LAB 2: Information Leakage – Authorization - Web Application Security

1. Introduction

The Web Server is a crucial part of web-based applications. Apache Web Server is often placed at the edge of the network hence it becomes one of the most vulnerable services to attack.

Having default configuration supply much sensitive information which may help hacker to prepare for an attack the web server.

The majority of web application attacks are through XSS, Info Leakage, Session Management and PHP Injection attacks which are due to weak programming code and failure to sanitize web application infrastructure.

According to the security vendor Acunetix, 84% of tested applications have one or more vulnerabilities.

This practical guide provides you the necessary skill set to secure Apache Web Server.

In this course, we will talk about how to Harden & Secure Apache Web Server on Unix platform.

Following are tested on Apache 2.4.x version.

1. This assumes you have installed Apache on UNIX platform. If not, you can go through Installation guide.
2. We will call Apache installation directory **/etc/apache2 as $Web_Server**, **/var/www/html as $Web_Html** throughout this guide.
3. You are advised to take a backup of existing configuration file before any modification.

Contents

1.1 Audience

This is designed for Middleware Administrator, Application Support, System Analyst, or anyone working or eager to learn Hardening & Security guidelines.

Fair knowledge of Apache Web Server & UNIX command is mandatory.

Bonus: Click here to download this guide in PDF format.

1.1 Notes

We require some tool to examine HTTP Headers for some of the implementation verification. There is a way to do this.

Use browser inbuilt developer tools to inspect the HTTP headers. Usually, it's under Network tab

Mozilla Firefox

To view the request or response HTTP headers in Mozilla Firefox, take the following steps:

a. In Firefox, visit a URL, right-click, select Inspect Element to open the developer tools.

b. Select the Network tab or directly press Ctrl+Shift+E together from your computer keyboard.

c. Reload the page, select any HTTP request, and the HTTP headers will be displayed on the right panel.

Google Chrome:

To view the request or response HTTP headers in Google Chrome, take the following steps :

a. In Chrome, visit a URL, right click, select Inspect to open the developer tools.

b. Select Network tab.

c. Reload the page, select any HTTP request on the left panel, and the HTTP headers will be displayed on the right panel.

2. Information Leakage

In default Apache configuration you would have much sensitive information disclosures, which can be used to prepare for an attack.

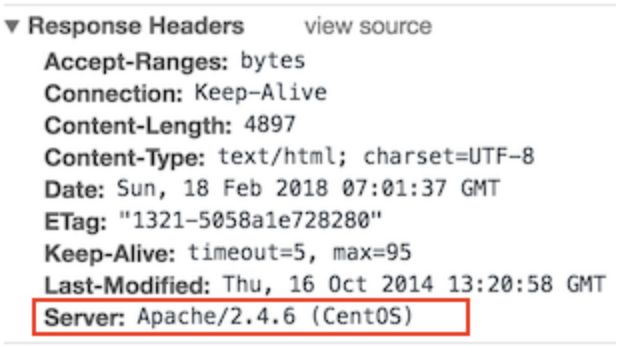
It's one of the most critical tasks for an administrator to understand and secure

them.

2.1 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.



Implementation

- Go to **$Web_Server** folder
- Modify **apache2.conf** by using vi or nano editor
- Add the following directive and save the apache2.conf

```
ServerTokens Prod

ServerSignature Off
```

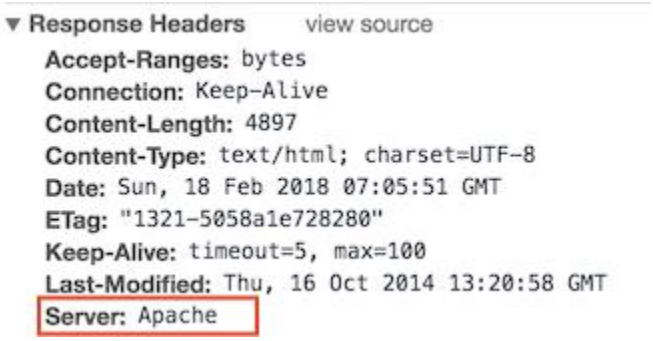
- Restart apache

`ServerSignature` will remove the version information from the page generated by apache web server.

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

Verification

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



2.2 Disable directory browser listing

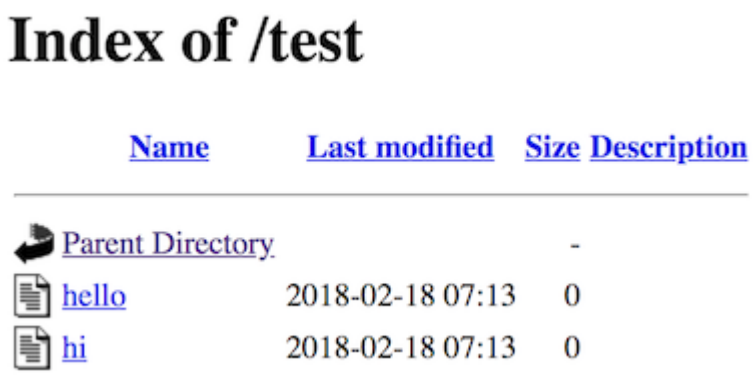
Disable directory listing in a browser, so the visitor doesn't see what all file and folders you have under root or subdirectory.

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

- Go to **$Web_Html** directory
- Create a folder ***test*** and few files inside that

```
# mkdir test

# touch hi

# touch hello
```

Now, let's try to access Apache by http://localhost/test



As you could see it reveals what all file/folders you have which are probably you don't want to expose.

Implementation

- Go to **$Web_Server** directory
- Open apache2.conf using vi
- Search for Directory and change Options directive to **None** or **–Indexes**

```
<Directory /var/www/html>

Options -Indexes

</Directory>
```

(or)

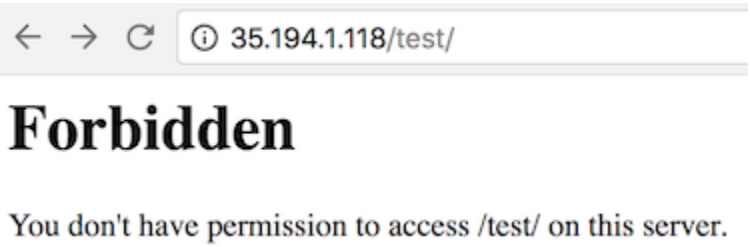
```
<Directory /var/www/html>

Options None

</Directory>
```

- Restart Apache

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

Verification

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.

2.3 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.

Implementation

- Go to $Web_Server directory
- Add the following directive and save the apache2.conf

```
FileETag None
```

- Restart apache

3. Authorization

3.1 Run Apache from non-privileged account

Default apache configuration is to run as nobody or daemon. It's good to use a separate non-privileged user for Apache.

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

Implementation

- Create a user and group called apache

```
# groupadd apache

# useradd –G apache apache
```

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

```
# chown –R apache:apache /etc/apache2
```

- Go to $Web_Server
- Modify apache2.conf using nano
- Search for User & Group Directive and change as non-privileged account apache

```
User apache

Group apache
```

- Save the apache2.conf
- Restart Apache

Verification

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

```
# ps  –ef   |   grep http
```

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.

3.2 Protect binary and configuration directory permission

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

Implementation

- Go to $Web_Server directory
- Change permission of bin and conf folder

```
# chmod –R 750 bin conf
```

3.3 System Settings Protection

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

This must be done at the root level.

Implementation

- Go to $Web_Server directory
- Open apache2.conf using vi
- Search for Directory at root level

```
<Directory />

Options -Indexes

AllowOverride None

</Directory>
```

- Save the apache2.conf
- Restart Apache

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

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

Implementation

- Go to $Web_Server directory
- Open apache2.conf using **nano**
- Search for Directory and add following

```
<LimitExcept GET POST HEAD>

deny from all

</LimitExcept>
```

- Restart Apache

4. Web Application Security

Apache web server misconfiguration or not hardened properly can exploit web application. It's critical to harden your web server configuration.

4.1 Cookies

4.1.1 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.#
```

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

Implementation

- Go to $Web_Server directory
- Add the following directive and save the apache2.conf

TraceEnable off

- Restart apache

Verification

- 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.#
```

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.

4.1.2 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.

Implementation

- Ensure mod_headers.so is enabled in your apache2.conf
- Go to $Web_Server directory
- Add the following directive and save the apache2.conf

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

- Restart apache

4.2 Clickjacking Attack

Clickjacking is well-known web application vulnerabilities.

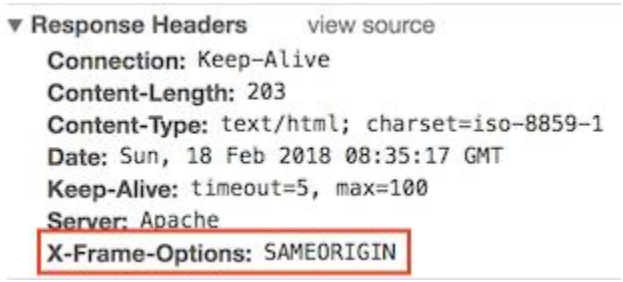
Implementation:

- Ensure mod_headers.so is enabled in your apache2.conf
- Go to $Web_Server directory
- Add the following directive and save the apache2.conf

```
Header always append X-Frame-Options SAMEORIGIN
```

- Restart apache

Verification



X-Frame-Options also support two more options which I explained here.

4.3 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.

Implementation

- Go to $Web_Server directory
- Open apache2.conf using nano
- Search for Directory and add Includes in Options directive

```
<Directory /etc/apache2/htdocs>

Options –Indexes -Includes

Order allow,denyAllow from all

</Directory>
```

- Restart Apache

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

4.4 X-XSS Protection

Cross Site Scripting (XSS) protection can be bypassed in many browsers. You can 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.

Implementation

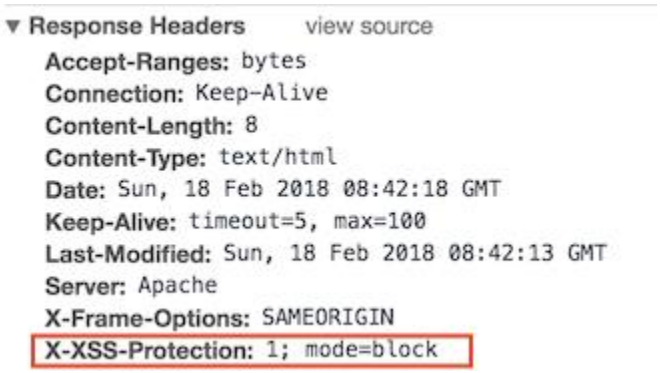
- Go to $Web_Server directory
- Open apache2.conf using vi and add following Header directive

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

- Restart Apache

Verification

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



4.5 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.

Implementation

- Ensure to load mod_rewrite module in apache2.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]
```

4.6 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.

Implementation

- Go to $Web_Server directory
- Open apache2.conf using nano
- Add following in apache2.conf

```
Timeout 60
```