Oracle® Fusion Middleware

Administering Security for Oracle HTTP Server

12c (12.2.1.4.0)
E98954-02
January 2023

Oracle Fusion Middleware Administering Security for Oracle HTTP Server, 12c (12.2.1.4.0)

E98954-02

Copyright © 2021, Oracle and/or its affiliates.

Primary Author: Oracle Corporation

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle®, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Preface

Administering Security for Oracle HTTP Server guide describes about Oracle HTTP Server (OHS) security, how to configure Oracle HTTP Server, and log files. It also provides information on increasing the security of an OHS deployment, including:

- Best practices for implementing TLS
- Configuring HTTP Secure Headers to mitigate security issues
- Protecting Oracle HTTP Server against known web server attacks

Audience

This guide is intended for security administrators, application developers, and others responsible for managing the application operations securely and efficiently. This documentation is based on the assumption that you are already familiar with Apache HTTP Server.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documents

For more information, see the following documents in the Oracle Fusion Middleware documentation set:

- *Using Oracle WebLogic Server Proxy Plug-Ins*
- *Administering Oracle HTTP Server*

Note:

Readers using this guide in PDF or hard copy formats will be unable to access third-party documentation, which Oracle provides in HTML format only. To access the third-party documentation referenced in this guide, use the HTML version of this guide and click the hyperlinks.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 Introduction to Oracle HTTP Server Security

Oracle HTTP Server is a web server component for Oracle Fusion Middleware, which provides a listener for Oracle WebLogic Server and the framework for hosting static pages, dynamic pages, and applications over the web.

Oracle HTTP Server also provides key features such as single sign-on, clustered deployment, and high availability, which enhance its operations.

This document describes best practices, general security features, and guidelines for using Oracle HTTP Server. The topics are organized into the following chapters:

- Configuring SSL and TLS Security, describes how to use ciphers and protocols, server certificates, and location directives to ensure secure connections.
- Configuring HTTP Secure Headers, describes how different secure headers such as X-XSS-Protection, HTTP only, HSTS, content security policy headers, and so on, help to mitigate security issues while using Oracle HTTP Server.
- Protecting Oracle HTTP Server Against Known Web Server Attacks, provides information about protecting Oracle HTTP Server against DoS and slow HTTP attacks.

2

Configuring SSL and TLS Security

Oracle HTTP Server secures communication by using a Secure Sockets Layer (SSL) protocol.

SSL secures communication by providing message encryption, integrity, and authentication. The SSL standard allows the involved components such as browsers and HTTP servers to negotiate which encryption, authentication, and integrity mechanisms to use.

This chapter includes the following topics:

Configuring Protocols and Ciphers

Oracle recommends that you configure Oracle HTTP Server to support only the strongest ciphers and protocols. Following are the list of preferred protocols and ciphers:

Note:

In this release, the following are the most secure list of protocols and ciphers available. For the updated list of secure ciphers, see My Oracle Support (Doc ID: 2314658.1) "SSL Configuration Required to Secure Oracle HTTP Server After Applying Security Patch Updates".

- **Protocols**
  `TLSv1.2` is the only recommended protocol.
- **Ciphers**

```
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
```

```
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256
SSL_RSA_WITH_AES_128_CBC_SHA
SSL_RSA_WITH_AES_256_CBC_SHA
```

Using Server Certificates

Credentials such as certificates, trusted certificates, certificate requests, and private keys are stored in Oracle wallet.

Security best practices for keys and certificates include:

- Using Strong Keys
- Protecting the Keys
- Using Strong Cryptographic Hashing Algorithms
- Using a Certificate That Supports the Required Domain Name
- Using a CA Signed Certificate

Using Strong Keys

The private keys used to generate the cipher key must be strong for the anticipated lifetime of the private key and their corresponding certificates.

The best practice is to select a key size of minimum 2048 bits.

Protecting the Keys

Ensure that the wallet containing the private key is stored in a location that is protected from unauthorized access.

Using Strong Cryptographic Hashing Algorithms

Ensure that the certificates are signed using SHA-256 hashing algorithm. Certificates signed using MD5 or SHA-1 algorithms are not trusted by browsers as these algorithms are known to have cryptographic weaknesses.

Using a Certificate That Supports the Required Domain Name

Ensure that the server certificates in your Oracle HTTP server support the required domain name. The domain name or subject of the certificate must match the fully qualified name of the server that presents the certificate. Subject Alternative Name (SAN)s can be used to provide a specific listing of multiple names, in valid certificates.

For example, let us consider web applications accessible at `https://abc.example.com` and `https://xyz.example.com`. In this case, the certificate lists the subject's common name attribute as example.com, and lists two SANs - `abc.example.com` and `xyz.example.com`. These certificates are referred to as multiple domain certificates.

See Using SAN Certificates with Oracle HTTP Server in the *Administering Oracle HTTP Server*.

Also, ensure that the user does not see any certificate errors upon accessing the web application.

Using a CA Signed Certificate

For Internet facing applications, the certificates should be signed by one of the well-known certificate authorities (CAs) which are automatically trusted by operating systems and browsers.

Using Location Directive to Secure URIs

The `mod_ossl` module's `SSLCipherSuite` directive can be configured with `<Location>` blocks to allow only those clients that support strong SSL parameters to access an URI. This forces a renegotiation and allows only the clients that meet the new configuration.

Following is an example to configure a location directive to secure an URI:

```
# be liberal in general -
SSLCipherSuite ALL
<Location "/strong/area">
# but https://hostname/strong/area/ and below requires strong ciphersuites
SSLCipherSuite HIGH
</Location>
```

Enabling Perfect Forward Secrecy on Oracle HTTP Server

Perfect Forward Secrecy (PFS) is a feature of specific key agreement protocols that gives assurance that your session keys will not be compromised even if the private key of the server is compromised.

Oracle HTTP Server out of the box configuration does not explicitly enable Perfect Forward Secrecy feature. To enable PFS, do the following configuration changes in the Oracle HTTP Server:

1. Configure TLS1.2 protocol for OHS server using SSLProtocol directive. See SSLProtocol Directive in *Administering Oracle HTTP Server*.
2. Enforce the ordering of server cipher suites by setting SSLHonorCipherOrder to `ON`. See SSLHonorCipherOrder Directive in *Administering Oracle HTTP Server*.
3. Use ECC certificates in Oracle HTTP Server wallet. See Adding an ECC Certificate to Oracle Wallets with orapki in *Administering Oracle Fusion Middleware*.
4. Configure ECDHE_ECDSA Cipher Suites in OHS. For the list of supported ECDHE_ECDSA cipher suites, see SSLCipherSuite Directive in *Administering Oracle HTTP Server*.

3

Configuring HTTP Secure Headers

Oracle recommends that you set the values of the HTTP headers listed in this section to prevent exploitation of known vulnerabilities caused due to these headers not being set, or set with wrong/default values.

The following are some of the commonly used secure headers:

- Headers to Mitigate XSS Attacks
- HTTP Strict Transport Security Header
- Referrer-Policy
- X-Frame-Options Header to Mitigate Clickjacking Attempts
- X-Content-Type-Options
- ServerSignature
- ServerTokens
- Secure Flag for Cookies
- SameSite Flag for Cookies

Note:

Best practice is to set these headers at the application level. If it is not possible or if you want to exercise extra precaution, you can configure them in Oracle HTTP Server. See My Oracle Support document ID 2370975.1.

Headers to Mitigate XSS Attacks

Cross-Site Scripting (XSS) attacks occur when an attacker sends malicious code to a different end user through a web application, in the form of a browser side script. Faults in web applications allows XSS attacks to succeed and can occur wherever a web application uses input from a user without validating or encoding it.

For example, an attacker may use XSS to send a malicious script to an user. Unaware of the situation, the end user’s browser might execute the script assuming that the script is from a trusted source and might let the malicious script to access cookies, session tokens, or other sensitive information retained by the browser and used with that site.

Headers to mitigate XSS attacks include:

- Content Security Policy
- X-XSS-Protection
- HttpOnly

Content Security Policy

A Content Security Policy header helps to mitigate the risk of content injection by giving developers control over resources that can be requested on behalf of a worker.

The Content Security Policy is a browser side mechanism which allows you to create source whitelists such as JavaScript, CSS, images, and so on, for client side resources of your web application. The Content Security Policy instructs the browser through a special HTTP header, to only execute or render resources from those sources.

It is not possible for the web server to implement Content Security Policy because the web server does not know the list of domains from which an end-user's web application is allowed to load style sheets, images and so on. The web server also does not know if the application makes use of plugins, media, and so on and if these are to be allowed to be loaded.

Oracle recommends that a policy based on the application characteristics be defined and implemented using the Content Security Policy header.

The following are examples to use Content Security Policy rules:

Example: Rule 1

```
Content-Security-Policy: default-src 'self'
```

This is a restrictive rule and works in applications in which:

- All resources are hosted by the same domain of the given page.
- There are no inlines or evaluationss for scripts and style resources.

Example: Rule 2

```
Content-Security-Policy: default-src 'none'; script-src 'self';
connect-src 'self'; img-src 'self'; style-src 'self';
```

This policy:

- Does not allow resources like object, frame, and media to load.
- Allows images, scripts, AJAX, and CSS from the same origin.

Example: Rule 3

```
Content-Security-Policy: frame-ancestors self;
```

X-XSS-Protection

Note:

The X-XSS-Protection header has been deprecated by modern browsers and its use can introduce additional security issues on the client side. Therefore, Oracle recommends you to set the header as `X-XSS-Protection: 0` to disable the XSS Auditor, and not allow it to take the default behavior of the browser handling the response. Use Content-Security-Policy instead. See Content Security Policy.

The X-XSS-Protection header re-enables the XSS filter for a particular website, if the user has disabled it. It is a security best practice to include the X-XSS-Protection header in all HTTP responses.

This enables browser detection of reflected XSS attacks. Underlying reflected XSS vulnerabilities still need to be fixed in the underlying application. A browser detecting an XSS and blocking it does not mean that the application has an XSS vulnerability. In some cases, this header may block legitimate responses, for example, a message board where javascript programming is discussed.

Table 3-1 describes the X-XSS-Protection header values.

Table 3-1 Header Values and Descriptions

Value	Description
`0`	Filter disabled.
`1`	Filter enabled. If a cross-site site scripting attack is detected the browser sanitizes the page to stop the attack.
`1; mode=block`	Filter enabled. The browser prevents rendering of the page.
`1; report=http://<YOURDOMAIN>/<your_report_URI>`	Filter enabled. The browser sanitizes the page and reports the violation.

To mitigate the Cross-Site Scripting (XSS) attack, Oracle recommends that you make the following configuration changes:

1. Open the `httpd.conf` file using the Advanced Server Configuration page in Fusion Middleware Control or a text editor.
2. Add the following header configuration, after the `LoadModule` section:

```
<IfModule mod_headers.c>
Header set X-XSS-Protection "1; mode=block"
</IfModule>
```

Note:

Setting the header does not guarantee that these attacks will be prevented. You may want to consider other best practices as well.

HttpOnly

`HttpOnly` is an additional flag included in a Set-Cookie HTTP response header, which helps to mitigate the risk of client side script accessing the protected cookie.

If the `HttpOnly` flag is included in the HTTP response header, the cookie cannot be accessed through client side script (if the browser supports this flag). As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser will not reveal the cookie to a third party.

Example configuration:

```
<IfModule mod_headers.c>
 Header edit Set-Cookie ^(?!IGNOREME=).*$ $0;HttpOnly;secure
</IfModule>
```

Sometimes, it may be essential to make cookies available to javascript. The above configuration example provides a mechanism to specify that certain cookies should not have the `HttpOnly` flag set. If a particular cookie is not a candidate for the `HttpOnly` attribute, then replace the string `IGNOREME` with the cookie name in the configuration above.

To avoid the `HttpOnly` flag from being added to the response cookie called MYCOOKIE1, run the following command to replace `IGNOREME` with `MYCOOKIE1`:

```
Header edit Set-Cookie ^(?!MYCOOKIE1).*$ $0;HttpOnly;
```

To exclude multiple cookies, run the following command:

```
Header edit Set-Cookie ^(?!(IGNOREME=|IGNOREME1=)).*$ $0;HttpOnly;
```

HTTP Strict Transport Security Header

HTTP Strict Transport Security (HSTS) is a security enhancement in which a browser always connects to the site returning the HSTS headers over SSL/TLS, with-in a specific duration set in the header. All connections to the server over HTTP is automatically replaced with HTTPS, even if the user uses HTTP in the URL. HSTS header also prevents HTTPS click through prompts on browsers.

To enable HSTS policy header, add the following to your SSL enabled virtual host:

```
<VirtualHost example.com:4443>
Header always set Strict-Transport-Security "max-age=63072000;
preload; includeSubDomains"
</VirtualHost>
```

Referrer-Policy

The Referrer-Policy header contains the address of the previous web page that a user follows to accesses the currently requested page.

The Referrer-Policy header has many uses including analytics, logging, optimized caching, and more problematic uses such as tracking, or stealing information. It also has side effects such as inadvertent leaking of sensitive information.

To enable Referrer-Policy, set the Referrer-Policy header to:

```
Header always set Referrer-Policy "same-origin"
```

This ensures that a referrer is sent for same-site origins, and the cross-origin requests does not contain referrer information.

X-Frame-Options Header to Mitigate Clickjacking Attempts

Note:

The CSP frame-ancestors obsolete the X-Frame-Options header. If a resource has both policies, the CSP frame-ancestors policy is enforced and the X-Frame-Options policy is ignored.

X-Frame-Options is a server-side method of combating clickjacking.

Clickjacking, also known as a UI redress attack, is a method in which an attacker uses multiple, transparent or opaque layers to trick a user into clicking a button or link on a page, other than the one they believe they are clicking.

To enable X-Frame-Options, set the X-Frame-Options header to:

```
Header setifempty X-Frame-Options SAMEORIGIN
```

X-Content-Type-Options

The X-Content-Type-Options header is a response HTTP header used by the server to protect against MIME sniffing vulnerabilities.

MIME sniffing is used by browsers to determine an asset’s file format, when there is not enough metadata information for a particular asset.

The lack of X-Content-Type-Options header in response causes certain browsers to determine the content type and encoding of the response even when these properties are defined correctly. This can make the web application vulnerable to Cross-Site Scripting (XSS) attacks.

For example, some versions of Internet Explorer and Safari browsers treat responses with the content-type `text/plain` as HTML if they contain HTML tags.

To set the X-Content-Type-Options header, make the following change in all responses which contain user input:

```
<IfModule mod_headers.c>
Header always set X-Content-Type-Options nosniff
</IfModule>
```

ServerSignature

The `ServerSignature` directive allows you to configure a footer in the web server generated documents.

In a chain of proxies, the footer tells which of the chained servers produced a returned error message. The syntax is:

```
ServerSignature On | Off | EMail
```

The default value is `Off`, which suppresses the footer line. The value `On` adds a footer with the server version number and the server name of the serving virtual host. The value `EMail` additionally creates a `mailto:` reference to the value specified in the `ServerAdmin` directive of the referenced document.

ServerTokens

The `ServerTokens` directive allows you to configure the server HTTP response header.

This directive controls whether the server response header field which is sent back to the clients includes a description of the generic OS-type of the server as well as the information about compiled-in modules. The syntax is:

```
ServerTokens Full | OS | Minor | Minimal | Major | Prod | None | Custom
```

The default value is `Full`, which includes information about the OS-type of the server and the compiled-in modules. The value `Prod` includes the least information. The value `None` removes the server header. The value `custom` takes a string as a second argument, which is used as the value of the server header.

You can use `ServerTokens Custom` *`some-server-string`* to disguise the web server software when Oracle HTTP Server generates the response header. When a backend server generates the response, the server response header may come from the backend server depending on the proxy mechanism.

Note:

`ServerTokens Custom` *`some-server-string`* is a replacement for the `ServerHeader Off` setting in Oracle HTTP Server 10g.

Secure Flag for Cookies

The secure flag is an option that can be set when sending a new cookie to the user within an HTTP Response. The purpose of the secure flag is to prevent cookies from being transmitted in clear text. When the secure attribute is associated with a cookie, browsers will not transmit such cookies over clear-text, thus preventing unauthorized parties from accessing the cookie.

Example configuration:

```
<IfModule mod_headers.c>
Header edit Set-Cookie ^(.*)$ $1; Secure;
</IfModule>
```

SameSite Flag for Cookies

The `SameSite` attribute of the `Set-Cookie` HTTP response header allows you to declare if your cookie should be restricted to a first-party or same-site context.

The `SameSite` attribute accepts three values. The Syntax is:

```
SameSite=Lax | Strict | None
```

- `Lax`: Cookies are not sent on normal cross-site sub-requests (for example, to load images or frames into a third party site), but are sent when a user navigates to the origin site (that is, when following a link).
  This is the default cookie value if the `SameSite` attribute is not set explicitly. This ensures that users have reasonably robust defense against some classes of cross-site request forgery (CSRF) attacks.
- `Strict`: Cookies are sent in a first party context and not along with the requests initiated by the third party websites.
- `None`: Cookies are sent in all contexts, that is, in responses to both first-party and cross-origin requests. If you set `SameSite=None`, then you must also set the cookie `Secure` attribute. If you don't, the cookie will be blocked.

4

Protecting Oracle HTTP Server Against Known Web Server Attacks

This chapter provides guidance to protect Oracle HTTP Server against DoS attacks and slow HTTP attacks.

The web server attacks can be detected by monitoring frequently accessed URI and denying requests. The following are some of the ways to protect web servers against attacks:

- Securing Oracle HTTP Server Against DoS Attacks
- Protecting Oracle HTTP Server Against Slow HTTP Attacks
- Protecting Oracle HTTP Server Against Host Header Attacks

Securing Oracle HTTP Server Against DoS Attacks

Denial of Service (DoS) is the act of performing an attack, which prevents the system from providing services to legitimate users.

All network servers can be subject to DoS attacks that attempt to prevent responses to clients by tying up the resources of the server. It is not possible to prevent such attacks entirely, but you can mitigate the problems that they create.

Often the best anti-DoS tool can be a firewall or other operating-system configurations. For instance, you can configure almost all firewalls to limit the number of simultaneous connections from a single IP address or network, thereby preventing more than a few simple attacks.

Table 4-1 provides the list of directives that help tune Oracle HTTP Server to improve its performance. In addition, these directives enable server administrators to exercise greater control over abnormal client requests and thereby protect against potential DoS attacks. Tuning these directives can affect some of the applications' performance. Appropriate testing needs to be done to ensure proper working. The default value provided for each directive is acceptable in most cases.

Table 4-1 Tuning Directives for Oracle HTTP Server

Directive	Default Value	Description
`TimeOut`	`60 seconds`	Server wait time before failing a request.

`RequestReadTimeout`	`header=5-40 seconds, MinRate=500 seconds`
	
	`body=10 seconds, MinRate=500 seconds`
		Timeout for receiving request headers and request body from the client.
		
		See Protecting Oracle HTTP Server Against Slow HTTP Attacks.

Table 4-1 (Cont.) Tuning Directives for Oracle HTTP Server

Directive	Default Value	Description
`KeepAliveTimeout`	`5 milliseconds`	Amount of time the server waits for subsequent requests on a persistent connection. Setting this directive to a higher value may cause performance issues in servers that are heavily loaded. A higher timeout value may result in several server processes being occupied, waiting on connections with idle clients.
`LimitRequestBody`	`0`	Restricts the total size of the HTTP request body sent from the client. Configure this directive to limit resource consumption triggered by client input.
`LimitRequestFieldSize`	`8190 bytes`	Restricts the size of the HTTP request header allowed from the client. Configure this directive to limit resource consumption triggered by client input.
`LimitXMLRequestBody`	`1000000 bytes`	Limits the size of an XML-based request body. Configure this directive to limit the maximum size of an XML-based request body.
`AcceptFilter`	NA	Enables operating system specific optimizations for a listening socket. Using none as an argument will disable any accept filters for that protocol.
`MaxRequestWorkers`	`250`	Sets the limit on the number of simultaneous requests served. To increase the default value, you must also raise the `ServerLimit` directive.
`CGIDScriptTimeout`	NA	Limits the wait time for more output from the CGI program. By default, the value of `Timeout` directive is used. Configure the `CGIDScriptTimeout` directive to overwrite the default value. If you are using CGI applications, see CGIDScriptTimeout Directive in *Administering Oracle HTTP Server*.

For more information about performance tuning, see Tuning Oracle HTTP Server in *Tuning Performance*.

For more information about the directives, see Apache HTTP Server documentation.

Protecting Oracle HTTP Server Against Slow HTTP Attacks

Slow HTTP POST Denial of Service (DoS) attack is an application-level DoS attack that sends slow traffic to the server and consumes server resources by maintaining open connections for an extended period of time.

The slow HTTP POST DoS attacks, unlike bandwidth-consumption DoS attacks, does not require large amount of traffic to be sent to the server. It only requires that the client is able to maintain open connections for several minutes. If the server maintains too many open connections, then it may not be able to respond to new, legitimate connections.

The attack holds server connections open by sending crafted HTTP POST headers that contain content-length headers with large values, to inform the web server how much of data to expect. After the HTTP POST headers are fully sent, the HTTP POST message body is sent at slow speeds to prolong the completion of the connection and lock up server resources.

By waiting for the complete request body, the server helps the clients with slow or intermittent connections to complete requests, but exposes itself to abuse.

To address the slow HTTP attacks, Oracle recommends that you tune the `RequestReadTimeout` directive provided by `mod_reqtimeout` module. This directive specifies various timeouts for receiving the request headers and the request body from the client. If the client fails to send headers or body within the configured time, a 408 Request Timeout error is sent, thus preventing a denial of service attack.

To modify the `RequestReadTimeout` directive:

1. Open the `httpd.conf` file using the Advanced Server Configuration page of the Fusion Middleware Control application, or a text editor.
2. In the `LoadModule` section of the file, if `mod_reqtimeout` is not already configured, add the following line to load the `mod_reqtimeout` module:

   ```
   LoadModule reqtimeout_module "${PRODUCT_HOME}/modules/mod_reqtimeout.so"
   ```
3. Edit the following directive in the configuration:

   ```
   <IfModule reqtimeout_module>
     RequestReadTimeout header=20-40,MinRate=500 body=20,MinRate=500
    </IfModule>
   ```

> **Note:**
>
> Tuning this directive can affect some of the applications' performance. Appropriate testing needs to be done to ensure proper working. See "Required Testing" in My Oracle Support document ID 2350321.1.

Protecting Oracle HTTP Server Against Host Header Attacks

The HTTP Host header attacks exploit vulnerable websites that handle the value of the Host header in an unsafe way. If the server implicitly trusts the Host header, and fails to validate or escape it properly, an attacker may be able to use this input to inject harmful payloads that manipulate the server-side behavior.

One such way an attack can occur is when an application uses the input coming from the Host or X-Forwarded-Host request headers as part of the response without proper validation. Information from these headers should not be trusted as it is just another client side value an attacker can tamper with, which can result in unintended behavior. This can be mitigated by adding the following directives in your `httpd.conf` file:

- `RewriteCond %{HTTP_HOST} !^<your website>`
- `RewriteRule ^(.*)$ - [F,L]`

The `RewriteCond` directive checks the Host header which is sent with the request and matches it to the second argument `<your website>` provided in the directive. The `RewriteRule` directive will be executed only if the `RewriteCond` directive is set. The `RewriteRule` directive takes three arguments:

- The first argument specifies the URL that you want to replace. The value `^(.*)$` replaces any URL that is present.
- The second argument specifies the URL that you want to replace the first URL with. The value `-` specifies that you do not want to replace the first URL with any other value.
- The third argument is for flags. The first flag `F` signifies Forbidden, where the user receives a 403 Forbidden status code on the request. The second flag `L` signifies Last, which means that this will be the last `RewriteRule` considered in the configuration. If you have specified many `RewriteRule`s after the `L` flag, they will not be considered.

Note:

The `RewriteCond` and `RewriteRules` are not inherited. They have to be specified for each VirtualHost separately. If you don’t want to specify `RewriteRule` for every VirtualHost and want to keep the rules same throughout the configuration, you can make the scope of the above configuration global by adding the following lines in each of the VirtualHosts:

- `RewriteEngine On`
- `RewriteOptions Inherit #this will inherit global configuration`

Another way to achieve this is by specifying `RewriteOptions` in global context with the value `InheritDown` or `InheritDownBefore`. However, you must still add `RewriteEngine On` in the VirtualHosts.

See the My Oracle Support (Doc ID 2356329.1) and Apache Module mod_rewrite.