# TE.TE

The next example of HTTP request smuggling deals with a setting where both the reverse proxy and web server support chunked encoding. However one of the two systems does not act according to the specification such that it is possible to manipulate the TE header in such a way that one of the two systems accepts it and the other one does not, instead falling back to the CL header. Thus, it is possible to obfuscate the TE header such that one of the two systems does not parse it correctly. This type of HTTP request smuggling vulnerability is called TE.TE vulnerability. We will discuss how to identify and exploit this type of request smuggling vulnerability.

## Identification

To identify a TE.TE request smuggling vulnerability, we need to trick either the reverse proxy or the web server into ignoring the TE header. We can do this by slightly deviating from the specification to check whether the implementation of the two systems follows the specification accurately. For instance, some systems might only check for the presence of the keyword chunked in the TE header, while other systems check for an exact match. In such cases, it is sufficient to set the TE header to testchunked to trick one of the two systems to ignore the TE header and fall back to the CL header instead.

Here are a few options we could try to obfuscate the TE header from one of the two systems:

|  |  |
| --- | --- |
| **Description** | **Header** |
| Substring match | Transfer-Encoding: testchunked |
| Space in Header name | Transfer-Encoding : chunked |
| Horizontal Tab Separator | Transfer-Encoding:[\x09]chunked |
| Vertical Tab Separator | Transfer-Encoding:[\x0b]chunked |
| Leading space | Transfer-Encoding: chunked |

**Note:** The sequences [\x09] and [\x0b] are not the literal character sequences used in the obfuscation. Rather they denote the horizontal tab character (ASCII 0x09) and vertical tab character (ASCII 0x0b).

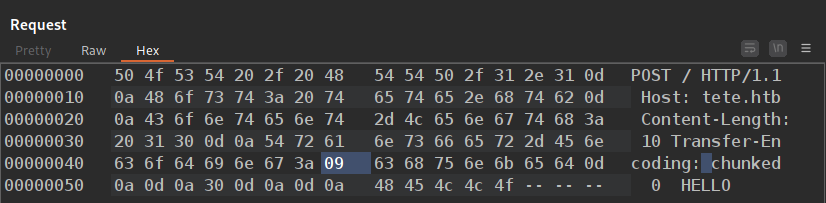
As an example, let's assume a scenario where we can trick the reverse proxy into ignoring the TE header with the Horizontal Tab method, while the web server parses it despite the tab. Since the reverse proxy falls back to the CL header, the identification and exploitation would then be the same as in a CL.TE scenario.

When starting the exercise below, we can see the same web application from the last section. However, this time it is not vulnerable to a simple CL.TE vulnerability. We can force a similar situation since the lab is vulnerable to a TE.TE attack. Let's copy the following request to a Burp Repeater tab:

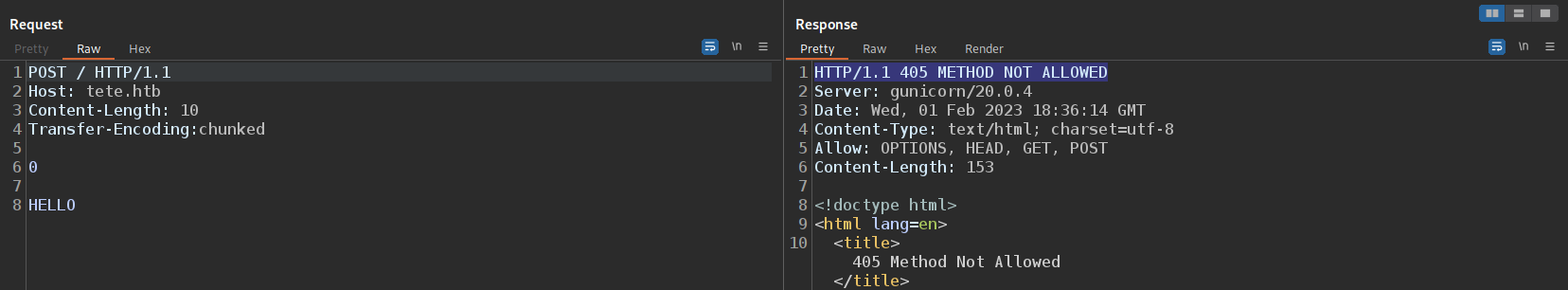
Code: http

POST / HTTP/1.1  
Host: tete.htb  
Content-Length: 10  
Transfer-Encoding: chunked  
  
0  
  
HELLO

Now let's replace the space that separates the TE header from the chunked value with a horizontal tab. We can do so by switching to the Hex view in the Repeater Tab and directly editing the space (0x20) to a horizontal tab (0x09):



When we now send the following request twice in quick succession, the second response should result in an HTTP 405 status code. We explored the reason for that in the previous section:



So, we successfully obfuscated the TE header from the reverse proxy, effectively leading to a CL.TE scenario.

## Exploitation

Since the setting is effectively equivalent to a CL.TE request smuggling scenario, the exploitation of our example is the same as discussed in the previous section. We just need to obfuscate the TE header in our request using the horizontal tab method to force the reverse proxy to fall back to the CL header.

We can use the following request to force the admin user to reveal the flag for us:

Code: http

POST / HTTP/1.1  
Host: tete.htb  
Content-Length: 46  
Transfer-Encoding: chunked  
  
0  
  
GET /admin?reveal\_flag=1 HTTP/1.1  
Dummy:

This time the lab is time-sensitive, so the admin request needs to hit the web server just after our malicious request. Therefore, we might need to send the request multiple times to get the timing right. Sending the request about once per second until the admin user hits the page and reveals the flag for us should work fine.

Generally, it is important to keep in mind that request smuggling vulnerabilities might be time-sensitive. There may be multiple worker threads or connection pools which may make exploitation of request smuggling vulnerabilities more challenging. Therefore, we often have to send our request multiple times. Particularly when targeting other users, we often need to get the timing just right just like in the sample lab below.

c