HTTP Request Smuggling Attacks

Team Members:-
Taha Adeel Mohammed - CS20BTECH11052
Shambhu Kavir - CS20BTECH11045

Problem Statement

Understanding and mitigating HTTP Request smuggling attacks

Steps involved:-

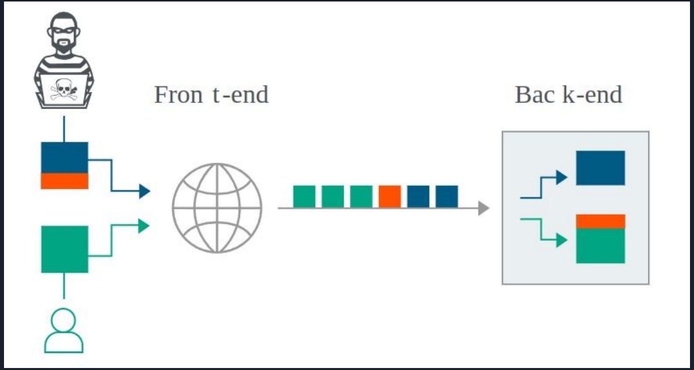
1. Understanding of HTTP Request Smuggling attack
2. Using existing labs/techniques to carry out attacks in a sandbox environment.
3. Simulating the attack and checking vulnerability of servers supporting HTTP 1.1 and 2
4. Deploy mitigation techniques to HRS

What is HTTP Smuggling attack? (Intro)

- **Majority of web applications use HTTP servers between users and the application logic.**
- **Requests from users are sent to a front-end/reverse proxy (load balancer).**
- **The front-end forwards the request to the backend server.**
- **The backend server processes the request and sends the response back.**
- **The reverse proxy forwards the response to the user.**

Vulnerability cause

- Front-end and backend must follow established rules for request processing.
- Ambiguous requests can be exploited by attackers.
- Attackers may send requests that are interpreted differently by the front-end and backend.
- This vulnerability is known as HTTP Request Smuggling.



Ambiguity in Request Interpretation

- HTTP specification offers two ways to interpret the request body: Content-Length header and Transfer-Encoding header.
- Content-Length specifies the length of the body in bytes.
- Transfer-Encoding specifies the encoding used to transmit the body.
-

Ambiguity Exploitation:

- Requests can use both Content-Length and Transfer-Encoding headers simultaneously.
- Attackers exploit this ambiguity to send requests interpreted differently by front-end and backend.
- HTTP specification states that Transfer-Encoding should take precedence when both headers are present.
- However, this solution fails when multiple servers are involved in the request chain.
- Disagreements arise if a server doesn't support Transfer-Encoding or manipulates the header.
- Front-end and backend servers become vulnerable due to disagreements in interpretation.

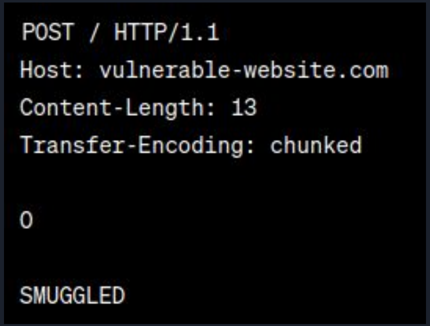
Exploiting HRS Vulnerabilities in HTTP 1.1

- Involves having both TE (Transfer-Encoding) and CL (Content-Length) headers in one request.
- Front-end processes one header while the backend processes the other.
- Three ways to exploit these vulnerabilities.

1) CL.TE

- Front-end processes Content-Length header.
- Backend processes Transfer-Encoding header.
- Front-end reads the specified Content-Length bytes of the body.
- Backend treats the remaining bytes as a new request.

Example:

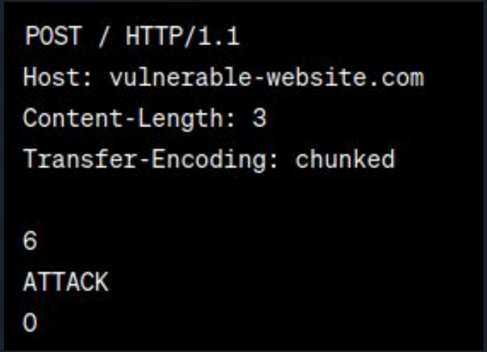


```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 13
Transfer-Encoding: chunked

0

SMUGGLED
```

2) TE.CL:

- Front-end processes Transfer-Encoding header.
- Backend processes Content-Length header.
- Front-end reads the specified chunked bytes of the body.
- Backend treats the remaining bytes as a new request.

Example:

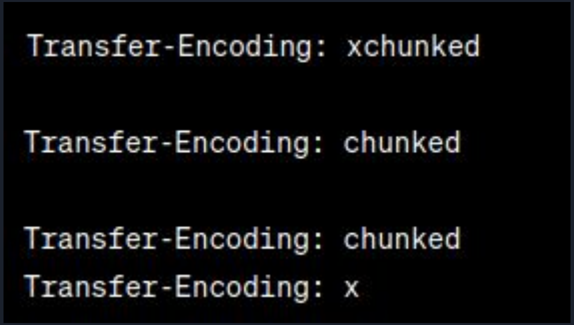


```
POST / HTTP/1.1
Host: vulnerable-website.com
Content-Length: 3
Transfer-Encoding: chunked

6
ATTACK
0
```

3) TE.TE

- Both front-end and backend support Transfer-Encoding (TE).
- One of them is manipulated to prevent processing one of the headers.
- Request smuggling is achieved by using the second header.

Example:



```
Transfer-Encoding: xchunked

Transfer-Encoding: chunked

Transfer-Encoding: chunked
Transfer-Encoding: x
```

Detecting HRS vulnerability in HTTP 1.1

Monitor for inconsistencies:

- Track requests and responses for front-end and backend servers.
- Look for inconsistencies in header interpretation.

Analyze request and response headers:

- Check for Content-Length and Transfer-Encoding headers.
- Identify discrepancies or unusual header combinations.

Perform vulnerability scanning:

- Use specialized tools for detecting HTTP request smuggling vulnerabilities.

Employ web application firewalls (WAFs):

- Use WAFs for protection against HTTP request smuggling attacks.

HTTP 2

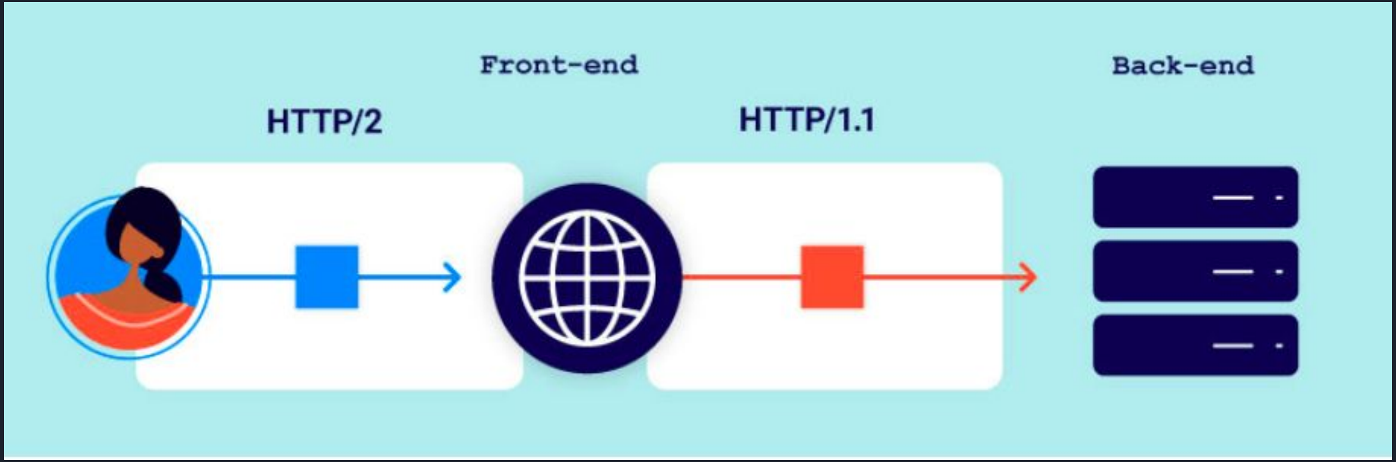
HTTP/2 is the successor of HTTP 1.1.

It was developed to improve the performance of HTTP 1.1, particularly in the areas of latency, network utilization and perceived user experience.

The main difference between HTTP 1.1 and HTTP 2 is the way the data is transmitted between the client and the server.

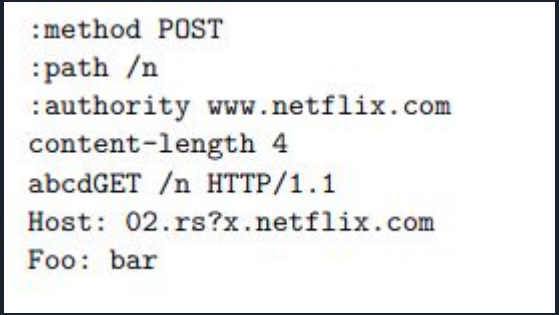
In HTTP 1.1 which uses a simple request response model, HTTP 2 uses a binary protocol.

Downgrading



In this process, the reverse proxy interacts with the client in HTTP 2 but forwards the requests to the backend in HTTP 1.1. This transition in protocol makes it possible to smuggle requests.

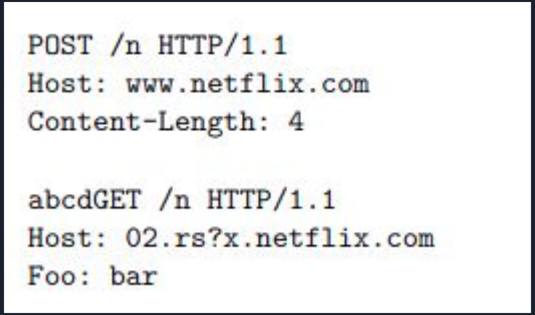
Illustration of HTTP 2CL



```
:method POST
:path /n
:authority www.netflix.com
content-length 4
abcdGET /n HTTP/1.1
Host: 02.rs?x.netflix.com
Foo: bar
```

HTTP 2 request

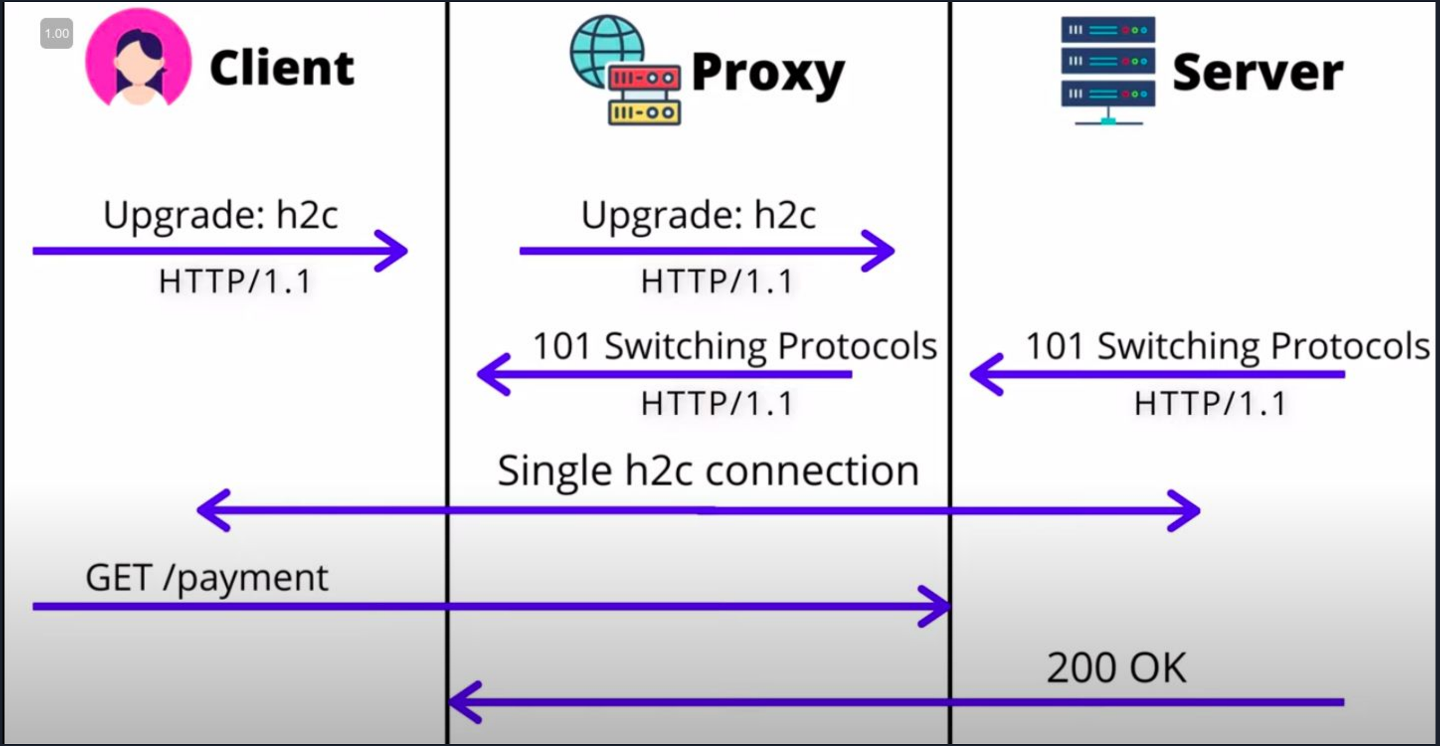
downgrade



```
POST /n HTTP/1.1
Host: www.netflix.com
Content-Length: 4

abcdGET /n HTTP/1.1
Host: 02.rs?x.netflix.com
Foo: bar
```

HTTP 1.1 request

h2c smuggling



Mitigation Techniques

1. Prioritizing TE over CL headers:
- Enforce Transfer-Encoding (TE) over Content-Length (CL) when both headers are present.
- TE covers both static and dynamic requests, while CL only covers static requests.
2. Securing logged HTTP Traffic:
- Secure HTTP logs and restrict access to authorized personnel.
- Prevent attackers from exploiting vulnerabilities by accessing unintended parts of requests.
3. Rejecting requests with double headers:
- Reject requests with TE.CL, CL.TE, and TE.TE headers.
- Respond with a 400 Bad Request to avoid ambiguity between front-end and backend processing.

Mitigation(Contd.)

4. Using Vulnerability Scanner:
- Employ vulnerability scanners to detect poorly configured HTTP requests.
- Regularly test web applications to reduce the risk of attacks.

5. Enforcing HTTP 2:
- Use HTTP 2 in both front-end and back-end systems.
- HTTP 2 prevents ambiguity in parsing requests and helps prevent HRS attacks.

HRS in HTTP 3

HTTP/3 Overview:

- HTTP/3 is a new protocol based on QUIC, designed for security, low-latency, and multiplexing.
- Developed to address vulnerabilities present in HTTP 1 and 2, including HRS attacks.

Advancements in HTTP/3:

- Separation of Requests and Responses:
  - Individual requests and responses are separated into separate streams.
  - Prevents ambiguity in parsing requests and responses.
- QUIC Datagram Framing:
  - HTTP/3 employs QUIC Datagram, a self-contained packet carrying a single HTTP/3 frame.
  - Each frame is transmitted independently, making it difficult for attackers to intercept or modify data in transit.
- Explicit Stream Multiplexing:
  - Unique stream identifiers explicitly map each individual request/response in HTTP/3.
  - Ensures unambiguous association between requests/responses and streams.
  - Alleviates confusion and misinterpretation between the front-end and backend.

HTTP/3 and HRS Prevention:

- HTTP/3 introduces these advancements to mitigate HRS attacks and enhance security.
- Designed with security in mind and to prevent common HRS vulnerabilities found in HTTP 1 and 2.

Implementing HTTP/3 can significantly improve the security and reliability of web applications by addressing HRS vulnerabilities and enhancing communication protocols.

Conclusion

- HTTP request is a serious vulnerability in the HTTP protocol that can be exploited to launch on web servers.
- Through hands on practice and research, we have gained deep understanding of how HRS works and different techniques that we can use to exploit it.
- It is important for web developers and security professionals to be aware of this vulnerability and take measures to prevent it.

Contributions of Team Members

1. **Taha :- Simulating attacks in HTTP 1.1 in port swigger labs, HTTP smuggling vulnerability detection methods, ways attacks can be carried out to exploit the vulnerability, researching mitigation techniques, setting up reverse proxy in sandbox environment for attempting mitigation techniques, report writing, presentation**
2. **Shambhu:- Simulating detection methods and attacks in HTTP 1.1 and 2 in a docker environment, including downgrading and h2c, exploring methods to mitigate attacks in HTTP 2, improvements in HTTP 3, report writing, presentation**