HTTP Request Smuggling:

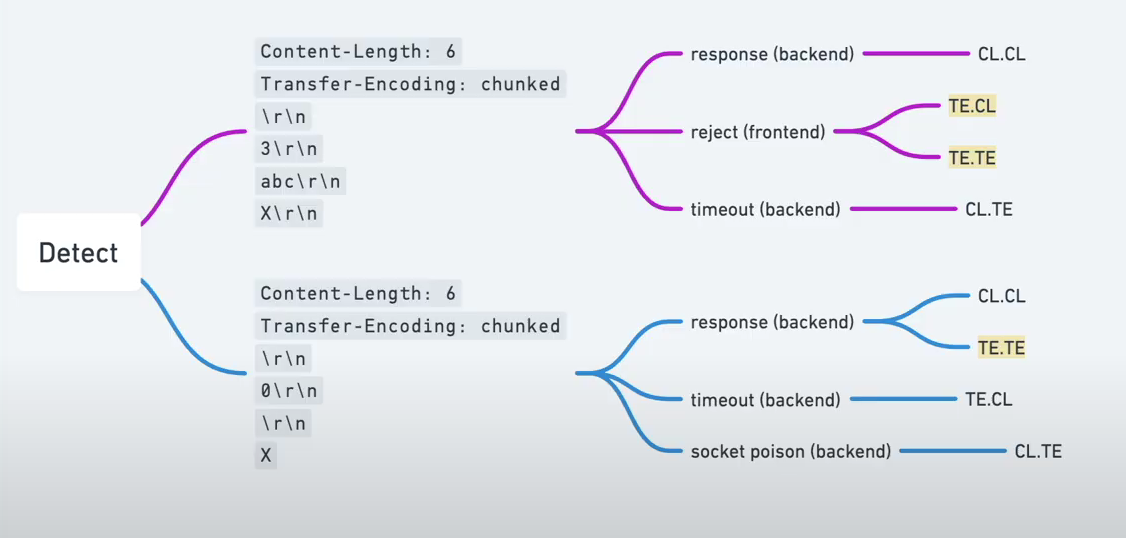
Methodology:

1. Use http request smuggler burp extension to find potentially vulnerable requests
2. Once found, send the request to repeater and configure it accordingly. Change req. method to post, show \r\n characters, and try to downgrade to HTTP 1.
3. Then use the detection payloads to determine the configuration of the servers
4. Confirm the detected vulnerability
5. Exploit

**Detection**

We have to detect the behavior of the front and backend servers. Specifically, we need to see how they behave when Transfer-Encoding (TE) and Content-Length (CL) headers are both present. 3 possible configs: CL.TE, TE.CL, TE.TE

To find these we can send various requests and observe the responses. Chunked encoding works by specifying the number of bytes in hex and at the end of the request ends with 0 following by \r\n\r\n.



\*\* Also note the speed of the requests in weird circumstances to help determine if requests are making it to the backend or not.

In the purple “payload” note that it ends with X, which is an invalid byte specifier as it has no hex value. If this request is rejected the frontend is likely using the transfer encoding header.

The other payload begins with a 0 which marks the end of a chunked encoding.

\*\* if we believe it is TE.TE we will need to reuse the above payloads but we will need to obfuscate the TE header. We need to look for different behaviors using the malformed header with the same payload. Here are some good obfuscation techniques:

-Transfer-Encoding: xchunked

-Transfer-Encoding : chunked

-Transfer-Encoding: chunked

Transfer-Encoding: x

-Transfer-Encoding:[tab]chunked

-[space]Transfer-Encoding: chunked

-X: X[\n]Transfer-Encoding: chunked

-Transfer-Encoding

: chunked

**Confirming Vulnerability:**

Once we determine the server configuration, we can begin to craft exploits. Of course, impact and exact specifications will vary a lot between targets. There are still general rules to follow based on these configurations.

CL.TE:

A screenshot of a computer program

Description automatically generated

TE.CL:

A screenshot of a computer

Description automatically generated

TE.TE: this basically turns into a TE.CL as or CL.TE as one of the servers will be more lenient with taking malformed headers.

**Exploiting Techniques:**

In the smuggled request, we will almost always need to add a content length to it. This content length must be at least 1 more then the body of the smuggled request. If in doubt always put a larger number then necessary. We also will likely need Content-Type header as well. This smuggled request also must end with 0\r\n\r\n (basically an extra new line after the 0). Also note that the first new line (\r\n) before the body is never included when specifying content length either in the smuggled or normal request.

When using chunked encoding the chunk must be specified in hex. So when doing this for a smuggled request, the 0 and the \r\n\r\n are not included in the byte size. We can highlight our smuggled request in a repeater tab and will see in the upper right of the inspector tab how many bytes there are in hex. Example:

A screenshot of a computer

Description automatically generated