HTTP Related Attacks

Challenges

- **HTTP Response Splitting :** https://www.root-me.org/en/Challenges/Web-Client/HTTP-Response-Splitting
- **HTTP - Open redirect :** https://www.root-me.org/en/Challenges/Web-Server/HTTP-Open-redirect
- **HTTP - User-agent :** https://www.root-me.org/en/Challenges/Web-Server/HTTP-User-agent
- **HTTP - Headers :** https://www.root-me.org/en/Challenges/Web-Server/HTTP-Headers
- **HTTP - POST :** https://www.root-me.org/en/Challenges/Web-Server/HTTP-POST
- **HTTP - Improper redirect :** https://www.root-me.org/en/Challenges/Web-Server/HTTP-Improper-redirect
- **HTTP - Cookies:** https://www.root-me.org/en/Challenges/Web-Server/HTTP-Cookies
- **CRLF:** https://www.root-me.org/en/Challenges/Web-Server/CRLF

Notes

CRLF

https://www.netsparker.com/blog/web-security/crlf-http-header/

The HTTP headers and the HTML response are separated by a specific combination of special characters, namely a carriage return and a line feed known as CRLF.
The web server uses the CRLF to understand when new HTTP header begins and another one ends.
The CRLF can also tell a web application or user that a new line begins in a file or in a text block.
The CRLF characters are a standard HTTP/1.1 message, so it is used by any type of web server, including Apache, Microsoft IIS and all others.

In a CRLF injection vulnerability attack the attacker inserts both the carriage return and linefeed characters into user input to trick the server, the web application or the user into thinking that an object is terminated and another one has started. As such the CRLF sequences are not malicious characters, however they can be used for malicious intend, for HTTP response splitting etc.

123.123.123.123 - 08:15 - /index.php?page=home&
127.0.0.1 - 08:15 - /index.php?page=home&restrictedaction=edit

/index.php?page=home&**%0d%0a**127.0.0.1

HTTP Response Splitting

Refrences:
https://owasp.org/www-community/attacks/HTTP_Response_Splitting
https://blog.detectify.com/2019/06/14/http-response-splitting-exploitations-and-mitigations/

Since the header of a HTTP response and its body are separated by CRLF characters an attacker can try to inject those. **A combination of CRLFCRLF will tell the browser that the header ends and the body begins.** That means that he is now able to write data inside the response body where the html code is stored. This can lead to a Cross-site Scripting vulnerability.

?name=Bob**%0d%0a%0d%0a**<script>alert(document.domain)</script>

These characters not only give attackers control of the remaining headers and body of the response the application intends to send, but also allow them to create additional responses entirely under their control.

One of the most common attacks are to inject a second HTTP-response with defined headers and HTML content, which then leads to XSS. A vast majority of HTTP Response Splitting vulnerabilities occur due to the value of the `Location` header is not properly sanitized, and this is problematic for hackers as the browser will do the redirect and the hackers injected HTML will be ignored.

Because the `Location` value is empty, the web browser will ignore that header, meaning that the rest of the headers will be parsed by the web browser. As a result, no redirect will be made which means that the above attack would be successful.

Mitigations

- Properly sanitize values in the `Location` header. Filter characters such as \r and \n
- Don't let users control the whole value in the `Location` header
- When using a Content-Security-Policy, it should also be defined in the HTML

HTTP Header Injection

By exploiting a CRLF injection an attacker can also insert HTTP headers which could be used to defeat security mechanisms such as a browser's XSS filter or the same-origin-policy. This allows the attacker to gain sensitive information like CSRF tokens. He can also set cookies which could be exploited by logging the victim in the attacker's account or by exploiting otherwise unexploitable XSS.

Mitigation for CRLF / HTTP Header Injection

The best prevention technique is to not use users input directly inside response headers. If that is not possible, you should always use a function to encode the CRLF special characters. Another good web application security best practice is to update your programming language to a version that does not allow CR and LF to be injected inside functions that set HTTP headers.

HTTP Response splitting:

This type of Header Injection attack occurs when an attacker sends arbitrary data to the server, and this data is returned as part of the HTTP response.

the goal of the HTTP Response Splitting attack is to split the response into two parts by means of special characters such as:

- **CR** (Carriage Return) – [\r] - %0d
- **LF** (Line Feed) – [\n] - %0a

Since cookies are carried in the HTTP Headers, this is a good place to look for such a vulnerability.

example
GET http://target.site/trackUrl.php?url=http://elsfoo.com**%0d%0a**HTTP/1.1%2
0200%20OK%0d%0aContent%2dType:%20text/html%0d%0aContent%2dLength:%202
8%0d%0a%0d%0a<html><h1>Defaced</h1></html>

GET http://target.site/trackUrl.php?url=http://elsfoo.com%0d%0aHTTP/1.
1%20200%20OK%0d%0aContent%2dType:%20text/html%0d%0aContent%2dLengt h:
%2039%0d%0a%0d%0a<script>alert(document.cookie)</script>

HTML5 has introduced new origin headers to extend the origin concept. The attacker can inject these headers into the response and will be able to access the response body via JavaScript . Through Access-Control-Allow-Origin: http://attacker.site

Example:
GET http://target.site/getPersonalData.php?trackingUrl=http://elsfoo.com%0d% 0aAccess-Control-Allow-Origin:%20<attackerSite>%0d%0aAccess-Control- Allow-Credentials:%20true

Mitigation:

The simplest way to defend against header injection is to filter the input data and to remove these characters from the user input:
- **CR** (Carriage Return) – [\r]
- **LF** (Line Feed) – [\n]

In php : PHP offers a safe function named header, which is used to send a raw HTTP header.

```
<?php
        header(“Location: elsfoo.com”);
        die();
?>
```

Starting at PHP v5.1.4, both critical characters **CR** and **LF** have been denied within the function header, so it is considered safe against HTTP response splitting attacks; there is no need to filter the CR and LF characters.
Before PHP v5.1.4, you must explicitly filter the special characters from all user input.

Improper redirect :

sends the webpage and the correct content and makes the redirect on the browser’s side
Intercept with burp
Inside burp —> proxy —> options —> intercept requests.
Change the status to 200 OK

TO automatically change all responses in the proxy —> options —> match and replace