# Request Smuggling Tools & Prevention

After discussing different types of HTTP request smuggling vulnerabilities, let's have a look at tools that we can use to help us identify and exploit these types of attacks. Lastly, we will discuss how we can protect ourselves from HTTP request smuggling vulnerabilities.

## Tools of the Trade

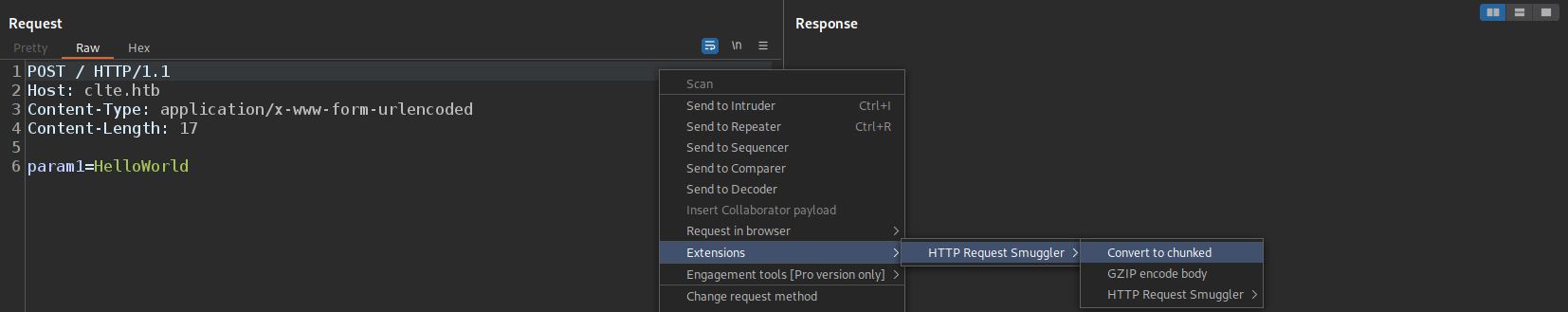
A useful tool to help us in the identification and exploitation of HTTP request smuggling vulnerabilities is the Burp Extension [HTTP Request Smuggler](https://github.com/PortSwigger/http-request-smuggler). We can install it from the Burp Extensions Store in the Extensions tab. Go to BApp Store and install the extension from there.

The first convenient functionality provided by the extension is the conversion of request bodies to chunked encoding. Since chunked encoding specifies the size of each chunk in hexadecimal format, we need to convert the length for each chunk from decimal to hexadecimal. The HTTP Request Smuggler extension can do this for us. To do so, send an arbitrary POST request to Burp repeater, for instance, the following:

Code: http

POST / HTTP/1.1  
Host: clte.htb  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 17  
  
param1=HelloWorld

We can then right-click the request and go to Extensions > HTTP Request Smuggler > Convert to chunked:

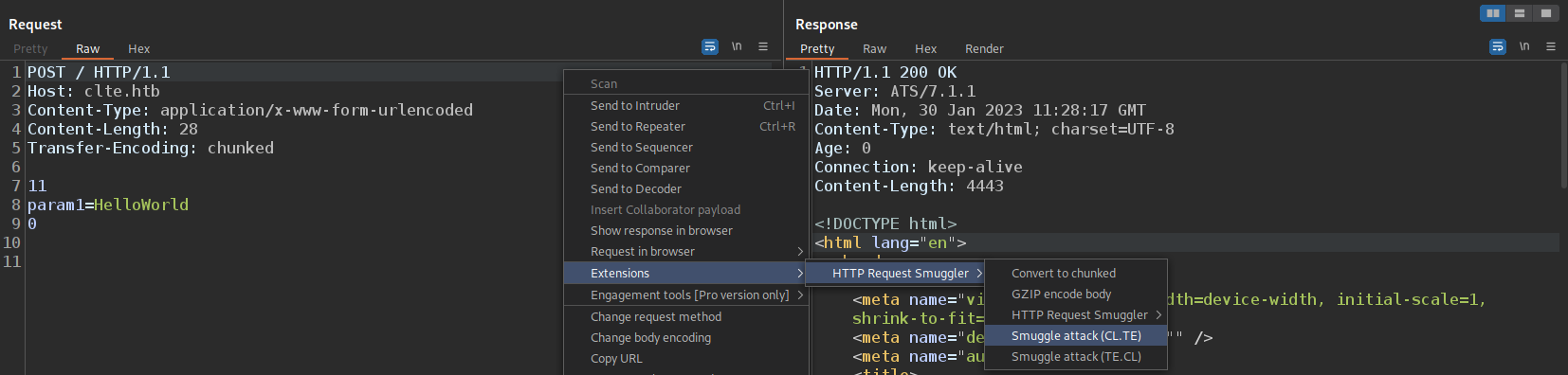


This will automatically convert the request to the chunked format for us:

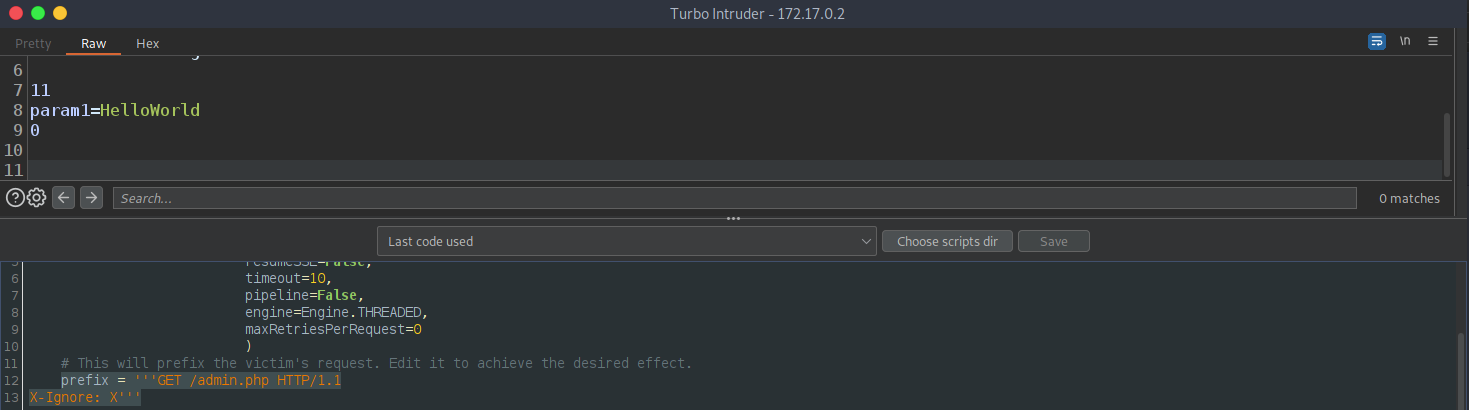
Code: http

POST / HTTP/1.1  
Host: clte.htb  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 28  
Transfer-Encoding: chunked  
  
11  
param1=HelloWorld  
0

Additionally, we can use the extension to exploit request smuggling vulnerabilities. As an example, let's consider a setup that is vulnerable to a CL.TE attack. We can exploit this using the extension by right-clicking our request formatted in chunked encoding and selecting Extensions > HTTP Request Smuggler > Smuggle attack (CL.TE):

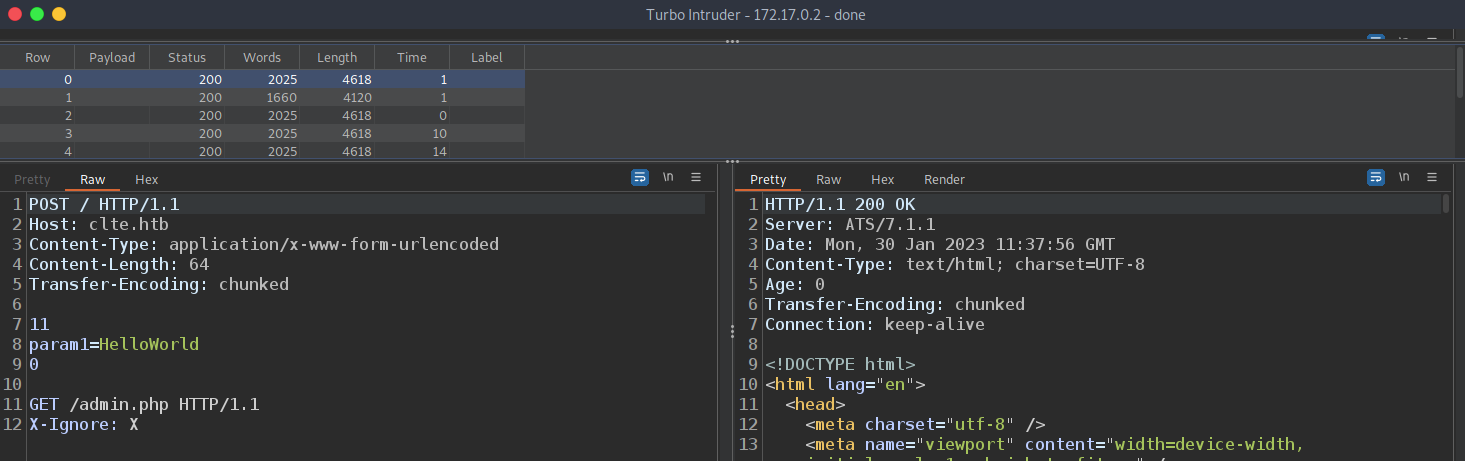


This will open a Turbo Intruder window for us. We can change the prefix in the attack script to customize the exploit. For a simple proof of concept, let's change the prefix to a GET request to /admin.php like so:

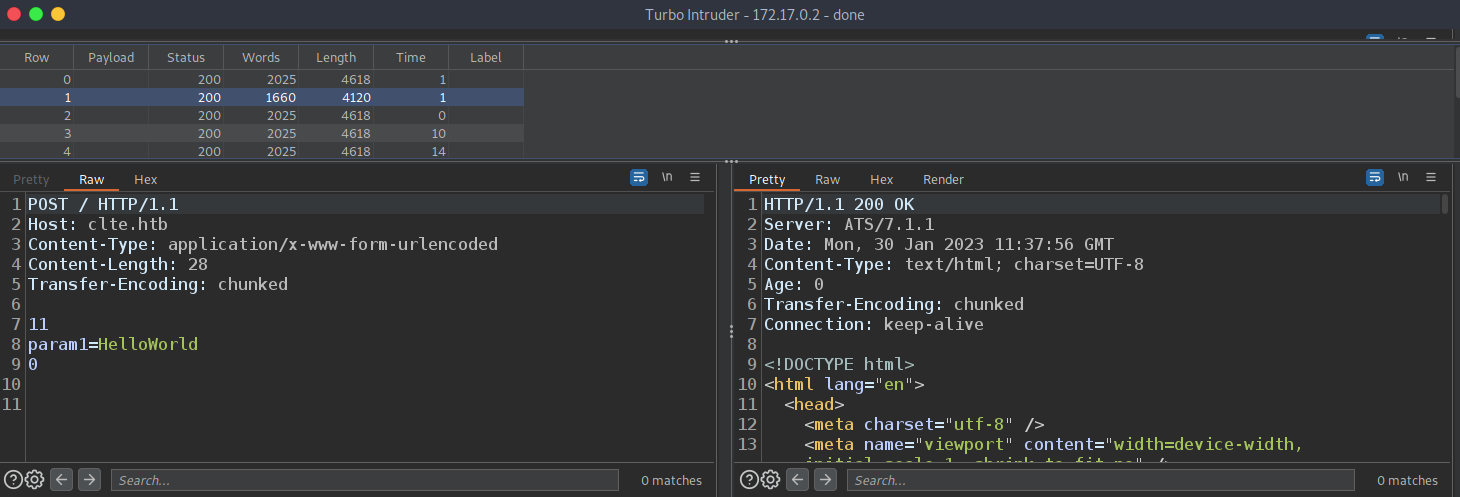


Click on the Attack button at the bottom of the Turbo Intruder window. Turbo Intruder will now periodically exploit the target once every second. After a few seconds, we can click on Halt to stop the attack and analyze the requests to determine whether the target is vulnerable.

The first request sent in each iteration is the crafted request that contains the smuggled request to /admin.php in its body:



While the remaining requests in each iteration do not contain the payload. They simulate the victim's request and are sent to trigger the vulnerability:



When looking at the response length in the table in the upper half of the two screenshots, we can see that the second request has a different response length. From that, we can conclude that the request smuggling vulnerability was successful. While the first request (and all other requests apart from the second one) have a response length of 4618 as the web server responds with the web application's index, the second response contains /admin.php, which is the response to our smuggled request. We can therefore conclude that the second request triggered the smuggled request, thus the setup is vulnerable to a CL.TE request smuggling vulnerability.

We could also adjust the exploit script to more specifically fit our needs by adding or removing victim requests, adding parameters to the smuggled request, or changing the sleep timer in-between iterations. Additionally, the HTTP request smuggler extension can be used the same way to exploit TE.CL vulnerabilities.

**Note:** To become familiar with the tool, feel free to play with it in the labs of the previous sections.

Tool