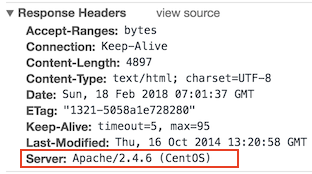
**Remove Server Version Banner**

I would say this is one of the first things to consider, as you don’t want to expose what web server version you are using. Exposing version means you are helping hacker to speedy the reconnaissance process.

The default configuration will expose Apache Version and OS type as shown below.



* Go to $Web\_Server/conf folder
* Modify httpd.conf by using the vi editor
* Add the following directive and save the httpd.conf

ServerTokens Prod

ServerSignature Off

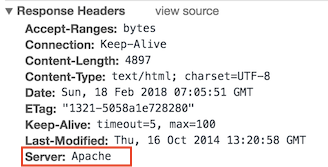
Copy

* Restart apache

ServerSignature will remove the version information from the page generated by Apache.

ServerTokens will change Header to production only, i.e., Apache

As you can see below, version & OS information is gone.



**Disable directory browser listing**

Disable directory listing in a browser, so the visitor doesn’t see what all file and folders you have under root or subdirectory.

Let’s test how does it look like in default settings.

* Go to $Web\_Server/htdocs directory
* Create a folder and few files inside that

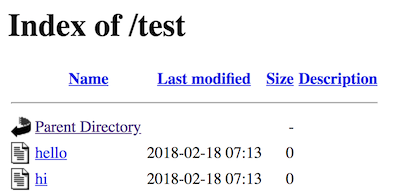
# mkdir test

# touch hi

# touch hello

Copy

Now, let’s try to access Apache by <http://localhost/test>



As you could see it reveals what all file/folders you have and I am sure you don’t want to expose that.

* Go to $Web\_Server/conf directory
* Open httpd.conf using vi
* Search for Directory and change Options directive to None or –Indexes

<Directory /opt/apache/htdocs>

Options -Indexes

</Directory>

Copy

(or)

<Directory /opt/apache/htdocs>

Options None

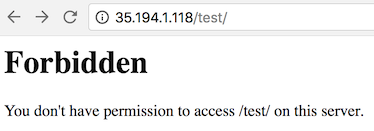
</Directory>

Copy

* Restart Apache

*Note*: if you have multiple Directory directives in your environment, you should consider doing the same for all.

Now, let’s try to access Apache by <http://localhost/test>



As you could see, it displays a forbidden error instead of showing test folder listing.

**Etag**

It allows remote attackers to obtain sensitive information like inode number, multipart MIME boundary, and child process through Etag header.

To prevent this vulnerability, let’s implement it as below. This is required to fix for PCI compliance.

* Go to $Web\_Server/conf directory
* Add the following directive and save the httpd.conf

FileETag None

Copy

* Restart apache

**Run Apache from a non-privileged account**

A default installation runs as nobody or daemon. Using a separate non-privileged user for Apache is good.

The idea here is to protect other services running in case of any security hole.

* Create a user and group called apache

# groupadd apache

# useradd –G apache apache

Copy

* Change apache installation directory ownership to a newly created non-privileged user

# chown –R apache:apache /opt/apache

Copy

* Go to $Web\_Server/conf
* Modify httpd.conf using vi
* Search for User & Group Directive and change as non-privileged account apache

User apache

Group apache

Copy

* Save the httpd.conf
* Restart Apache

grep for running http process and ensure it’s running with apache user

# ps –ef |grep http

Copy

You should see one process is running with root. That’s because Apache is listening on port 80 and it has to be started with root.

**Protect binary and configuration directory permission**

By default, permission for binary and configuration is 755 that means any user on a server can view the configuration. You can disallow another user to get into conf and bin folder.

* Go to $Web\_Server directory
* Change permission of bin and conf folder

# chmod –R 750 bin conf

Copy

**System Settings Protection**

In a default installation, users can override apache configuration using .htaccess. If you want to stop users from changing your apache server settings, you can add AllowOverride to None as shown below.

This must be done at the root level.

* Go to $Web\_Server/conf directory
* Open httpd.conf using vi
* Search for Directory at a root level

<Directory />

Options -Indexes

AllowOverride None

</Directory>

Copy

* Save the httpd.conf
* Restart Apache

**HTTP Request Methods**

HTTP 1.1 protocol support many request methods which may not be required and some of them are having potential risk.

Typically you may need GET, HEAD, POST request methods in a web application, which can be configured in the respective Directory directive.

Default configuration support OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, CONNECT method in HTTP 1.1 protocol.

* Go to $Web\_Server/conf directory
* Open httpd.conf using vi
* Search for Directory and add the following

<LimitExcept GET POST HEAD>

deny from all

</LimitExcept>

Copy

* Restart Apache

**Disable Trace HTTP Request**

By default Trace method is enabled in Apache web server.

Having this enabled can allow Cross Site Tracing attack and potentially giving an option to a hacker to steal cookie information. Let’s see how it looks like in default configuration.

* Do a telnet web server IP with listening port
* Make a TRACE request as shown below

#telnet localhost 80

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

TRACE / HTTP/1.1 Host: test

HTTP/1.1 200 OK

Date: Sat, 31 Aug 2013 02:13:24 GMT

Server: Apache

Transfer-Encoding: chunked

Content-Type: message/http 20

TRACE / HTTP/1.1

Host: test

0

Connection closed by foreign host.

#

Copy

As you could see in above TRACE request, it has responded my query. Let’s disable it and test it.

* Go to $Web\_Server/conf directory
* Add the following directive and save the httpd.conf

TraceEnable off

Copy

* Restart apache

Do a telnet web server IP with listen port and make a TRACE request as shown below

#telnet localhost 80

Trying 127.0.0.1...

Connected to localhost.

Escape character is '^]'.

TRACE / HTTP/1.1 Host: test

HTTP/1.1 405 Method Not Allowed

Date: Sat, 31 Aug 2013 02:18:27 GMT

Server: Apache Allow:Content-Length: 223Content-Type: text/html; charset=iso-8859-1 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"> <html><head>

<title>405 Method Not Allowed</title> </head><body>

<h1>Method Not Allowed</h1>

<p>The requested method TRACE is not allowed for the URL /.</p> </body></html>

Connection closed by foreign host.

#

Copy

As you could see in above TRACE request, it has blocked my request with HTTP 405 Method Not Allowed.

Now, this web server doesn’t allow TRACE request and help in blocking Cross Site Tracing attack.

**Set cookie with HttpOnly and Secure flag**

You can mitigate most of the common Cross Site Scripting attack using HttpOnly and Secure flag in a cookie. Without having HttpOnly and Secure, it is possible to steal or manipulate web application session and cookies, and it’s dangerous.

* Ensure mod\_headers.so is enabled in your httpd.conf
* Go to $Web\_Server/conf directory
* Add the following directive and save the httpd.conf

Header edit Set-Cookie ^(.\*)$ $1;HttpOnly;Secure

Copy

* Restart apache

**Clickjacking Attack**

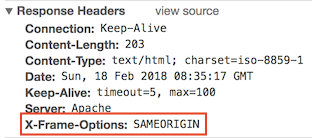
Clickjacking is a well-known web application vulnerabilities.

* Ensure mod\_headers.so is enabled in your httpd.conf
* Go to $Web\_Server/conf directory
* Add the following directive and save the httpd.conf

Header always append X-Frame-Options SAMEORIGIN

Copy

* Restart apache



X-Frame-Options also support two more options which I explained [here](https://geekflare.com/secure-apache-from-clickjacking-with-x-frame-options/).

**Server Side Include**

Server Side Include (SSI) has a risk of increasing the load on the server. If you have shared the environment and heavy traffic web applications you should consider disabling SSI by adding Includes in Options directive.

SSI attack allows the exploitation of a web application by injecting scripts in HTML pages or executing codes remotely.

* Go to $Web\_Server/conf directory
* Open httpd.conf using vi
* Search for Directory and add Includes in Options directive

<Directory /opt/apache/htdocs>

Options –Indexes -Includes

Order allow,denyAllow from all

</Directory>

Copy

* Restart Apache

Note: if you have multiple Directory directives in your environment, you should consider doing the same for all.

**X-XSS Protection**

Cross Site Scripting (XSS) protection can be bypassed in many browsers. You could apply this protection for a web application if it was disabled by the user. This is used by a majority of giant web companies like Facebook, Twitter, Google, etc.

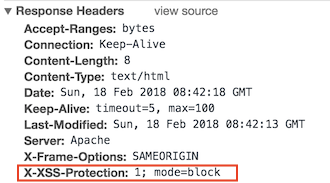
* Go to $Web\_Server/conf directory
* Open httpd.conf using vi and add following Header directive

Header set X-XSS-Protection "1; mode=block"

Copy

* Restart Apache

As you can see, XSS-Protection is the injected in the response header.



**Disable HTTP 1.0 Protocol**

When we talk about security, we should protect as much we can. So why do we use older HTTP version of the protocol, let’s disable them as well?

HTTP 1.0 has security weakness related to session hijacking. We can disable this by using the mod\_rewrite module.

* Ensure to load mod\_rewrite module in httpd.conf file
* Enable RewriteEngine directive as following and add Rewrite condition to allow only HTTP 1.1

RewriteEngine On

RewriteCond %{THE\_REQUEST} !HTTP/1.1$

RewriteRule .\* - [F]

Copy

**Timeout value configuration**

By default, Apache time-out value is 300 seconds, which can be a victim of Slow Loris attack and DoS. To mitigate this, you can lower the timeout value to maybe 60 seconds.

* Go to $Web\_Server/conf directory
* Open httpd.conf using vi
* Add the following in httpd.conf

Timeout 60

Copy

**SSL**

Having SSL is an additional layer of security you are adding into Web Application. However, default SSL configuration leads to certain vulnerabilities, and you should consider tweaking those configurations.

**SSL Key**

Breaching SSL key is hard, but not impossible. It’s just a matter of computational power and time.

As you might know, using a 2009-era PC cracking away for around 73 days you can [reverse engineer a 512-bit key](http://www.ticalc.org/archives/news/articles/14/145/145154.html).

So the higher key length you have, the more complicated it becomes to break SSL key. The majority of giant Web Companies use 2048 bit key, as below so why don’t we?

* Outlook.com
* Microsoft.com
* Live.com
* Skype.com
* Apple.com
* Yahoo.com
* Bing.com
* Hotmail.com
* Twitter.com

You can use OpenSSL to generate CSR with 2048 bit as below.

openssl req -out geekflare.csr -newkey rsa:2048 -nodes -keyout geekflare.key

Copy

It will generate a CSR which you will need to send to a [certificate authority](https://namecheap.pxf.io/c/245992/386170/5618?u=https%3A%2F%2Fwww.namecheap.com%2Fsecurity%2Fssl-certificates.aspx) to sign it. Once you receive the signed certificate file, you can add them in httpd-ssl.conf file

SSLCertificateFile #Certificate signed by authority

SSLCertificateChainFile #Certificate signer given by authority

SSLCertificateKeyFile #Key file which you generated above

Copy

* Restart Apache web server and try to access the URL with https

**SSL Cipher**

SSL Cipher is an encryption algorithm, which is used as a key between two computers over the Internet. Data encryption is the process of converting plain text into secret ciphered codes.

It’s based on your web server SSL Cipher configuration the data encryption will take place. So it’s important to configure SSL Cipher, which is stronger and not vulnerable.

* Go to $Web\_Server/conf/extra folder
* Modify SSLCipherSuite directive in httpd-ssl.conf as below to accept only higher encryption algorithms

SSLCipherSuite HIGH:!MEDIUM:!aNULL:!MD5:!RC4

Copy

* Save the configuration file and restart apache server

Note: if you have many weak ciphers in your SSL auditing report, you can quickly reject them adding ! at the beginning.

**Disable SSL v2 & v3**

SSL v2 & v3 has many security flaws, and if you are working towards penetration test or PCI compliance, then you are expected to close security finding to disable SSL v2/v3.

Any SSL v2/v3 communication may be vulnerable to a Man-in-The-Middle attack that could allow data tampering or disclosure.

Let’s implement apache web server to accept only latest TLS and reject SSL v2/v3 connection request.

* Go to $Web\_Server/conf/extra folder
* Modify SSLProtocol directive in httpd-ssl.conf as below to accept only TLS 1.2+

SSLProtocol –ALL +TLSv1.2

Copy

Once you are done with SSL configuration, it’s a good idea to test your web application with [online SSL/TLS Certificate tool](https://geekflare.com/ssl-test-certificate/) to find any configuration error.

**Mod Security**

Mod Security is an open-source Web Application Firewall, which you can use with Apache.

It comes as a module which you have to compile and install. If you can’t afford a commercial [web application firewall](https://geekflare.com/web-application-firewall/), this would be an excellent choice to go for it.

To provide generic web applications protection, the Core Rules use the following techniques:

* HTTP Protection – detecting violations of the HTTP protocol and a locally defined usage policy
* Real-time Blacklist Lookups – utilizes 3rd Party IP Reputation
* Web-based Malware Detection – identifies malicious web content by check against the Google Safe Browsing API.
* HTTP Denial of Service Protections – defense against HTTP Flooding and Slow HTTP DoS Attacks.
* Common Web Attacks Protection – detecting common web application security attack
* Automation Detection – Detecting bots, crawlers, scanners, and another malicious surface activity
* Integration with AV Scanning for File Uploads – identifies malicious files uploaded through the web application.
* Tracking Sensitive Data – Tracks Credit Card usage and blocks leakages.
* Trojan Protection – Detecting access to Trojans horses.
* Identification of Application Defects – alerts on application misconfigurations.
* Error Detection and Hiding – Disguising error messages sent by the server.

**Download & Installation**

Following prerequisites must be installed on the server where you wish to use Mod Security with Apache. If any one of these doesn’t exist then Mod Security compilation will fail. You may use yum install on Linux or Centos to install these packages.

* apache 2.x or higher
* libpcre package
* libxml2 package
* liblua package
* libcurl package
* libapr and libapr-util package
* mod\_unique\_id module bundled with Apache web server

Now, let’s download the latest stable version of Mod Security 2.7.5 from [here](https://www.modsecurity.org/download.html)

* Transfer downloaded file to /opt/apache
* Extract modsecurity-apache\_2.7.5.tar.gz

# gunzip –c modsecurity-apache\_2.7.5.tar.gz | tar xvf –

Copy

* Go to extracted folder modsecurity-apache\_2.7.5

# cd modsecurity-apache\_2.7.5

Copy

* Run the configure script including apxs path to existing Apache

# ./configure –with-apxs=/opt/apache/bin/apxs

Copy

* Compile & install with make script

# make

# make install

Copy

* Once the installation is done, you would see mod\_security2.so in modules folder under /opt/apache

Now this concludes, you have installed Mod Security module in existing Apache web server.

**Configuration**

To use Mod security feature with Apache, we have to load mod security module in httpd.conf. The mod\_unique\_id module is pre-requisite for Mod Security.

This module provides an environment variable with a unique identifier for each request, which is tracked and used by Mod Security.

* Add following a line to load module for Mod Security in httpd.conf and save the configuration file

LoadModule unique\_id\_module modules/mod\_unique\_id.so

LoadModule security2\_module modules/mod\_security2.so

Copy

* Restart apache web server

Mod Security is now installed!

Next thing you have to do is to install Mod Security core rule to take full advantage of its feature.

Latest Core Rule can be downloaded from following a link, which is free. <https://github.com/SpiderLabs/owasp-modsecurity-crs/zipball/master>

* Copy downloaded core rule zip to /opt/apache/conf folder
* Unzip core rule file
* You may wish to rename the folder to something short and easy to remember. In this example, I will rename to crs.
* Go to crs folder and rename modsecurity\_crs10\_setup.conf.example to modsecurity\_crs10\_setup.conf

Now, let’s enable these rules to get it working with Apache web server.

* Add the following in httpd.conf

<IfModule security2\_module>

Include conf/crs/modsecurity\_crs\_10\_setup.confInclude conf/crs/base\_rules/\*.conf

</IfModule>

Copy

In the above configuration, we are loading Mod Security main configuration file modsecurity\_crs\_10\_setup.conf and base rules base\_rules/\*.conf provided by Mod Security Core Rules to protect web applications.

* Restart apache web server

You have successfully configured Mod Security with Apache!

Well done. Now, Apache Web server is protected by Mod Security web application firewall.

**Getting Started**

Let’s get it started with some of the critical configurations in Mod Security to harden & secure web applications.

In this section, we will do all configuration modification in /opt/apache/conf/crs/modsecurity\_crs\_10\_setup.conf.

We will refer /opt/apache/conf/crs/modsecurity\_crs\_10\_setup.conf as setup.conf in this section for example purpose.

It’s important to understand what are the OWASP rules are provided for free. There are two types of rules provided by OWASP.

**Base Rules** – these rules are heavily tested, and probably false alarm ratio is less.

**Experimental Rules** – these rules are for an experimental purpose, and you may have a high false alarm. It’s important to configure, test and implement in UAT before using these in a production environment.

**Optional Rules** – these optional rules may not be suitable for the entire environment. Based on your requirement you may use them.

If you are looking for CSRF, User tracking, Session hijacking, etc. protection, then you may consider using optional rules. We have the base, optional and experimental rules after extracting the downloaded crs zip file from OWASP download page.

These rules configuration file is available in crs/base\_rules, crs/optional\_rules and crs/experimental\_rules folder. Let’s get familiar with some of the base rules.

* modsecurity\_crs\_20\_protocol\_violations.conf: This rule is protecting from Protocol vulnerabilities like response splitting, request smuggling, using non-allowed protocol (HTTP 1.0).
* modsecurity\_crs\_21\_protocol\_anomalies.conf: This is to protect from a request, which is missing with Host, Accept, User-Agent in the header.
* modsecurity\_crs\_23\_request\_limits.conf:This rule has the dependency on application specific like request size, upload size, a length of a parameter, etc.
* modsecurity\_crs\_30\_http\_policy.conf:This is to configure and protect allowed or disallowed method like CONNECT, TRACE, PUT, DELETE, etc.
* modsecurity\_crs\_35\_bad\_robots.conf:Detect malicious robots
* modsecurity\_crs\_40\_generic\_attacks.conf:This is to protect from OS command injection, remote file inclusion, etc.
* modsecurity\_crs\_41\_sql\_injection\_attacks.conf:This rule to protect SQL and blind SQL inject request.
* modsecurity\_crs\_41\_xss\_attacks.conf:Protection from Cross-Site Scripting request.
* modsecurity\_crs\_42\_tight\_security.conf:Directory traversal detection and protection.
* modsecurity\_crs\_45\_trojans.conf:This rule to detect generic file management output, uploading of HTTP backdoor page, known signature.
* modsecurity\_crs\_47\_common\_exceptions.conf:This is used as an exception mechanism to remove common false positives that may be encountered suck as Apache internal dummy connection, SSL pinger, etc.

**Logging**

Logging is one of the first things to configure so you can have logs created for what Mod Security is doing. There are two types of logging available; Debug & Audit log.

Debug Log: this is to duplicate the Apache error, warning and notice messages from the error log.

Audit Log: this is to write the transaction logs that are marked by Mod Security rule Mod Security gives you the flexibility to configure Audit, Debug or both logging.

By default configuration will write both logs. However, you can change based on your requirement. The log is controlled in SecDefaultAction directive. Let’s look at default logging configuration in setup.conf

SecDefaultAction “phase:1,deny,log”

Copy

To log Debug, Audit log – use “log” To log only audit log – use “nolog,auditlog” To log only debug log – use “log,noauditlog” You can specify the Audit Log location to be stored which is controlled by SecAuditLog directive.

Let’s write audit log into /opt/apache/logs/modsec\_audit.log by adding as shown below.

* Add SecAuditLog directive in setup.conf and restart Apache Web Server

SecAuditLog /opt/apache/logs/modsec\_audit.log

Copy

* After the restart, you should see modsec\_audit.log getting generated

**Enable Rule Engine**

By default Engine Rule is Off that means if you don’t enable Rule Engine you are not utilizing all the advantages of Mod Security.

Rule Engine enabling or disabling is controlled by SecRuleEngine directive.

* Add SecRuleEngine directive in setup.conf and restart Apache Web Server

SecRuleEngine On

Copy

There are three values for SecRuleEngine:

* On – to enable Rule Engine
* Off – to disable Rule Engine
* DetectionOnly – enable Rule Engine but never executes any actions like block, deny, drop, allow, proxy or redirect

Once Rule Engine is on – Mod Security is ready to protect with some of the common attack types.

**Common Attack Type Protection**

Now web server is ready to protect with common attack types like XSS, SQL Injection, Protocol Violation, etc. as we have installed Core Rule and turned on Rule Engine. Let’s test a few of them.

**XSS Attack**

* Open Firefox and access your application and put <script> tag at the end or URL
* Monitor the modsec\_audit.log in apache/logs folder

You will notice Mod Security blocks request as it contains <script> tag which is the root of XSS attack.

Directory Traversal Attack:- Directory traversal attacks can create a lot of damage by taking advantage of this vulnerabilities and access system related file. Ex – /etc/passwd, .htaccess, etc.

* Open Firefox and access your application with directory traversal
* Monitor the modsec\_audit.log in apache/logs folder

http://localhost/?../.../boot

Copy

* You will notice Mod Security blocks request as it contains directory traversal.

**Change Server Banner**

Earlier in this guide, you learned how to remove Apache and OS type, version help of ServerTokens directive.

Let’s go one step ahead, how about keeping server name whatever you wish? It’s possible with SecServerSignature directive in Mod Security. You see it’s interesting.

Note: to use Mod Security to manipulate Server Banner from a header, you must set ServerTokesn to Full in httpd.conf of Apache web server.

* Add SecServerSignature directive with your desired server name in setup.conf and restart Apache Web Server

SecServerSignature YourServerName

Copy

Ex:

[/opt/apache/conf/crs] #grep SecServer modsecurity\_crs\_10\_setup.conf

SecServerSignature geekflare.com

[/opt/apache/conf/crs] #

Copy

**General Configuration**

Let’s check out some of the general configurations as best practice.

**Configure Listen**

When you have multiple interfaces and IP’s on a single server, it’s recommended to have Listen directive configured with absolute IP and Port number.

When you leave apache configuration to Listen on all IP’s with some port number, it may create the problem in forwarding HTTP request to some other web server. This is quite common in the shared environment.

* Configure Listen directive in httpd.conf with absolute IP and port as a shown example below

Listen 10.10.10.1:80

Copy

**Access Logging**

It’s essential to configure access log properly in your web server. Some of the important parameter to capture in the log would be the time taken to serve the request, SESSION ID.

By default, Apache is not configured to capture these data. You got to configure them manually as follows.

* To capture time taken to serve the request and SESSION ID in an access log
* Add %T & %sessionID in httpd.conf under LogFormat directive

LogFormat "%h %l %u %t "%{sessionID}C" "%r" %>s %b %T" common

Copy

You can refer [http://httpd.apache.org/docs/2.2/mod/mod\_log\_config.html](https://httpd.apache.org/docs/2.2/mod/mod_log_config.html) for a complete list of parameter supported in LogFormat directive in Apache Web Server.

**Disable Loading unwanted modules**

If you have compiled and installed with all modules, then there are high chances you will have many modules loaded in Apache, which may not be required.

Best practice is to configure Apache with required modules in your web applications. Following modules have security concerns, and you might be interested in disabling in httpd.conf of Apache Web Server.

WebDAV (Web-based Distributed Authoring and Versioning) This module allows remote clients to manipulate files on the server and subject to various denial-of-service attacks. To disable comment following in httpd.conf

#LoadModule dav\_module modules/mod\_dav.so

#LoadModule dav\_fs\_module modules/mod\_dav\_fs.so

#Include conf/extra/httpd-dav.conf

Copy

Info Module The mod\_info module can leak sensitive information using .htaccess once this module is loaded. To disable comment following in httpd.conf

#LoadModule info\_module modules/mod\_info.so

Copy

Reference: This wouldn’t be possible without guidance from the following link:

* [http://httpd.apache.org/docs/2.4/](https://httpd.apache.org/docs/2.4/)
* [http://www.modsecurity.org/documentation/](https://www.modsecurity.org/documentation.html)
* <https://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project>

So that was some of the best practices you can use to secure your Apache web server.

If you are new to Apache HTTP, then I would recommend taking [the Apache HTTP administration course](https://click.linksynergy.com/deeplink?id=jf7w44yEft4&mid=39197&murl=https%3A%2F%2Fwww.udemy.com%2Fapache-web-server-administration%2F).