit encryption supported by the Global Server ID requires that the browser and server first handshake over a 40-bit cipher before stepping up to 128-bits. If you do not select any of the 40-bit ciphers, the “step up” mechanism will fail, even if you select all the 128-bit ciphers available.

To set encryption preferences for SSL, refer to the online documentation for the mod_ssl module for the SSLCipher Suite directive located at **httpd.apache.org/docs-2.0.**

When you have finished configuring your encryption preferences for a server, you must restart the Webproxy server instance for your changes to take effect.

Restarting the Webproxy Server

For your encryption configuration changes to take effect, you must restart the Webproxy server. To restart the Webproxy server, execute the following commands.

```
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl \
stopall
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl \
startssl
```

When prompted for the passphrase, type the password you created when you requested the server certificate. If you fail to enter the correct password on the first attempt or within 60 seconds, repeat the step above.

Note: Typically, *apachectl* is executed with the "stop" argument to stop the server. When an Apache server is confined in a chroot compartment, it cannot access the pid file that it created at startup (**/opt/hpws/apache/webproxy/servers/wp_<*serverid*>/run/httpd.pid**), and so it cannot be stopped by the normal stop process. Webproxy has added the "stopall" target to kill the httpd watchdog process and its children.

SHM SSL Session Caching Support

SHM SSL is supported with this Webproxy release. The `SSLSessionCache` directive configures the storage type of the global interprocess SSL Session Cache, which speeds up parallel request processing. The SHM storage type makes use of a circular buffer method inside a shared memory segment in RAM to synchronize the local OpenSSL memory caches of the server processes.

The SSL session cache directives are in the “Inter-Process Session Cache” section of the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/ssl.conf** file. Comment out one `SSLSessionCache` statement to select the caching mechanism.

Refer to **httpd.apache.org/docs-2.0** for more information on the `SSLSessionCache` directive.

Configuring Webproxy to Authenticate to back-end Servers

To configure the Webproxy to authenticate itself to a back-end server, the following changes should be made to the Webproxy configuration file, **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/ssl.conf**. Uncomment the following line by removing the leading #:

```
#SSLVVProxyClientCertificate On
```

Maintaining Webproxy Server Certificate and Key Pair Files

When encryption is enabled on the Webproxy server, a pair of digital keys are used to cryptographically protect communication between the Webproxy system intranet web server and internal networks.

The security of the Webproxy server’s key pair must not be compromised. The private key is encrypted with a password known only by the Host administrator.

On server startup, the Host administrator provides the private key password in order to utilize the private key.

The public key is distributed to clients, which use the key to encrypt requests and replies sent to the intranet administration server. The public key is also used by the intranet Webproxy server. The server certificate is used to identify the server to both the internet Webproxy servers, as well as, the inside/intranet network servers.

The public key is not kept secret but it must be protected from tampering. The certificate authority, a trusted individual or organization, must certify the public key and create the certificate. The Host Administrator is also responsible for backing up the key pair so that they can be restored in case the files on disk are corrupted or destroyed.

After requesting and receiving the server certificate from your chosen certificate authority, the site administrator must install it on the appropriate Webproxy server instance, turn on encryption, and restart the server(s) to apply the security settings.

The site administrator has two ongoing maintenance responsibilities.

- Changing or removing key pair passwords
- Backing up server certificate and key pair files so they can be recovered

The following sections provide step-by-step instructions for these procedures.

Adding or Restoring a Key Pair Password

To put back the passphrase that has been removed from a private key or to add a passphrase for the first time, from

/opt/hpws/apache/webproxy/servers/wp_<*serverid*>/conf/ssl.key/server.key, as **root** enter the following command:

```
# /opt/hpws/apache/ssl/bin/openssl rsa -des3 -in server1.key -out \
server2.key
```

If necessary, update
/opt/hpws/apache/webproxy/servers/wp_<*serverid*>/conf/ssl.conf :

Replace:

```
SSLCertificateKeyFile /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.key/server1.key
```

With:

```
SSLCertificateKeyFile /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.key/server2.key
```

Removing a Key Pair Password

To remove the passphrase in the private key that's stored within **/opt/hpws/apache/webproxy/servers/wp_<*serverid*>/conf/ssl.key/server.key**, as **root** enter the following command:

```
# /opt/hpws/apache/ssl/bin/openssl rsa -in server.key -out \
server1.key
```

In **/opt/hpws/apache/webproxy/servers/wp_<*serverid*>/conf/ssl.conf** :

Replace:

```
SSLCertificateKeyFile /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.key/server.key
```

With:

```
SSLCertificateKeyFile /opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/ssl.key/server1.key
```

Backing Up Server Certificate and Key Pair Files

The Webproxy server certificate and key pair files should always be backed up to a removable medium in case an error corrupts or deletes the files on the system. If these files are corrupted or destroyed, you can restore them using the backup copy. Without the backup copy, you would have to generate a new certificate request, which takes its toll in time and money. You must also ensure that the backup is physically secure. Never leave backups unprotected.

The key pair files for each server certificate are installed in the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/ssl.key** and **ssl.crt** directories by default.

4 • Configuring the Proxy Server

When an internet Webproxy server forwards a client's request via trusted IPC to the intranet Webproxy server, the proxy server directs it over the intranet to a specific back-end server. When that back-end server responds, the intranet proxy server forwards the response back to the internet web server, which then sends it over the Internet to the client. Simple as this sounds, the proxy servers is far more than a mere relay. In fact, Webproxy is a powerful agent that can be configured for a wide range of functions. With such flexibility, however, comes some complexity.

The following sections are intended to help you make the Webproxy operational as quickly as possible. To customize the proxy server further, build on the examples provided here and the examples shown in the sources listed in the "Additional Documentation" section.

HTTP proxying on Webproxy requires a pair of proxy server instances. One proxy listens to the internet network interface for client connections, and forwards requests to a localhost port. The second, intranet proxy instance listens on the localhost port and forwards the request to a back-end application over the intranet network interface.

Basic Configuration

To get the Webproxy up and running, the following steps must be performed.

1. Route requests from the internet Webproxy server to the intranet Webproxy server.
2. Enable proxying.
3. Route requests from the intranet proxy server to back-end servers.
4. Hide the identity of back-end servers.

5. Start the proxy servers.

To perform these tasks, follow the procedures provided in this chapter, paying particular attention to the tips or warnings at the end of each section. If you are unfamiliar with the proxy server configuration files or configuration directives, refer to “Configuration Reference” on page 53 for a brief introduction before proceeding with the following procedures.

Routing Requests from the Web Server

To route all requests from the web server to the proxy server, follow these steps.

1. Create an intranet/internet Webproxy server pair using wp_create. Execute the web_proxy_config script by typing the following command on the command-line.
   ```
   # /opt/hpws/apache/webproxy/bin/web_proxy_config
   ```
2. When prompted:
   - enter the name of the internet web server.
   - enter the port number at which the intranet Webproxy server will listen. The port number must be greater than 1024. Port number 4441 is used as an example throughout this section.
3. Restart the internet Webproxy server.
4. Restart the intranet Webproxy server.

Tips:

- The directives, `Listen`, `ServerAdmin`, `ServerName`, `User`, and `Group` are located in “Section 2: Main Server Configuration” of the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf** file. Read the comments in the **httpd.conf** configuration file carefully and verify that these directives are valid. Failure to set up this file correctly could cause your server to not work properly or securely.
- The `Listen` directive in the proxy server configuration file ensures that the proxy server accepts requests only from the internet Webproxy server.

Enabling Proxying

Open the proxy server's configuration file **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf**. You can now enable proxying in one of the following two ways:

1. Uncomment and edit the statements related to the `ProxyRequests` directive as follows:
```
<IfModule mod_proxy.c>
ProxyRequests On
<Directory proxy:*>
    Order deny,allow
#   Deny from all
    Allow from all
</Directory>
</IfModule>
```
2. Add the `RewriteEngine` directive.
```
RewriteEngine On
```

Tip: Statements containing proxy directives are located between `<IfModule mod_proxy.c>` and the corresponding `</IfModule>`. To jump to this section of the **httpd.conf** file, search for "*mod_proxy.c*".

Note: The `Directory` directive usually contains directories while the `Location` directive contains URLs, but the proxy is controlled using <*Directory*> as a matter of legacy because <*Location*> was added later. A future version of the configuration language may switch this to <*Location*>.

Routing Requests to Back-End Servers

Now that the proxying capability is enabled, you can set up Webproxy to perform its simplest task... mirroring a back-end server. To make the proxy server function as a stand-in for the back-end server, for example, *web1*, use the `ProxyPass` directive as follows:

```
ProxyPass / http://web1.domain.com/
```

To map files and directories on the back-end server into the space of the proxy server, add statements similar to the following examples to your proxy server's configuration file.

```
ProxyPass /index.html http://web1.domain.com/top.html
ProxyPass /cgi-bin/   http://web1.domain.com/cgi-bin/
ProxyPass /images/    http://web1.domain.com/images/
ProxyPass /ssi/       http://web1.domain.com/ssi/
ProxyPass /           http://web1.domain.com/
```

Note: Ensure that the order of directives is from the most-specific to the least-specific.

To map files and directories on more than one back-end server into the space of the proxy server, add statements similar to the following examples.

```
ProxyPass /app1/   http://web1.domain.com/cgi-bin/
ProxyPass /app2/   http://web2.domain.com/cgi-bin/
ProxyPass /static/ http://10.10.10.10:8080/html/
```

Hiding the Identity of the Back-End Server

By configuring Webproxy to conceal the identity (hostname and IP address) of the back-end servers, you reduce their exposure to targeted Internet attacks. Webproxy ensures that the browser's location view displays the URL relative to the outside web server's name instead of the back-end server's name.

Webproxy ordinarily hides the identity of the back-end server when the server returns a document in response to the client's request. For example, if *web1*

responds to a client's request by providing **foo.html**, the browser's location view displays the URL **http://vault.domain.com/foo.html**.

You should also configure Webproxy to hide the identity of the back-end server when the server redirects the browser to another URL. For example, if instead of serving **foobar.html**, *web1* redirects the client to **error.html**, Webproxy can be configured to ensure that the client's browser is redirected to the URL **http://vault.domain.com/error.html**.

To mask the identity of two back-end servers, for example *web1* and *web2*, with that of the outside web server, for example *vault*, use the `ProxyPassReverse` directive as follows:

```
ProxyPassReverse /  http://web1.domain.com:8081/
ProxyPassReverse /  http://web2.domain.com:8082/
```

Note: Webproxy cannot hide back-end server identity when the HTML content provided by back-end servers contains absolute pathnames instead of links relative to the document root. For example, if *web1* redirects the browser from **foo/bar.html** to **http://web2.domain.com:8082/bad/error.html** instead of **bad/error.html**, Webproxy cannot adjust the URL displayed in the client's browser.

Configuring a Webproxy Server Instance to Filter POST Method Data

To configure the Webproxy to filter POST method data, the following changes should be made to the Webproxy configuration file, **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf**:

1. Uncomment the following line by removing the leading #:
   `#RewriteEngine On`
2. Uncomment the following line specify the log file name:
   `#RewriteLog /opt/hpws/apache/webproxy/servers/wp-<serverid>/logs/<log-file-name>`
3. Uncomment the following line and specify the log level

```
#RewriteLogLevel <number ranging from 0 to 9>
```

4. Uncomment the following line:
   ```
   #RewritePOST On
   ```
5. Uncomment the following lines to filter the string specified by "string-to-be-filtered":
   ```
   #RewriteCond%{HTTP:RewritePOST} ^.*string-to-be-filtered*
   #RewriteRule ^(.*)$ - [F]
   ```
6. Uncomment the following line to specify the back-end server:
   ```
   #RewriteRule ^(.*)$ http://<back-end.server.com>$1 [P]
   ```

The rewrite directives used in the above 6 steps should be inserted into the SSL VirtualHost section if the rewriting has to be enabled for an SSL enabled server.

Starting a Proxy Server Instance

To start the proxy server, execute the following command.

```
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl
start
```

The Webproxy server instance is now operational. To begin proxying between Internet clients and back-end application servers, it is necessary to start both the intranet proxy server and the internet proxy server instances.

Note: If you edit the configuration while the proxy server is running, you must restart the proxy server for the changes to take effect.

Advanced Configuration

Although Webproxy is now operational, it has many other features, such as content filtering, support for load balancing, and server affinity that can enhance the

security and performance of back-end servers. Webproxy consists of several modules, such as Rewrite Engine (mod_rewrite module) and Proxy (mod_proxy module) that work together to provide these features.

The connections between the mod_rewrite module and the mod_proxy module can introduce conflicts if you do not specify the directives carefully. Note that mod_proxy directives operate on requests before mod_rewrite directives. Some mod_rewrite directives may not work, for example, if mod_proxy directives specify that all requests should be proxied to back-end servers.

To be safe, comment out the following statement from your proxy server configuration file before proceeding.

```
ProxyPass  /  http://web1.domain.com/
```

Notes:

The "**\n**" characters at the end of some statements in the configuration file examples shown in the following sections indicate that the logical line continues into the next physical line. In the actual configuration file, do not reproduce the "**\n**" characters. Instead, ensure that each logical line occupies a single physical line.

The "[P]" characters at the end of some statements in the configuration file examples shown in the following sections indicate that the request should be rerouted through the proxy module.

Routing Requests from Multiple Web Servers

You can configure more than one internet Webproxy server instance to send HTTP requests to the same intranet Webproxy server instance. To do this, follow these steps.

1. Create a new web server instances for internet and intranet servers. Remember, the *web_proxy_config* script will assume that the internet server will listen to the internet network interface, and the intranet server will listen to localhost at 127.0.0.1. Listen directives placed in configuration files by *web_proxy_config* will utilize the IP address form of specification (i.e, `Listen xxx.xxx.xxx.xxx:<portnum>`).
2. Execute the *web_proxy_config* script by typing the following command on the command-line.
   ```
   # /opt/hpws/apache/webproxy/bin/web_proxy_config
   ```
3. When prompted:
   - enter the name of the new internet proxy server instance.
   - enter the port number at which the intranet proxy server will listen. Ensure that this is the same port number that you specified earlier for the first instance of the internet proxy server instance. Recall that there is no default value for the port number, but it must be greater than 1024.
4. Start the new internet proxy server instance.The script updates the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf** file for the new web server accordingly.

Routing Specific Requests from the Web Server

You can configure the internet proxy server instance to send only certain HTTP requests to an intranet proxy server instance. Requests not matching the pattern specified are then handled by other mechanisms specified by the internet server instance (i.e. error message or cgi).

To specify the requests you want to send to the proxy server, follow these steps.

1. Open a web server's **/opt/hpws/apache/webproxy/servers/ws-**<*serverid*>**/conf/httpd.conf** configuration file.
2. Find the following configuration file proxy module section.
   ```
   <IfModule mod_proxy.c>
   ProxyRequests On
   RewriteRule ^/(.*)$ http://127.0.0.1:<PORT_NUM>/$1 [p]
   ```
3. Edit the `RewriteRule ^/(.*)$` element to restrict the range of the requests to be proxied.
   For example, `"^/mycompany(.*)$"` proxies only those requests whose URLs are of the form **http://vault.domain.com/mycompany/foo.html**.
4. Restart the internet web server instance.

In most cases, Webproxy is configured to proxy all requests to the back-end servers. This special configuration may be useful unless the deployment requires the functionality of both proxying and other typical web server functions (cgis, java programs, local applications). For example, an application may have some CGI programs that run on the Webproxy system, whereas other modules of the application or its static content files are available only on back-end intranet servers. In this deployment, you may configure the web server so that a Webproxy server proxies to processes on the local system using following statement.

```
RewriteRule ^/app2(.*)$ http://127.0.0.1:<PORT_NUM_1>/$1 [P]
RewriteRule ^/html(.*)$ http://127.0.0.1:<PORT_NUM_1>/$1 [P]
```

Note that it is not recommended for CGIs to run on the HPUX Webproxy system, but the Apache server shipped with Webproxy does support CGI execution. CGI execution would take place in the chroot compartment environment, and may require additional integration beyond what one would expect in a normal Apache execution environment.

Denying Specific Requests from All Internet Clients

You can block access to specific back-end servers, files, or certain file types using the `ProxyBlock` and `ProxyBlockContent` directives. To block access to a

resource (hostname or IP Address), add statements similar to the following examples to your proxy server's configuration file.

```
ProxyBlock  beta
ProxyBlock  10.10.10.10 inside.domain.com
```

The first statement blocks requests containing the string "beta" in the hostname, such as **http://vault-beta.domain.com/**. The second statement blocks access to the back-end server with the specified IP address or hostname.

Tips: Although not essential, add these directives between the statement containing `<IfModule mod_proxy.c>` and the statement containing `</IfModule>` in order to ease maintenance.

Denying All Requests from Specific Internet Clients

You can deny requests to back-end resources from specific clients or users on the Internet using the `RewriteCond` and `RewriteRule` directives. In addition, the `RewriteMap` directive allows you to create a list of forbidden Internet hosts. This section provides some simple examples that illustrate the configuration.

Using the `RewriteCond` and `RewriteRule` directives is adequate when the web site is small, but for a web site with millions of users and thousands of files in its domain, creating a long list of users or groups, and the files that they are allowed access to, can be cumbersome and tedious.

Note: To disable the URL rewriting module, use the `RewriteEngine` directive with the value `off` instead of commenting out all the `RewriteRule` directives.

Denying Access from a Host

To deny all requests from a specific host on the Internet, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST}                    \n
^badhost\.clientdomain\.com$
```

```
RewriteRule ^/(.*)$  -  [F]
```

The first statement examines the PROXY-REMOTE-HOST header for a match on the specified hostname. The second statement blocks access to the request if the match is successful.

Denying Access from a Domain

To deny all requests from any host at a specific domain, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST} *\.baddomain\.com$
RewriteRule ^/.*  -  [F]
```

Denying Access from a Top Level Domain

To deny all requests from any domain in a specific country, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST} *\.<TLD>$
RewriteRule ^/.*  -  [F]
```

<*TLD*> in the search pattern indicates the Top Level Domain for the country.

Denying Access from Multiple Hosts

To deny all requests from a list of hosts, create a map file called **hosts.deny** as follows:

```
##  hosts.deny
10.10.10.10 -
badhost.baddomain.badTLD  -
```

Note: The **hosts.deny** file is a map of key-value pairs, even though it is treated as a list. Verify that each key (hostname/IP) is mapped to the dummy value "-".

Then, add the following statements to your proxy server's configuration file.

```
RewriteMap  hosts-deny  txt:/path/to/hosts.deny
```

```
RewriteCond                                              \n
${hosts-deny:%{HTTP:PROXY-REMOTE-HOST}|NOT-FOUND}        \n
!=NOT-FOUND [OR]
RewriteCond                                              \n
${hosts-deny:%{HTTP:PROXY-REMOTE-ADDR}|NOT-FOUND}        \n
!=NOT-FOUND
RewriteRule ^/.*  -  [F]
```

Denying Access from a User

To deny all requests from a specific user at a domain, add the following statements to your proxy server's configuration file.

```
RewriteCond                                          \n
%{HTTP:REMOTE_IDENT}@%{HTTP:PROXY-REMOTE-HOST}       \n
^baduser@clienthost\.clientdomain\.com$
RewriteRule ^/.*  - [F]
```

Denying Specific Requests from Specific Internet Clients

Commercial web sites often generate substantial revenue by selling advertising on their front pages, typically the most highly-visited pages of any site. Although an unwritten code seemingly allows anyone to link to anything on the Internet, by allowing users to bypass the front-page of a web site, “deep links” may subvert a site’s access policy.

Denying Requests from Search Agents and Robots

Search engines, robots, web crawlers, and other agents obviously perform a useful function, but they may, in some cases, allow users to bypass registration, violate copyright protections, or create an uneven performance load on some parts of the site. The standard method of blocking search engines is to use the **robots.txt** file which excludes robots, but is often ineffective or overbearing. The `RewriteCond` and `RewriteRule` directives operating on HTTP headers provide a convenient and fine-tuned alternative to the standard **robots.txt** file.

To block access to certain requests from robots, add the following statements to your proxy server’s configuration file.

```
RewriteCond %{HTTP:USER-AGENT}  ^BadRobot.*
RewriteCond %{HTTP:PROXY-REMOTE-ADDR}                  \n
^123\.45\.67\.[8-9]$ [OR]
RewriteCond %{HTTP:PROXY-REMOTE-HOST}                  \n
^search\.domain\.com$
RewriteRule ^/foo/bar/deep/.+   -   [F]
```

Denying Requests through Deep Links

Like search engines and robots, deep links allow a user to bypass the “front page” of a site and may take away its value. If someone were to develop their web page with in-lined graphics whose source is located on your web site, the link from their page would add traffic to your site without adding any revenue.

In some cases of severe trespassing, corporations have pursued legal redress against deep link providers, but Webproxy offers a simple technical solution. Using the `RewriteCond` and `RewriteRule` directives, you can block access through deep links whenever the browser sends out the HTTP_REFERER header.

To block access to certain requests through deep links, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:REFERER} !^$
RewriteCond %{HTTP:REFERER} !^http://www\.domain\.com/.*$
RewriteRule ^/foo/bar/deep/.+      -    [F]

RewriteCond %{HTTP:REFERER}   !^$
RewriteCond %{HTTP:REFERER}   !.*/foo-with-gif\.html$
RewriteRule ^inlined-in-foo\.gif$  -   [F]
```

Routing Specific Requests from Specific Internet Clients

Although much of the preceding discussion focused on denying access, security is not just about keeping people out. Good security is about letting the right people in so they can access the resources to which they are entitled to quickly, safely, and privately.

The mod_rewrite and the mod_proxy modules work together to provide a powerful mechanism for routing HTTP requests to the back-end server and redirecting requests based on HTTP headers. The following subsections illustrate only some of the many ways in which the capabilities of these two modules can be exercised. For additional resources, refer to "Additional Documentation" on page 9.

Routing Requests to Back-End Servers

Recall that the `ProxyPass` directive allows you to route incoming requests to a back-end server in the following statements.

```
ProxyPass /index.html http://web1.domain.com/top.html
ProxyPass /cgi-bin/   http://web1.domain.com/cgi-bin/
ProxyPass /images/    http://web1.domain.com/images/
```

```
ProxyPass /ssi/       http://web1.domain.com/ssi/
ProxyPass /           http://web1.domain.com/
```

The `RewriteRule` directive provides a simple alternative to the above statements as shown in the following statements.

```
RewriteRule                                          \n
^/index.html         http://web1.domain.com/top.html  [P]
RewriteRule ^/(.*)$        http://web1.domain.com/$1  [P]
```

The `ProxyPass` directive also allows you to route incoming requests from the proxy server to more than one back-end server as shown in the following statements.

```
ProxyPass /app1/   http://web1.domain.com/cgi-bin/
ProxyPass /app2/   http://web2.domain.com/cgi-bin/
ProxyPass /static/ http://10.10.10.10:8080/html/
```

To handle this using the `RewriteRule` directive, add the following statements to your proxy server's configuration file.

```
RewriteRule ^/app([1-9])/(.*)$ http://web$1.domain.com/cgi-bin/$2
[P]
RewriteRule ^/static(.*)$ http://10.10.10.10:8080/html$1 [P]
```

The first statement takes advantage of the ordered pairs (app1,web1) and (app2,web2) to provide a generalized rule that handles both cases.

Note: The order of the `RewriteRule` directives is the order in which the rules are applied at run-time. Directives should be added typically in the most-specific to the least-specific order. But when directives are equally specific, add them in the most-requested to the least-requested order.

Routing Requests to Client-Specific Content

Content providers often create multiple versions of the same file to account for differences in browser display. Using Webproxy, you can provide one version of a document to Netscape Navigator, another to Internet Explorer, and a third version to all other browsers.

```
RewriteCond %{HTTP:USER-AGENT}  ^Mozilla/.*
RewriteRule ^foo\.html$         foo\.NS\.html     [P,L]
```

```
RewriteCond %{HTTP:USER-AGENT}  ^MSIE/.*
RewriteRule ^foo\.html$         foo\.IE\.html     [P,L]
RewriteRule ^foo\.html$         foo\.x\.html      [P,L]
```

In the first statement, the `RewriteCond` directive evaluates HTTP_USER_AGENT against the regular expression `^Mozilla/.*`. If this condition is satisfied, the `RewriteRule` directive rewrites **foo.html** to **foo.NS.html**. The P (proxy) flag notifies the mod_rewrite module to forward the request to the mod_proxy module. The L (last) flag indicates that no further rules operate on this request.

If the request does not match the first condition, the rewrite engine checks for the condition specified in the second `RewriteCond` statement. Upon a match, the `RewriteRule` directive rewrites **foo.html** to **foo.ie.html**. All other browsers are served the generic **foo.x.html** file.

If the browser making the request is in fact a search agent, as illustrated in “Denying Requests from Search Agents and Robots” on page 41, you can force the server to respond with the “front page” by adding the following statements to your proxy server’s configuration file.

```
RewriteCond %{HTTP:USER-AGENT}  ^BadRobot.*
RewriteCond %{HTTP:PROXY-REMOTE-ADDR}                    \n
^123\.45\.67\.[8-9]$ [OR]
RewriteCond %{HTTP:PROXY-REMOTE-HOST}                    \n
^search\.domain\.com$
RewriteRule ^/foo/bar/deep/.+   index\.html$        [P,L]
```

Unlike search engines and robots, deep links that bypass the “front page” of a site cannot be identified by the client that originated the request. Yet, the browser often sends out the HTTP_REFERER header which identifies the referring site. If the referring site is not a part of your own web site’s domain, you can redirect the incoming request to the “front page” of your site, effectively thwarting the deep link.

To redirect requests through deep links to the front page, add the following statements to your proxy server’s configuration file.

```
RewriteCond %{HTTP:REFERER} !^$
RewriteCond %{HTTP:REFERER} !^http://www\.domain\.com/.*$
```

```
RewriteRule ^/foo/bar/deep/.+ http://www1.domain.com/index.html
[P]
```

Redirecting All Requests from Specific Internet Clients

You can redirect requests to back-end resources from specific clients or users on the Internet using the `RewriteCond` and `RewriteRule` directives. The `RewriteMap` directive allows you to also create a list of Internet hosts that are always redirected. This section provides some simple examples that illustrate the configuration.

Redirecting Access from a Host

To redirect all requests from a specific host on the Internet, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST}                  \n
^badhost\.clientdomain\.com$
RewriteRule ^/.*$  http://www1.domain.com/error.html [P]
```

The first statement examines the REMOTE_HOST header for a match on the specified hostname. If the match is successful, the second statement rewrites the URL to a custom error page.

Redirecting Access from a Domain

To redirect all requests from any host at a specific domain, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST} *\.baddomain\.com$
RewriteRule ^/.*$  http://www1.domain.com/error.html [P]
```

Redirecting Access from a Top Level Domain

To redirect all requests from any domain in a specific country, add the following statements to your proxy server's configuration file.

```
RewriteCond %{HTTP:PROXY-REMOTE-HOST} *\.<TLD>$
RewriteRule ^/.*$  http://www1.domain.com/error.html [P]
```

<*TLD*> in the search pattern indicates the Top Level Domain for the country.

Redirecting Access from Multiple Hosts

To redirect all requests from a list of hosts, create a map file called **hosts.deny** as follows:

```
##  hosts.redirect
10.10.10.10 -
badhost.baddomain.badTLD  -
```

Note: The **hosts.redirect** file is a map of key-value pairs, even though it is treated as a list. Verify that each key (hostname/IP) is mapped to the dummy value "-". Then, add the following statements to the configuration file:

```
RewriteMap  hosts-deny  txt:/path/to/hosts.redirect
RewriteCond ${hosts-redirect:%{HTTP:PROXY-REMOTE-HOST}|NOT-FOUND}
!=NOT-FOUND [OR]
RewriteCond ${hosts-redirect:%{HTTP:PROXY-REMOTE-ADDR}|NOT-FOUND}
!=NOT-FOUND
RewriteRule ^/.*$  http://www1.domain.com/error.html [P]
```

Redirecting Access from a User

To redirect all requests from a specific user at a domain, add the following statements to your proxy server's configuration file.

```
RewriteCond                                              \n
%{HTTP:REMOTE-IDENT}@%{HTTP:PROXY-REMOTE-HOST}           \n
^baduser@clienthost\.clientdomain\.com$
RewriteRule ^/.*$  http://www1.domain.com/error.html [P]
```

Balancing Load in a Replicated Server Set

Webproxy can forward incoming HTTP requests to a set of back-end servers, each of which has identical static and dynamic content. Moreover, you can configure Webproxy to balance the load among the back-end servers in the replicated server set. Although we do not cover that configuration here, you can configure Webproxy to remove a server when it becomes temporarily unavailable from the server set until it comes back online.

Suppose we want to balance the traffic to **www.foo.com** across **www[0-5].foo.com** (a total of 6 servers). This section provides instructions on how this can be done.

DNS Round-Robin

The simplest method for load balancing is to use the DNS round-robin feature of BIND. First, configure **www[0-5].foo.com** in your DNS with A (address) records:

```
www0   IN  A       1.2.3.1
www1   IN  A       1.2.3.2
www2   IN  A       1.2.3.3
www3   IN  A       1.2.3.4
www4   IN  A       1.2.3.5
www5   IN  A       1.2.3.6
```

Then add the following statements:

```
www    IN  CNAME   www0.foo.com.
       IN  CNAME   www1.foo.com.
       IN  CNAME   www2.foo.com.
       IN  CNAME   www3.foo.com.
       IN  CNAME   www4.foo.com.
       IN  CNAME   www5.foo.com.
```

When **www.foo.com** gets resolved, BIND gives **www0** to **www5**, in a slightly rotated order each time.

Proxy Round-Robin

Using the capabilities of the mod_rewrite module, you can configure Webproxy to use the proxy server for load balancing by adding the following statements to the **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf** file.

```
RewriteEngine on
RewriteMap lb   prg:/opt/hpws/bin/lb.pl
RewriteRule     ^/(.+)$  ${lb:$1}       [P,L]
```

This ruleset establishes a load balancing script **lb.pl** for all URLs.

The **lb.pl** script can then be written as follows:

```
#!/path/to/perl
##
##  lb.pl -- load balancing script
##

$| = 1;

$name   = "www";     # the hostname base
$first  = 0;         # the first server
$last   = 5;         # the last server in the round-robin
$domain = "foo.com"; # the domainname

$cnt = 0;
while (<STDIN>) {
       $cnt = (($cnt+1) % ($last+1-$first));
       $server = sprintf("%s%d.%s", $name, $cnt+$first, $domain);
       print "http://$server/$_";
}
##EOF##
```

Maintaining Server Affinity in a Replicated Server Set

When Webproxy forwards incoming HTTP requests to a set of back-end servers, each of which has identical static and dynamic content, you can configure the proxy server so that a client maintains affinity with a particular server in the replicated server set.

As in the previous round-robin example, consider a deployment where Internet clients send requests to **http://www.foo.com** and Webproxy forwards them to one of six servers (**www[0-5].foo.com**). In some cases, the back-end application may maintain its state while processing the request, which requires the client to continue sending requests to the same server over the course of the transaction. If the client were to connect to another server in the replicated server set, its request would be treated as a new connection and the transaction may be aborted.

Webproxy provides a convenient way to maintain server affinity through the use of cookies. You can configure Webproxy to examine the HTTP request headers for a cookie header containing the **HPWP=http://www1.foo.com** name-value pair. If the cookie header does not contain this name-value pair, Webproxy writes a cookie into the server's response with the name "HPWP" and value of the back-end server.

If the cookie header contains the name-value pair **HPWP=http://www1.foo.com**, a cookie is not written unless the back-end server explicitly sets a new cookie. If the back-end server sets new cookies, Webproxy resets its own cookie to prevent the original from being overwritten. You can then configure Webproxy to redirect subsequent requests from the same client (whose request headers contain the Webproxy cookie) to the back-end server specified in the cookie.

To maintain server affinity in a replicated server set, follow these steps.

1. Enable server affinity by adding the following statements to your proxy server's configuration file.

```
ProxyCookie         On
ProxyCookieExpires  300
```

2. Configure Webproxy to examine incoming HTTP requests for the cookie header and redirect the request to the appropriate back-end server by adding the following statements to your proxy server's configuration file.

```
RewriteCond  %{HTTP:COOKIE}   HPWP=http://www0.foo.com/
RewriteRule  ^/(.+)$          http://www0.foo.com/$1  [P]

RewriteCond  %{HTTP:COOKIE}   HPWP=http://www1.foo.com/
RewriteRule  ^/(.+)$          http://www1.foo.com/$1  [P]

RewriteCond  %{HTTP:COOKIE}   HPWP=http://www2.foo.com/
RewriteRule  ^/(.+)$          http://www2.foo.com/$1  [P]

RewriteCond  %{HTTP:COOKIE}   HPWP=http://www3.foo.com/
RewriteRule  ^/(.+)$          http://www3.foo.com/$1  [P]

RewriteCond  %{HTTP:COOKIE}   HPWP=http://www4.foo.com/
RewriteRule  ^/(.+)$          http://www4.foo.com/$1  [P]

RewriteCond  %{HTTP:COOKIE}   HPWP=http://www5.foo.com/
RewriteRule  ^/(.+)$          http://www5.foo.com/$1  [P]
```

Notes:

- The `ProxyCookie` directive allows Webproxy to send cookies to the client. To disable server affinity, set the `ProxyCookie` directive to `off`.
- The `ProxyCookieExpires` directive determines the duration for which the cookie is active. The default value of `ProxyCookieExpires` is 5 minutes (300 seconds) and its range is [0-999999999] seconds.
- Ensure that the system date and time are set correctly before specifying the `ProxyCookieExpires` value.
- Test the `ProxyCookieExpires` value with the browsers commonly used by your customers. Some browsers may not expire the cookie correctly.
- HTTPS can also be used where HTTP is shown in the above examples if you have SSL enabled.

Restarting the Proxy Server

For your configuration changes to take effect, you must restart the proxy server. To restart the proxy server, execute the following commands.

```
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl \
stopall
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl \
start
```

To restart the proxy server with SSL, execute the following command.

```
# /opt/hpws/apache/webproxy/servers/wp-<serverid>/bin/apachectl \
startssl
```

The examples in this chapter are intended to give you a hint of the power and flexibility of Webproxy. With a little extra effort, you can build on them to customize a solution that precisely meets your needs.

Configuring Webproxy to Run in a Chrooted Environment

By default, each Webproxy server runs in a chrooted environment. When HP-UX Apache-based Web Server is installed, the Webproxy component is installed with two default roots: **/var/jail/wp_internet** and **/var/jail/wp_intranet**. Other Chroot environments can be created with the **/opt/hpws/apache/webproxy/bin/mkchroot** script:

```
# /opt/hpws/apache/webproxy/bin/mkchroot  <rootname>
```

When this mkchroot command executes, it creates a server environment at **/var/jail/wp_<*rootname*>**. This root environment contains executables, libraries, devices and a base document directory for executing an apache server. The server

executes in this environment because the HP-UX Apache-based Web Server has been modified to utilize a Chroot directive in its configuration file,

/opt/hpws/apache/webproxy/servers/wp_<serverid>/conf/httpd.conf :

```
# Chroot /var/jail/wp_<rootname>
```

This directive is placed in the httpd.conf file during server creation. The root name is derived from the <*network*> parameter of the **/opt/hpws/apache/webproxy/bin/wp_create** script:

```
# /opt/hpws/apache/webproxy/bin/wp_create server1 server1 80 443
owww www internet
```

In the above example, */var/jail/wp_* is prepended to "internet" to form the Chroot variable directory, **/var/jail/wp_internet**. Since no check is made to determine if the chroot compartment exists, it is important to make sure that the root name is spelled correctly, and that the root exists. Otherwise, the server will fail to start.

This base root environment is designed to proxy only. If it is desired that the server should execute cgi programs or server custom static content, then the application integrator will need to modify the chroot environment to make it suitable for the desired purpose.

Chroot environments can be removed using the **/opt/hpws/apache/webproxy/bin/rmchroot** command.

```
# /opt/hpws/apache/webproxy/bin/rmchroot <rootname>
```

This command will remove the directory **/var/jail/wp_<*rootname*>** and all of its contents. If *rmchroot* is executed without a rootname, it removes all chroot environments (all environments that begin with **/var/jail/wp_**), so be careful when executing this command. The *rmchroot* command is executed to delete all chroot environments during Webproxy product removal.

5 • Configuration Reference

This chapter provides reference material that you can use to gain a deeper understanding of Webproxy configuration tasks. Following a brief introduction to regular expressions, this chapter describes the proxy server configuration files and the usage of some common configuration directives. This chapter also includes the list of HTTP headers upon which the RewriteCond directive operates, and the list of flags available with the RewriteRule directive.

Regular Expressions

Understanding regular expressions is a prerequisite to configuring the proxy server. Using regular expressions, you can define patterns on which the conditions and rules of the URL rewriting engine operate. The following table lists some commonly used expressions.

Expression	Meaning
.	Matches any single character except a newline
x?	Matches zero or one occurrences of regular expression x
x*	Matches zero or more occurrences of regular expression x
x+	Matches one or more occurrences of regular expression x
x{n,m}	Matches the character x where x occurs at least *n* times but no more than *m* times
x{n,}	Matches the character x where x occurs at least *n* times
x{n}	Matches the character x where x occurs exactly *n* times
[abc]	Matches any of the characters enclosed in the brackets
[^abc]	Matches any character not enclosed in the brackets

Expression	Meaning
[a-z]	Matches any characters within the range in the brackets
x	Matches the character x where x is not a special character
\x	Removes the meaning of special character x
"x"	Removes the meaning of special character x
xy	Matches the occurrence of regular expression x followed by the occurrence of regular expression y
x|y	Matches either the regular expression x or the regular expression y
^	Matches the beginning of a string
$	Matches the end of a string
(x)	Groups regular expressions

The more you know about regular expressions, the more there seems to be left to know. We recommend that you read the *regex*(3) manual page or acquire the book *Mastering Regular Expressions*, by Jeffrey E.F. Friedl, O'Reilly & Associates, 1997, to learn more.

Configuration Files

Configuring the Webproxy server involves modifying various directives in the central configuration file, **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/httpd.conf**. Most SSL Virtualhosts and SSL configuration directives can be found and modified in **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf/ssl.conf**. These two files control most aspects of the Webproxy server's configuration and operation.

Two configuration files are now distributed empty, because it is recommended that all directives be kept in a single file for simplicity. These two files, included primarily for historical purposes, are **srm.conf** and **access.conf**.

The configuration directives in the **httpd.conf** file are grouped into three main sections. These sections are:

1. Directives that control the operation of the server process as a whole.
2. Directives that define the parameters of the main or default server, which responds to requests that are not handled by a virtual host. These directives also provide default values for the settings of all virtual hosts.
3. Settings for virtual hosts, which allow web requests to be sent to different IP addresses or hostnames and have them handled by the same server process.

Notes:

- Unless otherwise noted, maintain all directives in the **httpd.conf** and **ssl.conf** files. In configuring the Webproxy server, you will be modifying a few directives (port number and server servername) in the first section, many directives in the second section, and perhaps some in the third section. Most SSL configuration directives should be specified in **ssl.conf.**
- The **/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/conf** directory also contains another file called **mime.types**. This file usually does not need editing.
- The **httpd.conf** file is processed before the **srm.conf** and **access.conf** files. If you find it necessary to use these files, you can override the processing of these later files using the `ResourceConfig` and `AccessConfig` directives in the **httpd.conf** file. The **ssl.conf** file is #included from the **httpd.conf** file.

- Read the comments in each file carefully. Failure to setup these files correctly could cause your server to not work properly or securely.

Configuration Directives

This section provides a brief description of directives commonly used in configuring Webproxy. Although this information is intended to be complete, the following sections do not provide comprehensive documentation of the mod_rewrite and mod_proxy modules. For more details about these and other modules, refer to “Additional Documentation” on page 9.

mod_proxy

The mod_proxy module provides the forward proxying, reverse proxying, and caching capabilities of the Webproxy server. This module works in conjunction with the mod_rewrite module to forward incoming or rewritten HTTP/1.0 requests to back-end servers. This module also adjusts the location headers of outgoing responses to hide back-end server identity. The following directives allow you to configure these capabilities.

ProxyRequests

```
Syntax: ProxyRequests <On|Off>
Default: ProxyRequests Off
Context: server config, virtual host
```

This directive enables or disables proxying. Setting ProxyRequests to `Off` does not prevent use of the ProxyPass directive.

ProxyPass

```
Syntax: ProxyPass <path> <URL>
Default: None
Context: server config, virtual host
```

In the syntax, <*path*> refers to the name of a local virtual path and <*URL*> refers to a partial URL for the remote server.

This directive allows remote servers to be mapped into the space of the local server so that the local server appears to be a mirror of the remote server.

ProxyPassReverse

```
Syntax: ProxyPassReverse <path> <URL>
Default: None
Context: server config, virtual host
```

This directive enables Webproxy to adjust the URL in the location header of HTTP redirect responses.

Notes:

- `ProxyPassReverse` does not depend on a corresponding `ProxyPass` directive; it can be used in conjunction with the `RewriteRule` directive provided by the mod_rewrite module.
- Without the `ProxyPassReverse` directive, HTTP redirect responses from back-end servers would bypass the proxy server. Under such circumstances, the client's browser would display the identity of the back-end server, diminishing the security offered by Webproxy.

ProxyBlock

```
Syntax: ProxyBlock <word host domain_list>
Default: None
Context: server config, virtual host
```

This directive specifies a list of words, hosts, or domains, separated by spaces. The Webproxy server blocks requests whose URLs match words, hosts, or domains in the list.

mod_rewrite

The following directives are included with the mod_rewrite module.

RewriteEngine

```
Syntax: RewriteEngine <on|off>
Default: RewriteEngine off
Context: server config, virtual host, directory, .htaccess
```

This directive enables or disables URL rewriting at runtime.

Note: By default, rewrite configurations are not inherited. If you have configured multiple virtual hosts (proxy servers), you must include a `RewriteEngine on` directive for each virtual host.

RewriteLog

```
Syntax: RewriteLog <filename>
Default: None
Context: server config, virtual host
```

This directive sets the name of the file to which the server records its rewriting actions. If the name does not begin with a slash ('/') then it is assumed to be relative to the Server Root.

Note: This directive should occur only once in the server configuration.

RewriteLogLevel

```
Syntax: RewriteLogLevel <level>
Default: RewriteLogLevel 0
Context: server config, virtual host
```

This directive determines the verbosity of the rewriting logs. The default level 0 means no logging, while 9 or more means that practically all actions are logged.

Notes:

- To disable the logging of rewriting actions, set <*level*> to 0. This disables all rewrite action logs.
- Using a high value for Level will slow down your server dramatically. Use the rewriting logfile at a Level greater than 2 only for debugging.

RewriteMap

```
Syntax: RewriteMap <MapName> <MapType:MapFile>
Default: not used per default
Context: server config, virtual host
```

This directive defines a map containing key-value pairs that the Webproxy server can look up to perform URL rewriting (insertion or substitution) operations.

<*MapName*> specifies a mapping-function in the form of one of the following constructs:

```
${ MapName : LookupKey}
${ MapName : LookupKey| DefaultValue }
```

When such a construct appears in a rewrite rule, the Webproxy server consults the `MapName` function with the `LookupKey` key. If the lookup succeeds, the function inserts or substitutes the key with the associated value. If the lookup fails, the function inserts the default value or, if `DefaultValue` was not specified, the function inserts an empty string.

<*MapType*> specifies the type of map file to be used by the mapping-function. <*Mapfile*> provides the path to the map file. The `MapType` can be one of the following codes.

MapType	Description
txt	This type indicates that the mapfile is a plain ASCII file containing either blank lines, comment lines (starting with a '#' character) or key-value pairs.
rnd	This element would be identical to **txt** except for a special post-processing feature. This mapfile can contain multiple values associated with a key, where each value is separated by the '|' character which means “or”'. The actual returned value is chosen randomly. This type of mapfile allows you to configure load balancing among the servers named by each value.
dbm	The mapfile is a binary NDBM format file containing the same contents as a plain text file, but in a special representation optimized for fast lookups. You can create such a file using any NDBM tool.
int	The mapfile is one of the following functions internal to the server: **toupper -** Converts the looked up key to all upper case **tolower -** Converts the looked up key to all lower case **escape -** Translates special characters in the key into hex-encodings **unescape -** Translates hex-encodings in the key back to special characters
prg	The mapfile is an executable program written in any language (either object-code or a script with the path to the interpreter in its first line). The program starts with the server and communicates with the rewrite engine over its stdin and stdout file-handles. For each map-function lookup, it receives the key as a string through stdin. If the match is successful, it returns the value as a string through stdout. If the match fails (there is no corresponding value for the given key), it returns the four-character string “NULL”.

RewriteCond

```
Syntax: RewriteCond <TestString> <CondPattern>
Default: None
Context: server config, virtual host, directory, .htaccess
```

This directive defines a condition, one or more, which can precede a `RewriteRule` directive. The rule is enforced only if all the preceding conditions specified by `RewriteCond` apply and the rule pattern specified by `RewriteRule` matches the current state of the URL.

<*`TestString`*> contains either a plain text or an expanded construct that the Webproxy server examines for a pattern match. <*`CondPattern`*> specifies the pattern against which <*`TestStrin`*g> is to be matched.

For a list of HTTP headers and server variables that you can use to construct an expanded pattern, refer to “HTTP Headers” on page 63.

RewriteRule

```
Syntax: RewriteRule <Pattern> <Substitution>
Default: None
Context: server config, virtual host, directory, .htaccess
```

The `RewriteRule` directive is the real rewriting workhorse. This directive can occur more than once, where each directive defines a single rewriting rule. The order of these rules specifies the order in which the rules apply at run-time.

<*`Pattern`*> is a regular expression that is applied to the current URL, where “current" refers to the value of the URL when this rule is applied. The current URL may not be the originally requested URL, because any number of rules may already have matched and made alterations to it.

<*`Substitution`*> is the string that replaces the original URL upon a pattern match. In addition to plain text, the substitution string may contain the following expanded constructs.

- $N back-references to the `RewriteRule` pattern
- %N back-references to the last matched `RewriteCond` pattern
- Server-variables in rule condition test-strings (%{VARNAME})
- Map function calls (${mapname:key|default})

Back-references, $N (N=[0-9]) and %N (N=[1-9]), identify the N^{th} group of the matched pattern. Variables identify the user, the server, the client, or the connection. Map functions are defined by the `RewriteMap` directive.

After the rewriting rules are applied to the pattern (in the order of definition in the configuration file), the URL is completely replaced by the substitution.

A special substitution string '-' indicates no substitution and is often useful in developing rewrite rules that forbid requests matching the pattern or, in conjunction with the C (chain) flag, match more than one pattern before applying the substitution.

RewriteRule Flags

The following table lists the flags that can qualify rewrite rule operations.

Flag	Description
redirect | R [=status-code]	Sends a status code between 300 and 400, specified by status-code. The default is 302.
forbidden | F	Forces the proxy server to return a status code of 403 (Forbidden). This flag can be used to conditionally block URLs.
gone | G	Forces the proxy server to return a status code of 410 (Gone)
proxy | P	Forces the substitution part through the proxy module. The substitution string must be a valid URL. This flag is useful as a sophisticated alternative to the `ProxyPass` directive.
last | L	Forces the rewrite process to end here
next | N	Reruns the rewrite process, starting with the first instance of `RewriteRule` and using the outcome of the current rewrite process as new input.
chain | C	Chains the current rule with the next rule, provided that the current rule matches.

Flag	Description
type | T [=mime-type]	Forces the proxy server to return the file as the specified MIME type.
nosubreq | NS	Indicates that the current rule applies only if the current request is not an internal subrequest.
passthrough | PT	Passes the substitution to the next handler, which should immediately follow the current `RewriteRule.`
skip | S [n]	Skips the next *n* rules in a sequence if the current rule matches.
env | E [=VAR ABLE:VALUE]	Sets the environment variable `VARIABLE` to the value `VALUE.`

HTTP Headers

The following table lists the base HTTP headers on which the `RewriteCond` and `RewriteRule` directives can operate:

Headers	
API_VERSION	REQUEST_METHOD
AUTH_TYPE	REQUEST_URI
DOCUMENT_ROOT	SCRIPT_FILENAME
HTTP_ACCEPT	SERVER_ADMIN
HTTP_COOKIE	SERVER_NAME
HTTP_FORWARDED	SERVER_PORT
HTTP_HOST	SERVER_PROTOCOL
HTTP_PROXY_CONNECTION	SERVER_SOFTWARE
HTTP_REFERER	SERVER_VERSION
HTTP_USER_AGENT	THE_REQUEST

Headers	
IS_SUBREQ	TIME_DAY
PATH_INFO	TIME_HOUR
QUERY_STRING	TIME_MIN
REMOTE_ADDR	TIME_MON
REMOTE_HOST	TIME_SEC
REMOTE_IDENT	TIME_WDAY
REMOTE_USER	TIME_YEAR
REQUEST_FILENAME	

Other Directives

The following directives are provided by additional modules or enhancements to the HP-UX Apache-based Web Server that are not available in the standard ASF Apache.

ProxyCookie

```
Syntax: ProxyCookie [On|Off]
Default: Off
```

This directive determines whether or not the Webproxy server sends a cookie to the client. To enable sending cookies to clients, set this directive to `On`. The “name=value” pair that is sent to the client consists of the name set by `ProxyCookieName` and the value determined by `ProxyCookieHashed` (hashed or not hashed name of the back-end server to which the client connects).

ProxyCookieName

```
Syntax:  ProxyCookieName character-string
Default: HPWP
```

This directive sets the name in the “name=value” pair of the cookie.

ProxyCookieExpires

```
Syntax: ProxyCookieExpires <integer>
Default: 300
```

This directive sets the expiration time, in seconds, of the cookie sent by the proxy. You should set its value to the maximum amount of time that you want to track or rewrite on the cookie header. For example, if the client-server affinity should last for 10 minutes, set the value to 600.

ProxyCookieForced

```
Syntax: ProxyCookieForced [On|Off]
Default: Off
```

When this directive is set to `Off` (or not set), the client is sent a cookie only if a cookie has not been set previously (and `ProxyCookie` is set to `On`). When this directive is set to `On`, a new cookie is sent to the client each and every time the client makes a request.

ProxyCookieHashed

```
Syntax:  ProxyCookieHashed [On|Off]
Default: Off
```

This directive determines whether or not the cookie contains a hashed representation of the name of the back-end server to which the client connects.

ProxyDowngradeRequest

```
Syntax:  ProxyDowngradeRequest [On|Off]
Default: Off
```

When this directive is set to On, all requests are downgraded to HTTP/1.0. When this directive is Off all requests pass unmodified.

Note: Webproxy 1.0 (for Virtualvault) was modified to pass 1.1 requests as 1.1, and 1.0 requests as 1.0 while setting the Connection header value to close (to prevent persistent connections to the proxy). Unfortunately, some back-end systems looked at the version of the request and not at the Connection header values to determine if persistence is allowed (i.e. they were not protocol compliant). The ProxyDowngradeRequest directive was added to restore the functionality of the original mod_proxy to work with back-end servers that are not protocol compliant with respect to persistence.

ProxyClientCertificate

```
Syntax:  ProxyClientCertificate [On|Off]
Default: Off
```

This directive extracts the client's verified certificate and places it in the header for propagation to a back-end server. In order for this directive to correctly propagate the client browser certificate, the Webproxy must have SSL enabled, and must require a client certificate (SSLVerifyClient).

ProxyVVCompatVars

```
Syntax: ProxyVVCompatVars [On|Off]
Default: Off
```

This is a Virtualvault/Webproxy compatibility directive. In Virtualvault, some transaction information concerning the internet browser/server connection is placed in header variables and passed to the back-end server. These variables were passed in the earliest versions of the Virtualvault, and subsequent HTTP protocol

specifications and servers have evolved to include most of these values in headers of a different name.

VVAllowAbsoluteURI

```
Syntax: VVAllowAbsoluteURI [On|Off]
Default: Off
```

This directive allows or denies the default handling of absolute URIs by the Webproxy Server. Under normal conditions, Apache parses a request of the form "GET http://system1.com/ HTTP/1.1" and forwards it directly to system1 without any other proxy processing. This allows clients to create URL specifications that will connect through one more proxies to a back-end network. Because no other filtering or processing takes place, clients may probe the back-end intranet network by chaining absolute URIs to bypass the Webproxy system. Normally, Apache allows absolute URI chaining.

SSLVVDowngradeLogLevel

```
Syntax: SSLVVDowngradeLogLevel [On|Off]
Default: Off
```

In some cases, Internet Explorer will abruptly shut down and renegotiate an SSL connection causing some error messages to be written to the error log at the "error" level. This directive allows the reclassification of these particular error messages to be written at the "warn" level.

SSLVVProxyClientCertificate

```
Syntax: SSLVVProxyClientCertificate [On|Off]
Default: Off
```

This directive presents the proxy's client certificate to a back-end proxy or server for authentication of the proxy. This directive is used in conjunction with the standard "SSLProxy*" directives to allow a client proxy to authenticate to a

back-end proxy without re-keying the private key password during the first (or subsequent) proxied SSL request.

VVOptRenegotiate (an option to SSLOptions)

```
Syntax: SSLOptions +VVOptRenegotiate
Default: Off
```

Under unknown circumstances or conditions, Internet Explorer will abruptly drop a negotiated SSL session and renegotiate the session. This is very CPU intensive and may also result in new certificate warnings or passphrase requests for the client browser. This new negotiation is often unnecessary if the previous session is cached, and if no errors were encountered when verifying the customer certificate.

RewritePOST

```
Syntax: RewritePOST  [On|Off]
Default: Off
```

This directive takes POST data (from POST HTTP requests) and places it in a RewritePOST header. Since header variables may be manipulated by the Rewrite module/engine, then the POST data is available for processing (filtering) by the RewriteEngine. Since the POST data must be reinserted and resent as the body of the request, this directive works ONLY for proxied POST requests, and will yield undefined results when the RewritePOST directive is used with other (cgi/static html/java) than proxy requests.

6 • Troubleshooting

One of the more difficult aspects of configuring Webproxy is deciding on a proxying architecture for a particular installation.Whether an application should set up several internet Web Servers that proxy to a single intranet web server, or whether you should set up several web servers versus a single web server with multiple VirtualHosts, most configurations will require troubleshooting HTTP requests and there replies. Methods in this section utilize native Apache logging, *tusc*, and *ssldump*.

Note: *tusc* and *ssldump* are not HP products but are described here for your reference. HP cannot provide support for any problems with building or using these tools.

The most common Apache troubleshooting is simply the use and interpretation of native Apache log files. Each Webproxy server instance has its own logs directory (**/opt/hpws/apache/webproxy/servers/wp-<*serverid*>/logs**) and writes to its own set of log files. These are Apache log files (error_log, access_log, etc.) and form the first step in troubleshooting Webproxy connections. Server connections are recorded in the file specified by the server's AccessLog directive. Client-server errors, cgi errors, and general errors are recorded in the file specified by the ErrorLog directive. SSL connections and problems are recorded in the file specified by the SSL. Generally, Apache logging events are assigned levels that include: debug, info, notice, warn, error, crit, alert, emerg , although the Rewrite module logging uses levels 0 through 9. Since logging affects overall server performance, log levels are normally set high (error or above) or turned off. For troubleshooting purposes, however, log levels should record all events starting with the lowest levels to gather debug and tracing information. For more information on Apache logging, see the Apache Software Foundation webpage at **www.apache.org**.

If information derived from Apache log files is not sufficient to debug a particular Apache request/reply transaction, then *tusc* (downloadable from one of the many Software Porting And Archive Center mirror sites) may be used to obtain byte-level transaction information. While *tusc* has many arguments and options, the most useful features for troubleshooting Webproxy connections is the ability to see

system connections and ASCII representations of client connections, requests, proxy connections, and reply. A sample *tusc* output follows:

```
# ps -ef |grep httpd
root 20302     1  9 08:00:59 ?           0:00
/opt/hpws/apache/bin/httpd -f /opt/hpws/apache/webproxy/serv-
ers/wp-odilbert/con      www 20304 20302  1 08:00:59 ?         0:00
/opt/hpws/apache/bin/httpd -f /opt/hpws/apache/webproxy/serv-
ers/wp-odilbert/con
www 20305 20302  1 08:01:00 ?         0:00 /opt/hpws/apache/bin/httpd
-f /opt/hpws/apache/webproxy/servers/wp-odilbert/con
root 20307  2606  2 08:01:04 ttyp0   0:00 grep httpd
www 20303 20302  2 08:00:59 ?         0:00 /opt/hpws/apache/bin/httpd
-f /opt/hpws/apache/webproxy/servers/wp-odilbert/con

#tusc -r all -w all 20303 20304 20305
( Attached to process 20303 ("/opt/hpws/apache/bin/httpd -f
/opt/hpws/apache/webproxy/servers/wp-
odilbert/con") [32-bit] )
accept(11, 0x7f7e0938, 0x7f7e0a3c) ..............................
[sleeping]
( Attached to process 20304 ("/opt/hpws/apache/bin/httpd -f
/opt/hpws/apache/webproxy/servers/wp-
odilbert/con") [32-bit] )
read(5, 0x7f7e0a38, 1) ..........................................
[sleeping]
ksleep(PTH_CONDVAR_OBJECT, 0x4007da00, 0x4007da08, NULL) ..........
[sleeping]
accept(3, 0x400ea37c, 0x400ea38c) ...............................
[sleeping]
( Attached to process 20305 ("/opt/hpws/apache/bin/httpd -f
/opt/hpws/apache/webproxy/servers/wp-
odilbert/con") [32-bit] )
read(5, 0x7f7e0a38, 1) ..........................................
[sleeping]
ksleep(PTH_CONDVAR_OBJECT, 0x4007da00, 0x4007da08, NULL) ..........
[sleeping]
semop(213, 0x7b009e08, 1) .......................................
[sleeping]
accept(3, 0x400ea37c, 0x400ea38c) ................................ =
4
semop(213, 0x7b009e08, 1) ....................................... =
0
```

```
semop(213, 0x7b009e0e, 1) ..................................... =
0
ksleep(PTH_CONDVAR_OBJECT, 0x4007da00, 0x4007da08, NULL) ..... =
0
kwakeup(PTH_CONDVAR_OBJECT, 0x4007da00, WAKEUP_ONE, 0x7ac262c8) ... =
0
ksleep(PTH_MUTEX_OBJECT, 0x4007d984, 0x4007d98c, NULL) ........... =
0
sched_yield() ................................................. =
0
kwakeup(PTH_MUTEX_OBJECT, 0x4007d984, WAKEUP_ONE, 0x7ac262cc) ..... =
0
ksleep(RELATIVE_TIMEOUT_VALUE|PTH_SPINLOCK_OBJECT, 0x4007d98c, NULL,
0x7ae57300)
= 0
getsockname(4, 0x400ea344, 0x400ea354) ........................ =
0
fcntl(4, F_GETFL, 0) ......................................... =
2
fcntl(4, F_SETFL, 65538) ..................................... =
0
read(4, 0x401b00b8, 8000) ..................................... =
1117
    G E T   /   H T T P / 1 . 0 \r\nC o n n e c t i o n :   K e e p
    - A l i v e \r\nU s e r - A g e n t :   M o z i l l a / 4 . 7
    [ e n ]   ( W i n N T ;   I ) \r\nH o s t :   d i l b e r t . i
    d e v . a t l . h p . c o m : 8 0 8 0 \r\nA c c e p t :   i m a
    g e / g i f ,   i m a g e / x - x b i t m a p ,   i m a g e / j
    p e g ,   i m a g e / p j p e g ,   i m a g e / p n g ,   * / *
    \r\nA c c e p t - E n c o d i n g :   g z i p \r\nA c c e p t -
    L a n g u a g e :   e n \r\nA c c e p t - C h a r s e t :   i s
    o - 8 8 5 9 - 1 , * , u t f - 8   \r\n\r\n
gettimeofday(0x7ae57448, NULL) ................................ =
0
stat("/opt/hpws/apache/htdocs/", 0x7ae575c8) ..................... =
0
stat("/opt/hpws/apache/htdocs/index.html", 0x7ae57788) ............ =
0
open("/opt/hpws/apache/htdocs/index.html", O_RDONLY, 0) .......... =
11
read(4, 0x401b00b8, 8000) ....................................
ERR#11 EAGAIN
sendfile(4, 11, 0, 1456, 0x7ae576c8, 0) ........................ =
1790
```

```
close(11) ..................................................... =
0
write(8, 0x400f0998, 73) ...................................... =
73
    1 5 . 4 7 . 2 2 4 . 2 0 4   -   -   [ 1 1 / S e p / 2 0 0 3 : 0
    8 : 1 0 : 2 9   - 0 6 0 0 ]   " G E T   /   H T T P / 1 . 0 "
    2 0 0   1 4 5 6 \n
write(9, 0x400f0a08, 17) ...................................... =
17
    -   - >   / i n d e x . h t m l \n
write(10, 0x400f0a30, 28) ..................................... =
28
    M o z i l l a / 4 . 7   [ e n ]   ( W i n N T ;   I ) \n
kwakeup(PTH_SPINLOCK_OBJECT, 0x4007d98c, WAKEUP_ALL, NULL) ........ =
0
read(5, 0x7f7e0a38, 1) .....................................
[sleeping]
ksleep(PTH_CONDVAR_OBJECT, 0x4007da00, 0x4007da08, NULL) ..........
[sleeping]
semop(213, 0x7b009e08, 1) ..................................
[sleeping]
```

In the above example, you can see the determination of which servers to connect to (note that the server with PPID 1 is the "lead" server, and the spawned servers actually handle the request), the "GET / HTTP/1.0" request, the sendfile to sent the response back to the server, and the follow-up write to the various logs.

Finally, debugging ssl transactions can be quite difficult because the transactions are encrypted. In order to debug the transaction, it is necessary to utilize an "ssl-aware" tool that will display the transaction in clear text. For this, we recommend *ssldump*. *ssldump* is much like *tcpdump*, except that it accepts a private key password and utilizes the private key to decrypt transaction information and display them for the application integrator. Unfortunately, *ssldump* is not part of the software porting archive, so it will need to be compiled for use in debugging HP-UX ssl connections.

ssldump utilizes the private key of the server to decode an SSL session and display the session in clear text. The following session shows an actual (but edited) session for an Internet Explorer client contacting a Webproxy server (dilbert.wp.domain.com) through an intermediate internet proxy

(proxyfromclient.domain.com). In this example, proxyfromclient has been configured (under Internet Explorer's Tools->Internet Options-> Connections -> LAN Settings -> Proxy server) to be a proxy to the client browser. This forces the proxyfromclient.domain.com system to utilize the AllowCONNECT method to create an SSL tunnel between the client browser and Webproxy.

In the following example, one can see the client and server negotiating the SSL connection: Agreeing on cipher, validating certificates, generating a session key, and completing the HTTP request and reply, all in a clear text readable format.

The session was created by executing the command:

```
#ssldump -Ad -k odilbert_pkey_080803.pem host dilbert.wp.domain.com
```

ssldump, in turn, prompts the user for the private key password for the encrypted key odilbert_pkey_080803.pem.

```
********************** ssldump sample session start
********************
New TCP connection #1: proxyfromclient.domain.com(19007) <-> dil-
bert.wp.domain.com(8443)
1 1  0.0355 (0.0355)  C>S SSLv2 compatible client hello
  Version 3.1
  cipher suites
  TLS_RSA_WITH_RC4_128_MD5
  TLS_RSA_WITH_RC4_128_SHA
  TLS_RSA_WITH_3DES_EDE_CBC_SHA
  SSL2_CK_RC4
  SSL2_CK_3DES
  SSL2_CK_RC2
  TLS_RSA_WITH_DES_CBC_SHA
  SSL2_CK_DES
  TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
  TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
  TLS_RSA_EXPORT_WITH_RC4_40_MD5
  TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
  SSL2_CK_RC4_EXPORT40
  SSL2_CK_RC2_EXPORT40
  TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
  TLS_DHE_DSS_WITH_DES_CBC_SHA
  TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
```

```
1 2  0.0390 (0.0035)  S>CV3.1(74)  Handshake
      ServerHello
        Version 3.1
        random[32]=
          40 3f 97 58 6b 63 82 a7 f1 78 00 5f 8b 3b d3 ba
          5c 44 b1 05 b5 31 f9 a4 35 ae 56 1f 82 5e 76 b0
        session_id[32]=
          92 67 0c 1f 4f 94 18 3e f0 aa 0c 90 c3 a3 3a 6b
          36 01 c3 5f 90 f2 4d c4 a2 5d 4d 9c 35 b0 d2 86
        cipherSuite         TLS_RSA_WITH_RC4_128_MD5
        compressionMethod                   NULL
1 3  0.0390 (0.0000)  S>CV3.1(1315)  Handshake
      Certificate
1 4  0.0390 (0.0000)  S>CV3.1(4)  Handshake
      ServerHelloDone
1 5  0.0477 (0.0086)  C>SV3.1(134)  Handshake
      ClientKeyExchange
        EncryptedPreMasterSecret[128]=
          02 c8 74 b6 89 74 2a 45 bf d5 81 7f 44 e1 7b 45
          e9 22 57 d3 fa 10 4d a7 a9 59 a9 cf 66 56 91 c5
          37 b3 a1 4c 31 64 7b bf 29 f4 b2 ad d3 2d 97 e4
          d3 ca a8 0d 23 1a a5 cf a7 b9 98 4d d4 21 a2 d2
          dc a4 a5 01 fb e3 bf c1 a7 80 3a d0 05 2d 33 5d
          e7 91 68 fc bb 62 3b c7 92 5c 18 50 ef 43 62 dd
          db 25 27 cf 31 13 1b 05 db bd bf 8c b2 57 ff f7
          f3 b6 08 69 f3 d3 26 03 4f b8 b7 36 3f f5 45 48
1 6  0.0477 (0.0000)  C>SV3.1(1)  ChangeCipherSpec
1 7  0.0477 (0.0000)  C>SV3.1(32)  Handshake
      Finished
        verify_data[12]=
          72 f9 05 83 9c 8c 9b e3 bb 7e 08 d9

1 8  0.2549 (0.2071)  S>CV3.1(1)  ChangeCipherSpec
1 9  0.2549 (0.0000)  S>CV3.1(32)  Handshake
      Finished
        verify_data[12]=
          d1 e4 ac e5 d0 a2 4c 7d 68 dc 16 ba

1    0.4353 (0.1804)  C>S  TCP FIN
1    0.4361 (0.0008)  S>C  TCP FIN
New TCP connection #2: proxyfromclient.domain.com(19550) <-> dil-
bert.wp.domain.com(8443)
2 1  0.0408 (0.0408)  C>SV3.1(97)  Handshake
      ClientHello
        Version 3.1
```

```
        random[32]=
          40 3f 99 6e 0b d0 e0 ce 2a ce 13 55 e0 bc a5 49
          6d 10 67 1e 66 48 49 b6 ba 4f 34 4b b9 e6 0e 8c
        resume [32]=
          92 67 0c 1f 4f 94 18 3e f0 aa 0c 90 c3 a3 3a 6b
          36 01 c3 5f 90 f2 4d c4 a2 5d 4d 9c 35 b0 d2 86
        cipher suites
        TLS_RSA_WITH_RC4_128_MD5
        TLS_RSA_WITH_RC4_128_SHA
        TLS_RSA_WITH_3DES_EDE_CBC_SHA
        TLS_RSA_WITH_DES_CBC_SHA
        TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
        TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
        TLS_RSA_EXPORT_WITH_RC4_40_MD5
        TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
        TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
        TLS_DHE_DSS_WITH_DES_CBC_SHA
        TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
        compression methods
                  NULL
2 2  0.0419 (0.0011)  S>CV3.1(74)  Handshake
      ServerHello
        Version 3.1
        random[32]=
          40 3f 97 5a 3d b4 e2 bb 03 ad 19 4c 44 8d 0a 7d
          51 51 34 94 8e 94 cf 87 08 b7 b2 cf f1 93 bd 95
        session_id[32]=
          92 67 0c 1f 4f 94 18 3e f0 aa 0c 90 c3 a3 3a 6b
          36 01 c3 5f 90 f2 4d c4 a2 5d 4d 9c 35 b0 d2 86
        cipherSuite         TLS_RSA_WITH_RC4_128_MD5
        compressionMethod                   NULL
2 3  0.0419 (0.0000)  S>CV3.1(1)  ChangeCipherSpec
2 4  0.0419 (0.0000)  S>CV3.1(32)  Handshake
      Finished
        verify_data[12]=
          aa 51 5f aa bb 28 4f 60 9a 19 b2 29

2 5  0.0441 (0.0021)  C>SV3.1(1)  ChangeCipherSpec
2 6  0.0441 (0.0000)  C>SV3.1(32)  Handshake
      Finished
        verify_data[12]=
          c4 40 41 04 be 42 c8 aa 57 58 e3 73

2 7  0.0447 (0.0005)  C>SV3.1(1380)  application_data
    ---------------------------------------------------------------
```

```
    GET / HTTP/1.0
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-powerpoint, application/vnd.ms-excel, applica-
tion/msword, application/x-shockwave-flash, */*
    Accept-Language: en-us
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0;
.NET CLR 1.0.3705; .NET CLR 1.1.4322)
    Host: dilbert.wp.domain.com:8443
    Connection: Keep-Alive

    ---------------------------------------------------------------
2 8  0.0458 (0.0011)  S>CV3.1(524)  application_data
    ---------------------------------------------------------------
    HTTP/1.0 200 OK
    Date: Fri, 27 Feb 2004 19:15:38 GMT
    Server: Apache/2.0.48  HP-UX_Apache-based_Web_Server (Unix) DAV/2
mod_ssl/2.0.48 OpenSSL/0.9.7c
    Content-Location: index.html.en
    Vary: negotiate,accept-language,accept-charset
    TCN: choice
    Last-Modified: Fri, 16 Jan 2004 22:19:06 GMT
    ETag: "6db1-5b0-fdfc0a80;6dde-961-fdfc0a80"
    Accept-Ranges: bytes
    Content-Length: 1456
    Connection: close
    Content-Type: text/html; charset=ISO-8859-1
    Content-Language: en
    Expires: Fri, 27 Feb 2004 19:15:38 GMT

    ---------------------------------------------------------------
2 9  0.0464 (0.0006)  S>CV3.1(1472)  application_data
    ---------------------------------------------------------------
    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
    <html xmlns="http://www.w3.org/1999/xhtml">
    <head>
    <title>Test Page for Apache Installation</title>
    </head>
    <!-- Background white, links blue (unvisited), navy (visited), red
    (active) -->
    <body bgcolor="#FFFFFF" text="#000000" link="#0000FF"
    vlink="#000080" alink="#FF0000">
    <p>If you can see this, it means that the installation of the <a
    href="http://www.apache.org/foundation/preFAQ.html">Apache web
    server</a> software on this system was successful. You may now add
```

```
    content to this directory and replace this page.</p>

    <hr width="50%" size="8" />
    <h2 align="center">Seeing this instead of the website you
    expected?</h2>

    <p>This page is here because the site administrator has changed
the
    configuration of this web server. Please <strong>contact the per-
son
    responsible for maintaining this server with questions.</strong>
    The Apache Software Foundation, which wrote the web server soft-
ware
    this site administrator is using, has nothing to do with
    maintaining this site and cannot help resolve configuration
    issues.</p>

    <hr width="50%" size="8" />
    <p>The Apache <a href="manual/">documentation</a> has been
included
    with this distribution.</p>

    <p>You are free to use the image below on an Apache-powered web
    server. Thanks for using Apache!</p>

    <div align="center"><img src="apache_pb.gif" alt="" /></div>
    </body>
    </html>

    ---------------------------------------------------------------
2    0.0743 (0.0278)  S>C  TCP FIN
2    0.0746 (0.0002)  C>S  TCP FIN
********************** sslldump sample session end
********************
```

Troubleshooting

Index

Index

I n d e x

I n d e x