

# Hardening a Linux Web App Part 2



Server Exploits - Module 5



# Response Headers

When you make a request to a web application, you get a response from the application with the HTML, JavaScript and other pages. Part of the response are the Response Headers, shown below. They are used to give more context about the site, set security controls and set cookies.

Request		Response	
Pretty	Raw	Pretty	Raw
1	GET / HTTP/1.1	1	HTTP/1.1 200 OK
2	Host: 192.168.11.134	2	Date: Mon, 10 Apr 2023 22:58:47 GMT
3	Cache-Control: max-age=0	3	Server: Apache/2.4.56 (Ubuntu)
4	Upgrade-Insecure-Requests: 1	4	X-Frame-Options: SAMEORIGIN
5	User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Safari/537.36	5	Expires: Tue, 23 Jun 2009 12:00:00 GMT
6	Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7	6	Cache-Control: no-cache, must-revalidate
7	Accept-Encoding: gzip, deflate	7	Pragma: no-cache
8	Accept-Language: en-US,en;q=0.9	8	Vary: Accept-Encoding
9	Connection: close	9	Set-Cookie: security=low; path=/; HttpOnly;
10		10	Set-Cookie: PHPSESSID=i97hisrcelonlksqla0jg8hkf; expires=Tue, 11-Apr-2023 22:58:47 GMT; Max-Age=86400; path=/; domain=192.168.11.134; HttpOnly;
11		11	Content-Length: 6284
		12	Connection: close
		13	Content-Type: text/html; charset=utf-8
		14	

# Response Headers

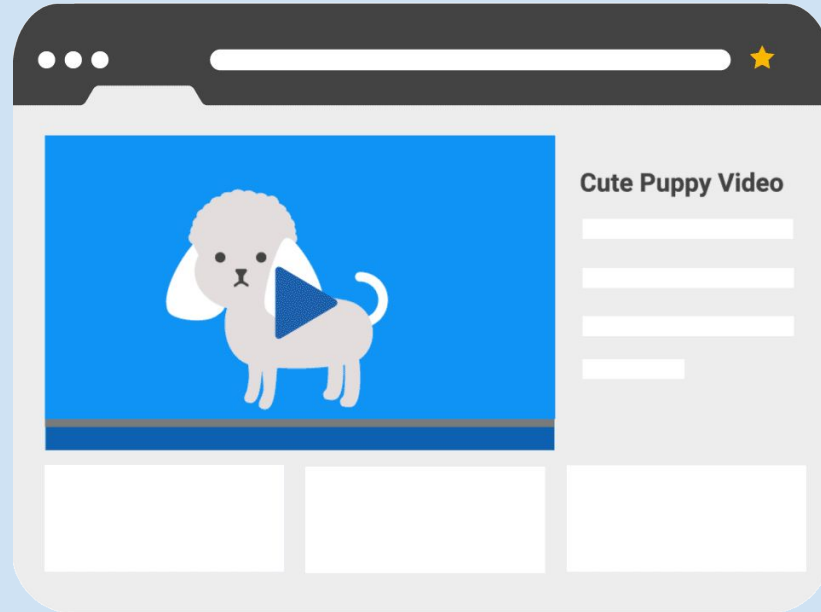
---

By default, there aren't many security headers set in a web application. It's up to us as an admin to set them.

# X-Frame-Options

---

If an attacker has the ability to inject HTML code (Cross-Site-Scripting) they can inject an invisible HTML element into the page. These are known as an inline frame (<iframe>), <object> and <embed>. These elements essentially puts another webpage (think embedded youtube videos), object (like a button) or embedded link within the original web application page. An iframe, etc. can be malicious in that it can go over a button and when a victim tries to click the button, they will interact with the iframe instead. This attack is called clickjacking.



# X-Frame-Options

---

A means of preventing this attack (in addition to filtering out special characters to prevent XSS) is to set an X-Frame-Options header. This is a response header can be used to indicate whether or not a browser should be allowed to render a page in an iframe element. There are several ways we can set the X-Frame-Options header:

1. Deny - completely disables the loading of the page in a frame or element. This is very secure, but can interfere with the functionality of a web application.
2. SameOrigin - only allows embedded pages from the same web app. In the case of DVWA, this would only allow embedded pages from our own DVWA site. This is the most common setting you'll see in web applications.
3. Allow-From <URL> - only allows the page to only be loaded in a frame on the specified origin and or domain. This is commonly used if you'd like to embed an external site, like YouTube.

# Enabling X-Frame-Options

In Apache 2.4.x you will first have to load mod\_headers with the following command. This module allows us to control and modify response headers within Apache's configuration files. Restart the service after running the command.

```
bryan@bryan-virtual-machine:/etc/apache2/sites-enabled$ sudo a2enmod headers
Enabling module headers.
To activate the new configuration, you need to run:
    systemctl restart apache2
```

# Enabling X-Frame-Options

---

We will then modify our site's configuration page (/etc/apache2/sites-enabled/000-default.conf) to have the line:

```
Header always set X-Frame-Options "SAMEORIGIN"
```

We can change the directive of the X-Frame-Options header by stating "DENY" or "ALLOW-FROM <URL>" here if we'd like to. This setting would prevent clickjacking in that any injected iframe elements will not be loaded into our DVWA

Restart the Apache service after making this change.

```
GNU nano 6.2                                000-default.conf
# Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
Header always set X-Frame-Options "SAMEORIGIN"
</VirtualHost>
```

# Enabling X-Frame-Options

If we then make a request through our Burp Proxy, we can see that the header is given in the response headers.

The screenshot displays the Burp Suite interface with a request and response captured. The 'Request' tab on the left shows an HTTP GET request to 192.168.11.134. The 'Response' tab on the right shows an HTTP 200 OK response from Apache/2.4.56 (Ubuntu). The 'X-Frame-Options: SAMEORIGIN' header is highlighted in the response headers, indicating that the server is configured to allow the page to be framed only by pages on the same origin.

```
Request
Pretty Raw Hex
1 GET / HTTP/1.1
2 Host: 192.168.11.134
3 Upgrade-Insecure-Requests: 1
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0
  Safari/537.36
5 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,im
  age/avif,image/webp,image/apng,*/*;q=0.8,application/sig
  ned-exchange;v=b3;q=0.7
6 Accept-Encoding: gzip, deflate
7 Accept-Language: en-US,en;q=0.9
8 Connection: close
9
10

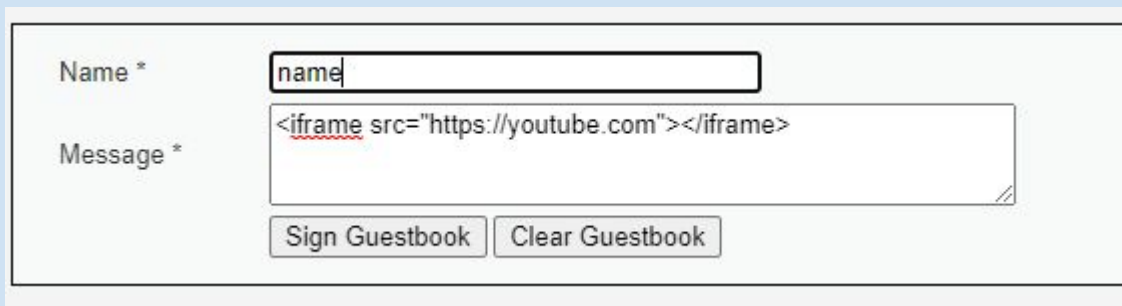
Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sun, 09 Apr 2023 16:02:55 GMT
3 Server: Apache/2.4.56 (Ubuntu)
4 X-Frame-Options: SAMEORIGIN
5 Set-Cookie: security=low; path=/
6 Set-Cookie: PHPSESSID=9q5o8drdqdlr9c0f4ferphqb16;
  expires=Mon, 10-Apr-2023 16:02:55 GMT; Max-Age=86400;
  path=/; domain=192.168.11.134
7 Expires: Tue, 23 Jun 2009 12:00:00 GMT
8 Cache-Control: no-cache, must-revalidate
9 Pragma: no-cache
10 Vary: Accept-Encoding
11 Content-Length: 6284
12 Connection: close
13 Content-Type: text/html; charset=utf-8
14
15 <!DOCTYPE html>
16
17 <html lang="en-GB">
18
19 <head>
20 <meta http-equiv="Content-Type" content="text/html;
  charset=UTF-8" />
21
22 <title>
  Welcome to Damn Vulnerable Web Application (DVWA)
```



# Testing X-Frame-Options

If we go to our stored XSS section, we can actually try adding an iframe element from an external site (I use youtube) and see the page doesn't load because of our header. This prevents attackers from loading malicious elements from external sites.

```
<iframe src="https://youtube.com"></iframe>
```



A screenshot of a web application's guestbook interface. It features a form with two input fields: "Name \*" and "Message \*". The "Name" field contains the text "name". The "Message" field contains the HTML payload `<iframe src="https://youtube.com"></iframe>`, with the word "iframe" underlined in red. Below the input fields are two buttons: "Sign Guestbook" and "Clear Guestbook". The entire form is enclosed in a light gray border.

# Testing X-Frame-Options

Name \*

Message \*

Sign Guestbook

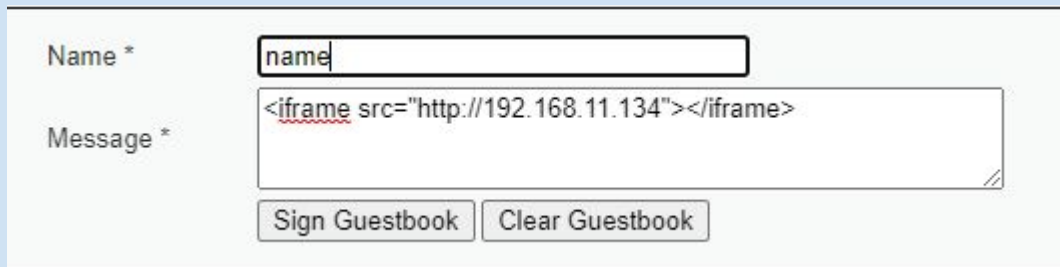
Clear Guestbook

Name: name  
Message:

# Testing X-Frame-Options

If we try to load an iframe from our own DVWA site, it will load properly because we set X-Frame-Options to SAMEORIGIN

```
<iframe src="http://<ubuntu_ip>"></iframe>
```



The screenshot shows a web form titled "Sign Guestbook" with two input fields and two buttons. The "Name \*" field contains the text "name". The "Message \*" field contains the HTML code `<iframe src="http://192.168.11.134"></iframe>`, with the word "iframe" underlined in red. Below the message field are two buttons: "Sign Guestbook" and "Clear Guestbook".

Name *	<input type="text" value="name"/>
Message *	<input type="text" value="&lt;iframe src='http://192.168.11.134'&gt;&lt;/iframe&gt;"/>
<input type="button" value="Sign Guestbook"/> <input type="button" value="Clear Guestbook"/>	

# Testing X-Frame-Options

Name \*

Message \*

Name: name  
Message:

Home

V

# Other Headers

---

Another important response header is HTTP Strict-Transport-Security (HSTS). This header when set will only allow clients to connect to the site over HTTPS. This prevents an attack known as SSL-Stripping, a type of downgrade attack. If an attacker is performing an adversary-in-the-middle attack and the victim connects to an HTTPS site without the HSTS header, the attacker can force the victim to communicate with the site over unencrypted HTTP.

We will not be setting this header for our DVWA as it does not use HTTPS.



# Cookies

---

In a web application cookies are data structures generated by a web application that are given to clients' browsers as they visit the web application. They can be used for just about anything, like remembering your searches on Amazon and your cart as you browse the site. Most commonly they are used as a session after someone logs in. After a user successfully logs into a site, they can be given a cookie that recognizes they have logged in, and as long as they have this cookie they have access to their profile on the site. These cookies will usually expire after the user is idle for a while or the user logs out.

# Cookies

In our DVWA, the site gives us 2 cookies - PHPSESSION and security. PHPSESSION is given after you login to DVWA to establish that you have logged in and then you can access all the sections. The security cookie is our difficulty setting. If we have a low security cookie, the site knows to render each section with a low difficulty. If we had changed our DVWA setting to high, the site would assign us a security cookie with the value "high". If you clear your cookies in your browser (usually in the settings of a browser) and visit DVWA, you'll see in the response Set-Cookie. This is the site setting the cookies for you as your browser doesn't have any for this site.

The image shows a web browser's developer tools with the Request and Response tabs open. The Request tab shows a GET request to /vulnerabilities/brute/. The Response tab shows an HTTP 200 OK response with various headers, including two Set-Cookie headers: security=low and PHPSESSID=8c0jr7pkn0ep2ml1j3lkes154b.

**Request**

```
1 GET /vulnerabilities/brute/ HTTP/1.1
2 Host: 192.168.11.134
3 Cache-Control: max-age=0
4 Upgrade-Insecure-Requests: 1
5 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64;
  x64) AppleWebKit/537.36 (KHTML, like Gecko)
  Chrome/111.0.0.0 Safari/537.36
6 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.
  9, image/avif, image/webp, image/apng, */*;q=0.8, applica
  tion/signed-exchange;v=b3;q=0.7
7 Accept-Encoding: gzip, deflate
8 Accept-Language: en-US,en;q=0.9
9 Connection: close
10
11
```

**Response**

```
1 HTTP/1.1 200 OK
2 Date: Sun, 09 Apr 2023 16:10:20 GMT
3 Server: Apache/2.4.56 (Ubuntu)
4 X-Frame-Options: SAMEORIGIN
5 Expires: Tue, 23 Jun 2009 12:00:00 GMT
6 Cache-Control: no-cache, must-revalidate
7 Pragma: no-cache
8 Vary: Accept-Encoding
9 Set-Cookie: security=low; path=/;HttpOnly;
10 Set-Cookie: PHPSESSID=8c0jr7pkn0ep2ml1j3lkes154b;
  expires=Mon, 10-Apr-2023 16:10:20 GMT;
  Max-Age=86400; path=/;
  domain=192.168.11.134;HttpOnly;
11 Content-Length: 4151
12 Connection: close
13 Content-Type: text/html; charset=utf-8
14
15 <!DOCTYPE html>
16
17 <html lang="en-GB">
18
19 <head>
20 <meta http-equiv="Content-Type" content="
```

# Cookies and Security

As we've seen when we exploit DVWA we can actually steal someone else's cookies via stored XSS. With these cookies, we can inherit the victim's session and gain access to the site as the victim without the need to know their credentials. There are some flags we can set for our cookies to prevent the cookies being sent via this attack.

1. Httponly - setting this flag prevents client-side scripts (JavaScript) from accessing the cookies. This stops our cookie-stealing XSS attack as our injected JavaScript won't be allowed to access a victim's cookies.
2. Secure - setting this flag will only allow the cookie to be used on an HTTPS connection, preventing attackers performing an adversary-in-the-middle attack and reading cookies in plaintext. We cannot set this one with our DVWA configuration as we do not use HTTPS.

# Setting the HTTPOnly flag

Like the X-Frame-Options header, we can set the httponly flag for all cookies in our 000-default.conf file. Usually, one would have to dive into DVWA's PHP code and modify how the PHP sets each specific cookie. However, Apache 2.4.x allows you to edit cookies before they are sent to the client as a response. This configuration will set the httponly flag for all cookies our DVWA application creates. If you wanted to add the secure flag as well, you'd append the line to have "Secure;" at the end. We will not be doing that as our DVWA does not use HTTPS.

```
GNU nano 6.2                                000-default.conf
# error, crit, alert, emerg.
# It is also possible to configure the loglevel for particular
# modules, e.g.
#LogLevel info ssl:warn

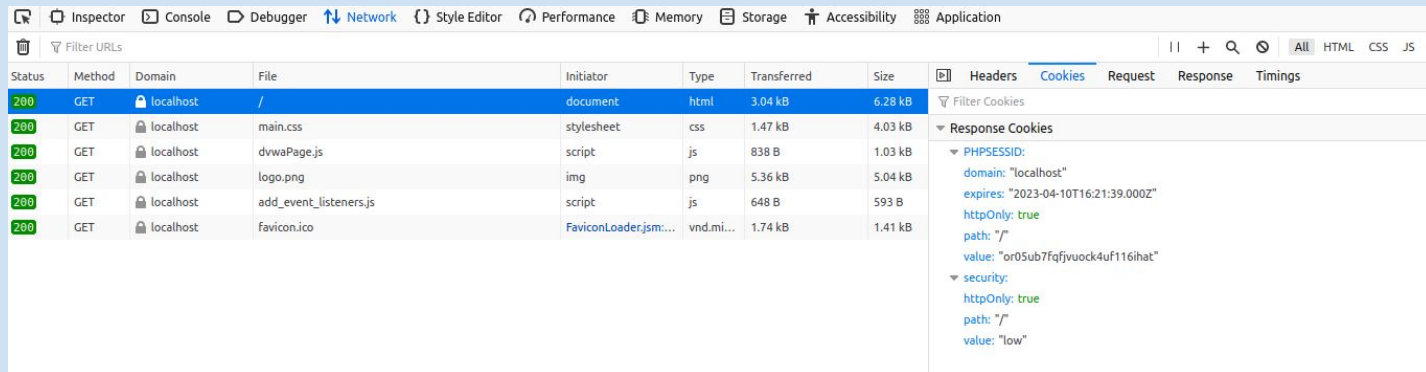
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# For most configuration files from conf-available/, which are
# enabled or disabled at a global level, it is possible to
# include a line for only one particular virtual host. For example the
# following line enables the CGI configuration for this host only
# after it has been globally disabled with "a2disconf".
#Include conf-available/serve-cgi-bin.conf
Header always set X-Frame-Options "SAMEORIGIN"
Header edit Set-Cookie ^(.*)$ $1;HttpOnly;
</VirtualHost>
```

# Setting the HTTPOnly flag

After you make this change and restart the Apache service, clear your cookies in your browser. BEFORE visiting DVWA, open your browser's developer tools (f12 key on Chrome and Firefox) and go to the Network tab. Then when you visit DVWA, you can see the cookies being assigned to you have the httponly flag. If you happen to visit DVWA before visiting the Network tab in your browser's developer tools, just clear your cookies again and refresh DVWA.

Chrome:



The screenshot shows the Chrome Developer Tools interface with the Network tab selected. The network log shows several requests to localhost, including the main page (index.html) and various assets (main.css, dvwaPage.js, logo.png, add\_event\_listeners.js, favicon.ico). The right-hand pane shows the 'Cookies' tab, which displays the 'Response Cookies' for the selected request. The cookies are:

- PHPSESSID:**
  - domain: "localhost"
  - expires: "2023-04-10T16:21:39.000Z"
  - httpOnly: true
  - path: "/"
  - value: "or0Sub7fqfjvuock4uf116ihat"
- security:**
  - httpOnly: true
  - path: "/"
  - value: "low"



# Setting the HTTPOnly flag

FireFox:

Name	× Headers Preview Response Initiator Timing Cookies								
exec/	<b>Request Cookies</b> <input type="checkbox"/> show filtered out request cookies								
main.css									
dvwaPage.js									
logo.png									
add_event_listeners.js									
favicon.ico									
	Name	Value	Domain	Path	Expires / ...	Size	HttpOnly	Secure	SameSite
	security	low	192.168.1...	/	Session	11	✓		
	PHPSESSID	8c0jr7pkn0ep2m11j3lkesl54b	192.168.1...	/	2023-04-...	35	✓		

# Setting the HTTPOnly flag

If we attempt our stored XSS attack (you can find the script to inject in Assignment 5) and run our HTTP server on our Kali machine, we will see the injected JavaScript make GET requests to our Kali machine. However, we will not see the cookies being sent over. This is because the httponly flag is preventing JavaScript from accessing the victim's cookies.

```
(bryan@kali)-[~]  
$ python3 -m http.server 8080  
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...  
192.168.11.1 - - [09/Apr/2023 12:14:08] "GET /? HTTP/1.1" 200 -  
192.168.11.1 - - [09/Apr/2023 12:14:16] "GET /? HTTP/1.1" 200 -
```