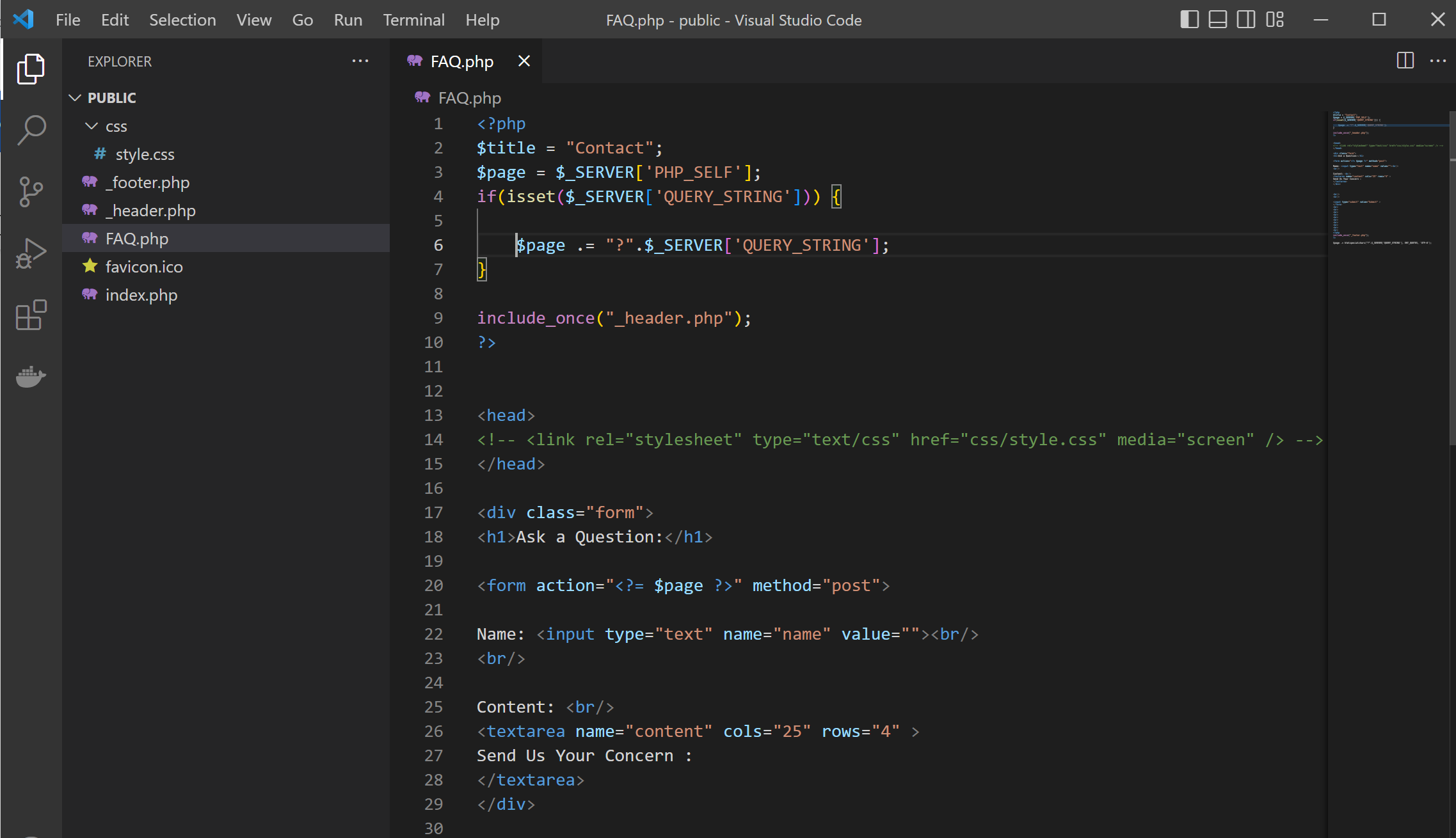
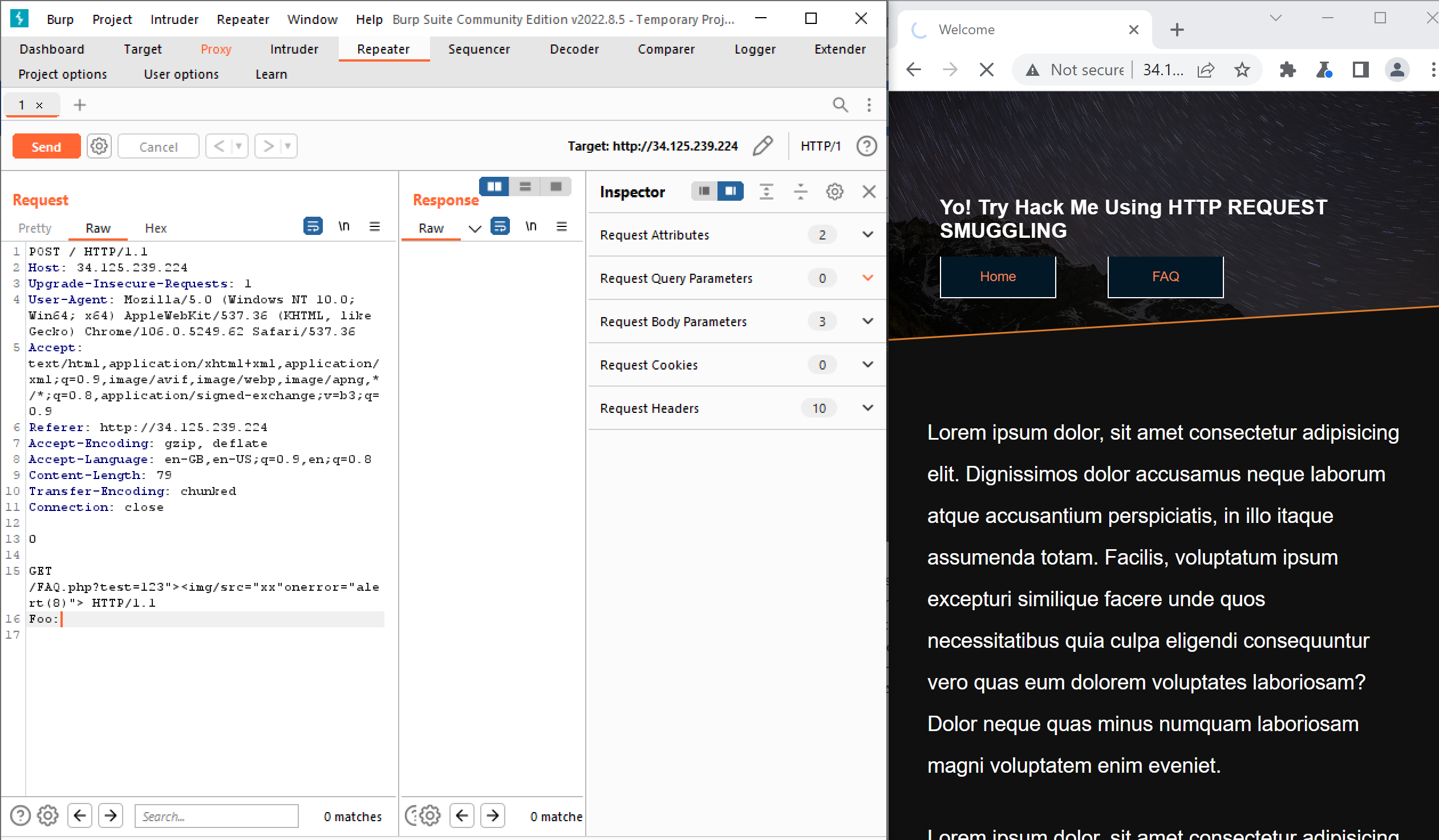
Vulnerable Web Application

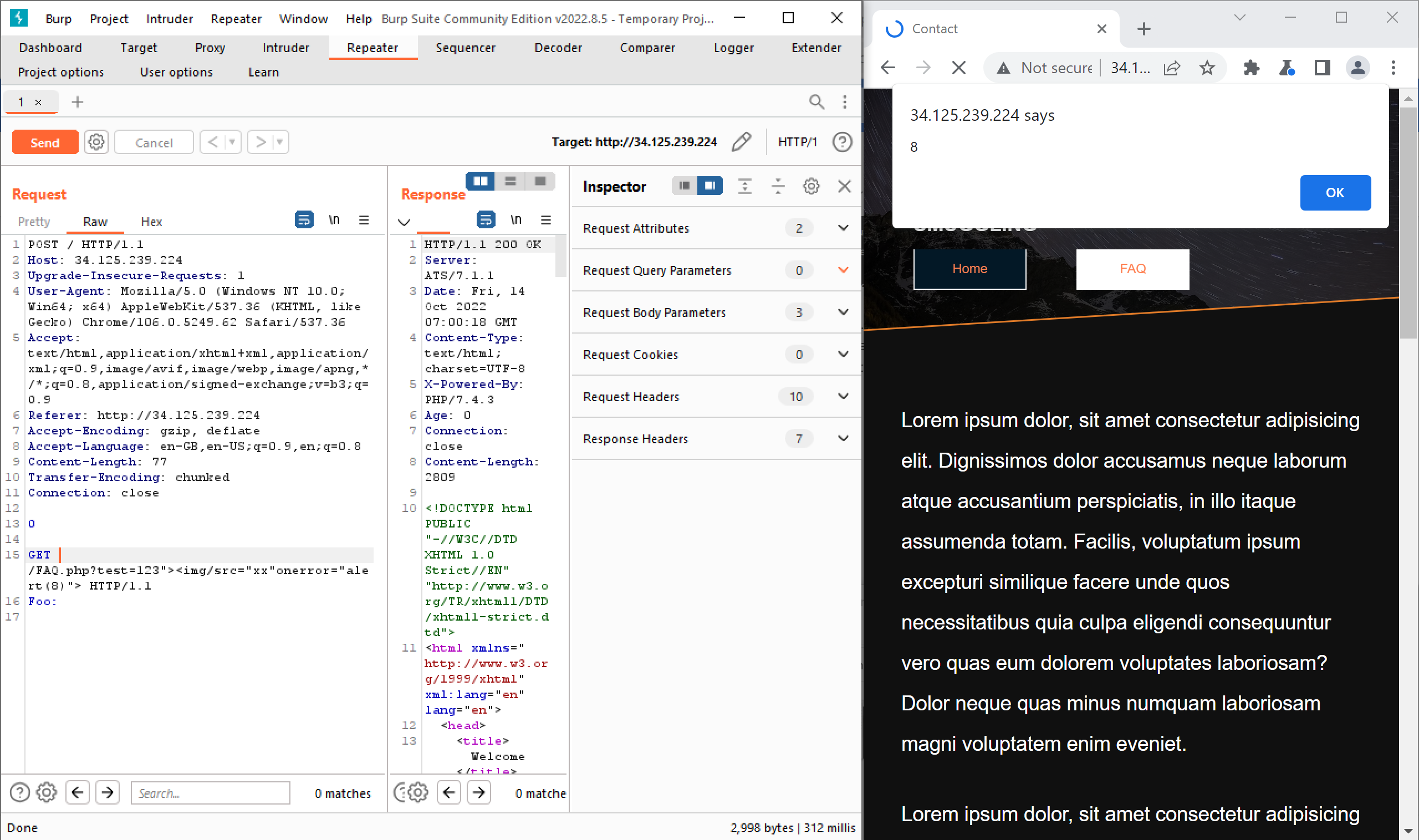
We developed a vulnerable web application from scratch, where we can apply http request smuggling techniques to perform an attack. Then we hosted the web application in google cloud. In the FAQ.php page in our web application, there is a text area for users to provide their feedback. But in the code, we developed the source code in a vulnerable way where the user-entered feedback is sent to the backend without encoding its’ characters. The vulnerable piece of code line is attached below (line number 6).



Because of this vulnerable piece of code the attacker can send crafted request as an input parameter using Http Request Smuggling Techniques (by parsing some headers like content length, transfer encoding) and perform the attack. In the demonstration, the attack we used is our plan was to pop an alert box in the web application using cross site scripting. What we did was we created a request which includes special characters like script tags. Because of the vulnerable piece of code in our web application the request we sent was not encoded and the web application understood it as a script tag and pop up an alert box in the web application. Attached below is the request we sent using burp suite.

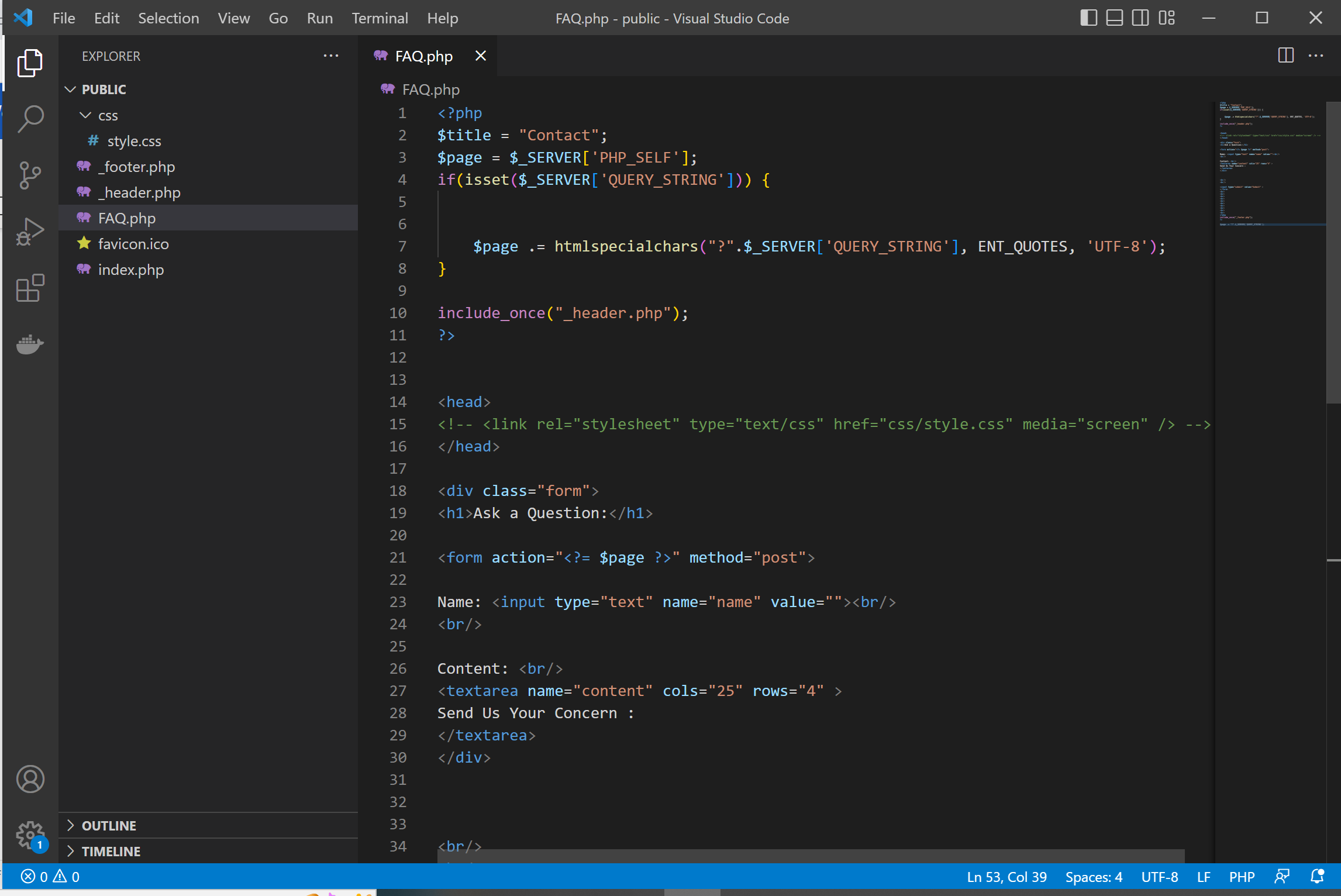


In this request, we can see we have used the POST method to pass a request. In the request we passed as an input parameter we can see a crafted, a cross-site scripted request is passing as the request. After we send this request and then we move to FAQ.php page in our web application, the attacker's intended alert box appeared there.



Now the attacker is successfully able to complete the attack. Why the attacker was able to perform the attack is the input parameters taken from the FAQ.php is not encoded and then pass to the backend. To apply the fix we found several methods. One is sending the requests using HTTP/2 instead of using HTTP/1.1. Another method is if we can use HTTPS by changing the port to 443 which is the secure port then the attacker won’t be able to perform the attack. Another method is source code side fixing. We planned to go with source code side fixing because, if we are going with changing HTTP to HTTPS then we need to host our web application again. So we planned to go with source code side fixing.

What we did was we removed the vulnerable piece of code and then applied the fixed code part which is responsible for encoding each special character user enters. Attached below is the fixed code part (Line number 7)



Now the attacker is unable to perform attacks using http request smuggling techniques since we fixed the code. In our case previously attacked was able to pop up a alert box in the web application by simply passing a crafted request. But now its fixed. Each special character attacker enters will be encoded and passes to the backend. Given below is a screenshot which proves the special characters are encoded (In line number 31 in the browser).

