SECURITYBOAT

Frontline Of Your Business

HTTP Request Smuggling HANDBOOK

www.securityboat.net

SECURITYBOAT
Frontline Of Your Business

Table of Contents

1. Fundamental Concepts

1.1 What is an HTTP Request

An HTTP request is a communication initiated by a client, typically a web browser, to request a particular action or resource from a server. It encompasses details like the request method (GET, POST), the target URL, and optional data. The server responds by processing the request and sending back an HTTP response, providing the requested information, or confirming the action.

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
```

```
POST /form.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Content-Length: 0
```

1.2 What is CL and TE

Content-Length (CL): Content-Length is an HTTP request header that specifies the size of the request body in bytes. It helps the server determine the end of the request content, enabling proper parsing and processing of the request.

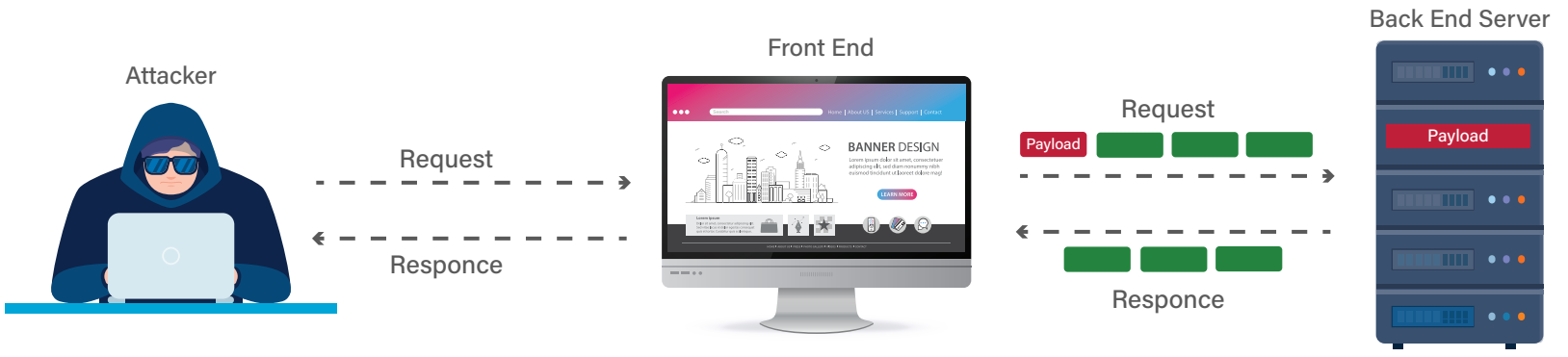
Transfer-Encoding (TE): Transfer-Encoding is an HTTP request header that indicates the encoding mechanisms applied to the message body. It informs the server about the transformation applied to the request content, such as chunked encoding or compression, allowing for proper decoding on the server side.

1.3 How Requests are Processed at the Front-End and Back-End

The front end of a web application is what users interact with directly. When you perform actions like clicking buttons or submitting forms, the front end handles your requests, carries out client-side tasks (such as form validation), and then forwards the necessary data to the back end for additional processing. On the other hand, the back end works behind the scenes, receiving requests from the front end. It manages server-side operations, including database tasks and business rules, and generates responses sent back to the front end. The back end is responsible for critical aspects like data storage, ensuring security measures, and ensuring the overall functionality and performance of the web application.

2. What is HTTP Request Smuggling

HTTP Request Smuggling is a vulnerability where an attacker manipulates the way a website handles sequences of HTTP requests. By exploiting inconsistencies in how various parts of the web server process requests, such as a backend server and a front-end proxy, the attacker can trick the system into processing requests differently than intended. This vulnerability is serious because it can bypass security measures, access confidential information without permission, and directly compromise other users of the application.

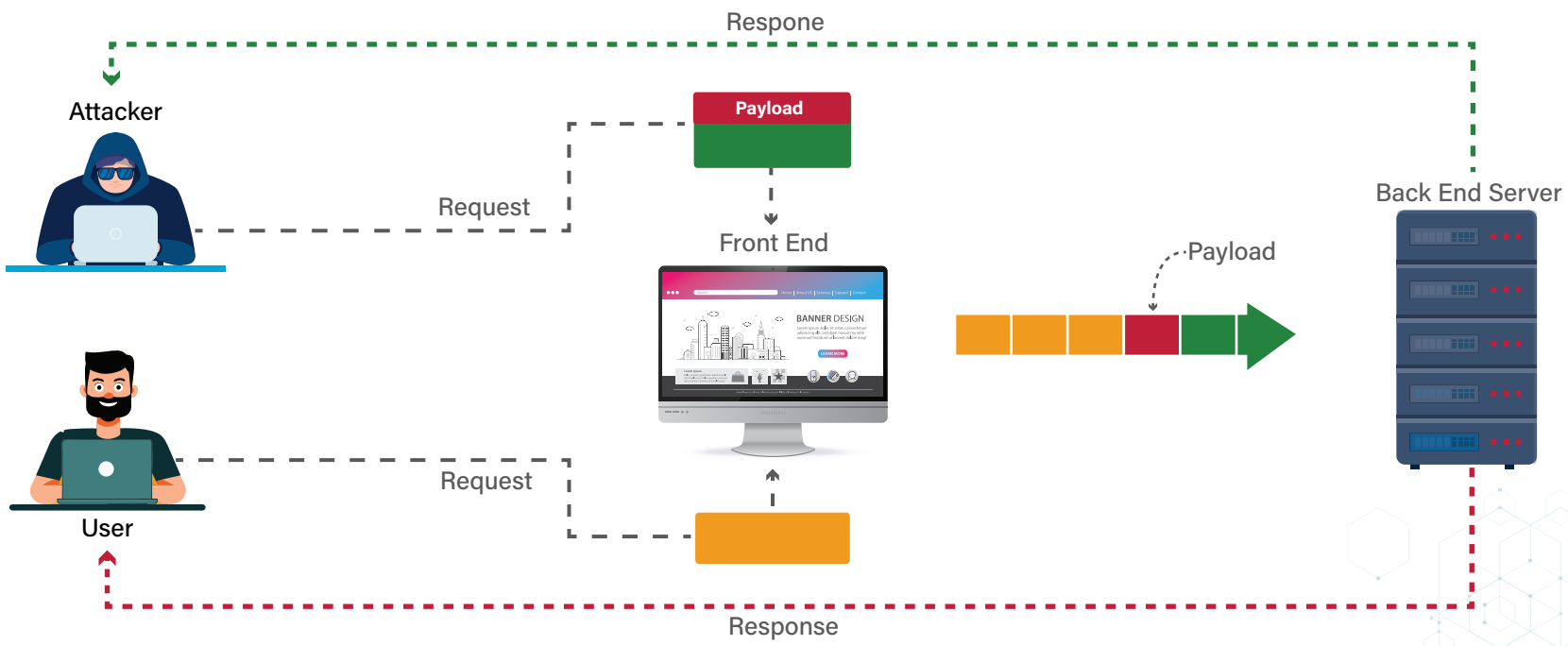


3. How HTTP Request Smuggling Works

HTTP Request Smuggling exploits the way servers manage user requests in a web application. As users interact with the application, their actions trigger HTTP requests. Attackers manipulate how servers interpret these requests, focusing on headers like Content-Length and Transfer-Encoding.

The attack begins with the attacker crafting a request to mislead the server. By manipulating headers to varying lengths or encoding methods, the front-end and back-end servers interpret the request differently. This inconsistency lets the attacker insert a second request within the first, exploiting confusion between servers.

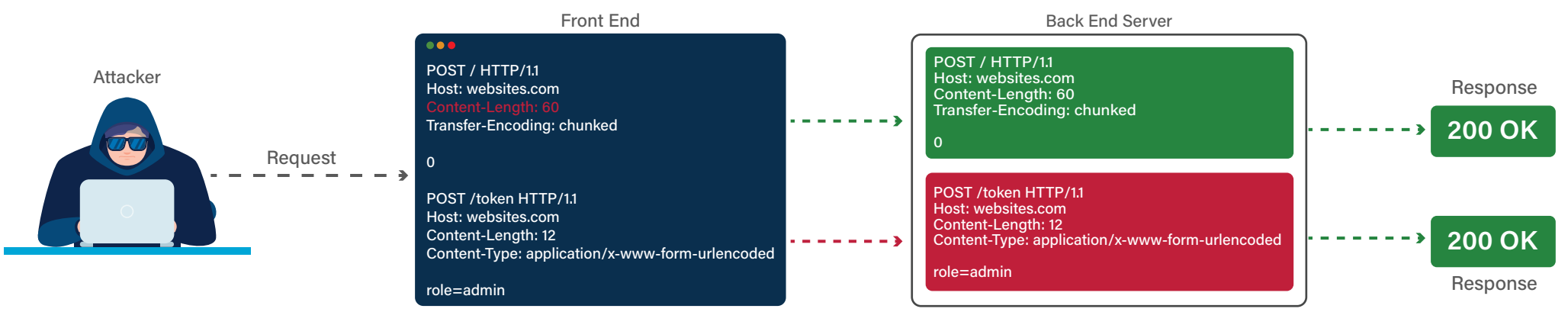
The attacker adds headers like Transfer-Encoding to further manipulate the process. This creates a scenario where the front-end server might view the payload as a single request, while the back-end server interprets it differently. This inconsistency could leave parts of the payload unprocessed, opening vulnerabilities for unauthorized access or malicious activities.



4. Attack Techniques for HTTP Request Smuggling

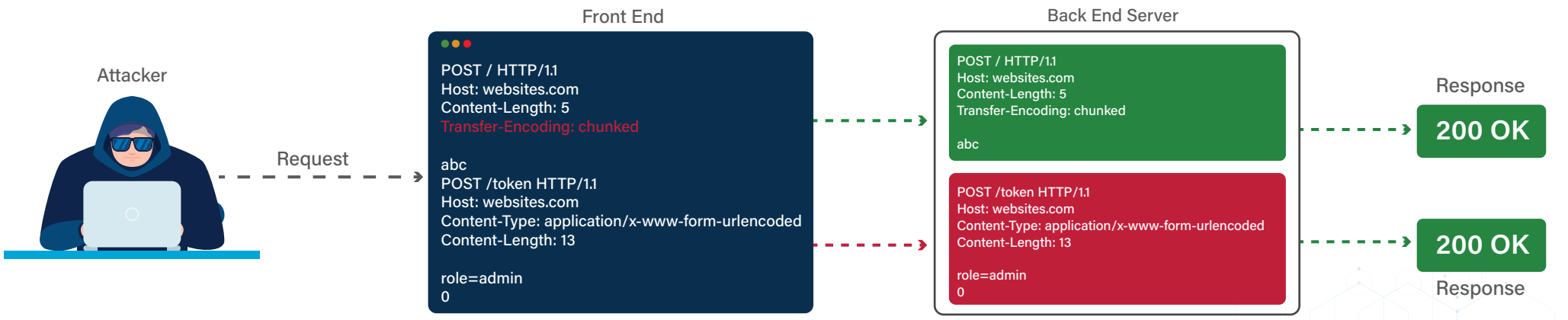
4.1 CL.TE

In a CL.TE (Content-Length and Transfer-Encoding) attack, the attacker manipulates how a web server interprets requests by exploiting differences in handling Content-Length and Transfer-Encoding headers between the front-end and back-end servers. The attacker crafts a request with conflicting headers, specifies different lengths, or encoding methods. The front-end server, relying on Content-Length, processes the request based on its rules, while the back-end server, focused on Transfer-Encoding, interprets the same request differently. This inconsistency creates an opportunity for the attacker to embed a second request within the first. The front-end may treat the entire payload as one request, while the back end sees it differently, potentially leaving parts unprocessed. This manipulation exploits the miscommunication between servers, allowing the attacker to mislead both into handling requests inconsistently.



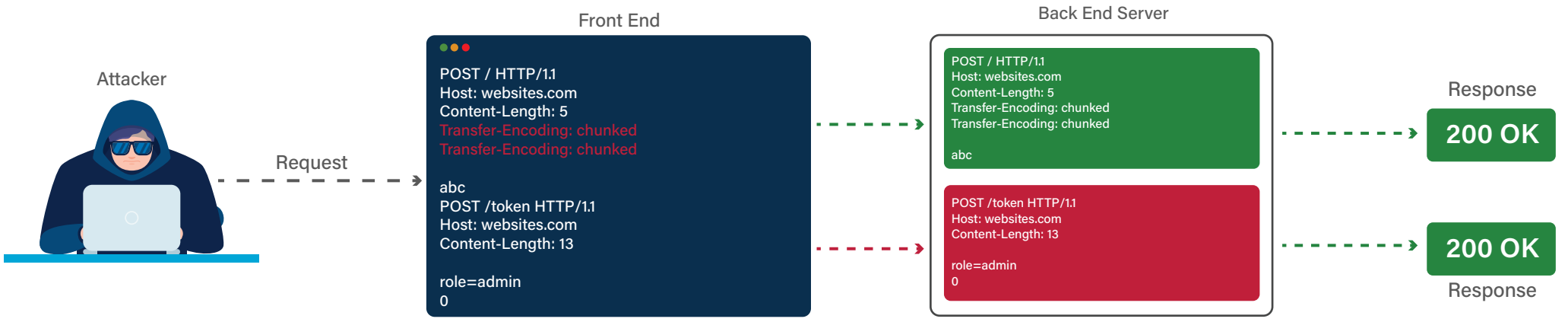
4.2 TE.CL

In a TE.CL (Transfer-Encoding and Content-Length) attack, the attacker exploits the difference between front-end and back-end servers in processing Transfer-Encoding (TE) and Content-Length (CL) headers. First, the attacker sends a request with conflicting TE and CL headers, creating confusion. The front-end, which prioritizes TE, interprets the request differently than the back end, which relies on CL. This inconsistency allows the attacker to smuggle a second request within the first one. The front-end may treat the payload as a single request, while the back end sees it as two, potentially leaving parts unprocessed. This manipulation leverages the misalignment in header processing, creating a misleading flow and lead the way for unauthorized actions.



4.3 TE.TE

In a TE.TE HTTP request smuggling attack, the attacker manipulates the Transfer-Encoding header to deceive the front-end and back-end servers. Initially, the attacker crafts a request with a Transfer-Encoding header set to "chunked." The front-end server interprets this header, expecting chunked encoding. However, the attacker cleverly inserts a second Transfer-Encoding header with the value "chunked" in the same request. As the request moves to the back-end server, it interprets the second Transfer-Encoding header and treats the request as chunked again, despite already processing it as such. This discrepancy allows the attacker to smuggle in a second request within the first one, exploiting the confusion between front-end and back-end servers. The attacker gains an opportunity to execute unauthorized actions or access sensitive resources by manipulating the Transfer-Encoding header in this way.



5. Tools for HTTP Request Smuggling

5.1 HTTP Request Smuggling

HTTP Request Smuggling is Burp Suite Extension, that can identify HTTP Request Smuggling vulnerabilities by analysing web application behaviour.
https://github.com/portswigger/http-request-smuggler

5.2 GitHub- Smuggler

Smuggler is an open-source tool designed for identifying and exploiting HTTP Smuggling vulnerabilities.
https://github.com/defparam/smuggler

6. Impact of HTTP Request Smuggling

HTTP Request Smuggling can lead to serious consequences, affecting the security and integrity of applications. Let's explore some of the potential impacts it can have.

6.1 Unauthorized Access

HTTP Request Smuggling can be exploited to bypass authentication mechanisms, enabling attackers to gain unauthorized access to restricted areas of the web application. By manipulating the request headers, attackers can mislead the server into processing requests incorrectly, granting them access to sensitive resources or functionalities.

6.2 Session Hijacking

Attackers can leverage HTTP Request Smuggling to hijack user sessions, assuming control of their accounts. By manipulating request headers and payloads, attackers can tamper with session identifiers, enabling them to impersonate legitimate users and access sensitive information or perform unauthorized actions within their accounts.

6.3 Server-Side Request Forgery (SSRF)

HTTP Request Smuggling can be utilized to perform Server-Side Request Forgery attacks, allowing attackers to send requests to internal or external resources from a trusted server. By manipulating request headers and payloads, attackers can trick the server into making unauthorized requests on their behalf, potentially accessing sensitive information, or carrying out malicious actions, such as retrieving confidential data or initiating network scans.

6.4 Cross-Site Scripting (XSS)

HTTP Request Smuggling can exacerbate the impact of Cross-Site Scripting (XSS) attacks by facilitating the injection of malicious payloads into unsuspecting web pages. By exploiting vulnerabilities in request handling, attackers can smuggle XSS payloads into requests processed by the server, leading to the execution of malicious scripts in users' browsers. This can result in data theft, session hijacking, or other malicious activities.

6.5 Remote Code Execution (RCE)

HTTP Request Smuggling can be utilized by attackers to execute their code on the server, extending beyond session hijacking. This potential for Remote Code Execution (RCE) allows malicious actors to assume complete control of the system, enabling them to manipulate server resources, deploy malware, and potentially escalate their privileges. This introduces significant threats to the overall integrity and security of the web application.

7. Prevention

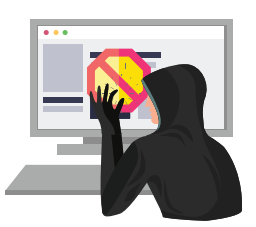
7.1 Implement Standardized Header Parsing

Use consistent and standardized methods for parsing and interpreting HTTP headers across both frontend and backend servers to prevent inconsistencies that attackers may exploit.

7.2 Strict Content-Length Verification

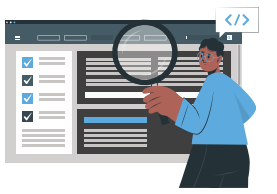
Enforce strict validation of Content-Length headers to ensure that the length specified matches the actual content length of the request body, preventing discrepancies that could be exploited by attackers.

7.3 Prevent Header Manipulation



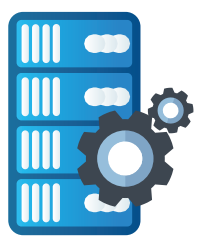
Implement measures to detect and block attempts to manipulate HTTP headers, including Content-Length and Transfer-Encoding, to prevent attackers from tricking servers into processing requests incorrectly.

7.4 Content Inspection



Perform thorough inspection of incoming requests to identify and remove any malicious or suspicious content, including embedded requests or payloads that could be used in HTTP request smuggling attacks.

7.5 Request Buffering



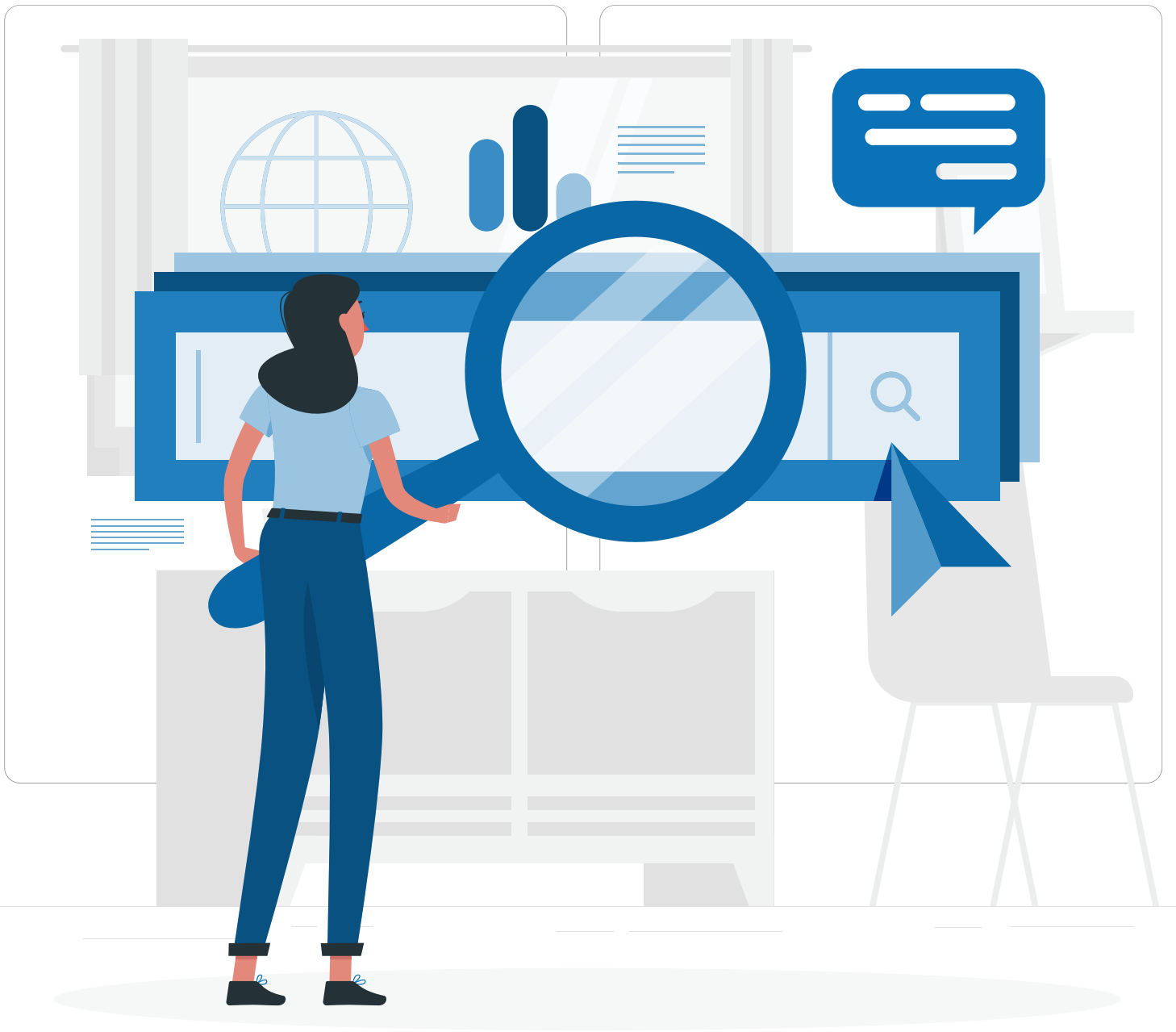
Enforce rate limiting and throttling mechanisms to control the rate of incoming Implement request buffering mechanisms to hold incoming requests until they are fully received and validated, preventing partial processing that could be exploited in smuggling attacks.

7.6 HTTP/2 Protocol Implementation

Implement the HTTP/2 protocol, which has built-in safeguards against HTTP request smuggling by using binary framing, thus reducing the likelihood of parsing inconsistencies.

8. Additional Resources and References

- https://portswigger.net/web-security/request-smuggling#what-is-http-request-smuggling
- https://capec.mitre.org/data/definitions/33.html
- https://workbook.securityboat.net/resources/web-app-pentest/http-request-smuggling
- https://book.hacktricks.xyz/pentesting-web/http-request-smuggling

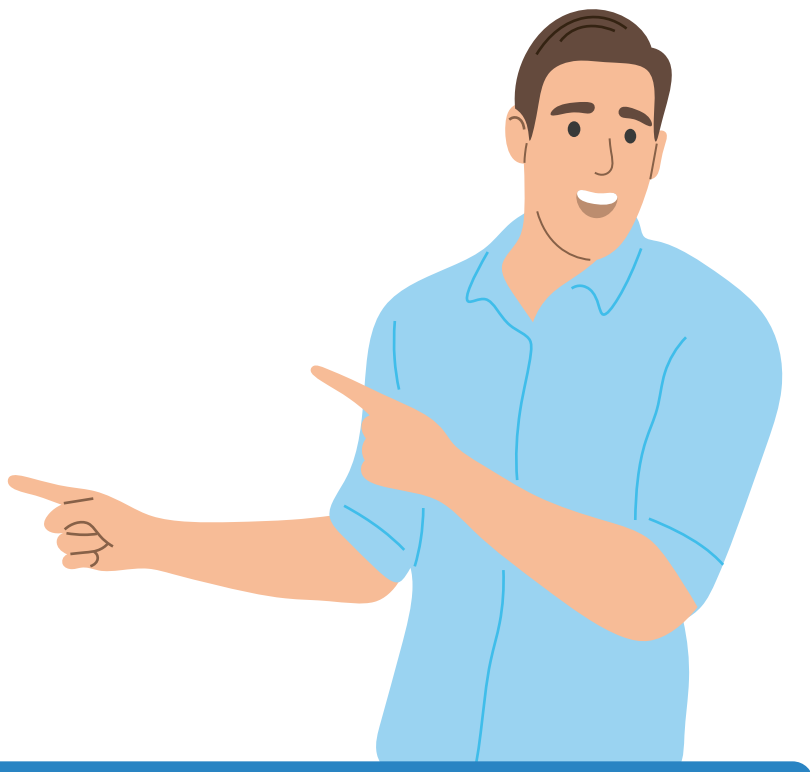


SECURITYBOAT

Frontline Of Your Business

HTTP REQUEST SMUGGLING

HANDBOOK



Scan QR Code to Download Handbook

www.securityboat.net