baby WAFfles order

Platform	HTB
Operating System	Web-CTF
Tags	LFI XXE

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

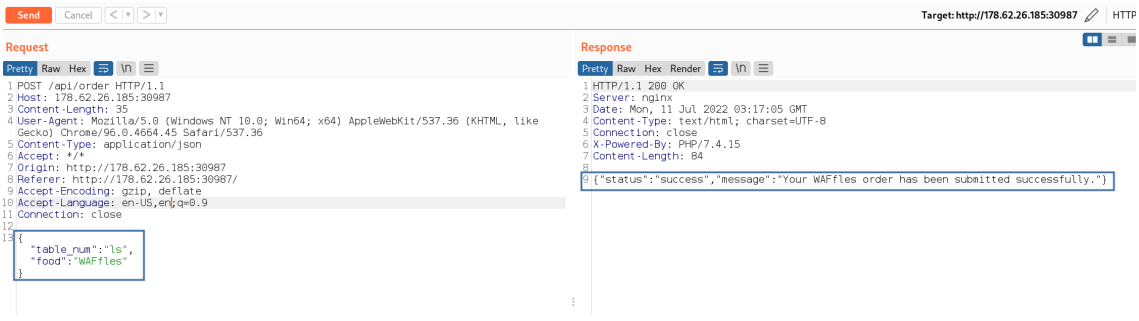
- ▼ `OrderController.php` file

```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



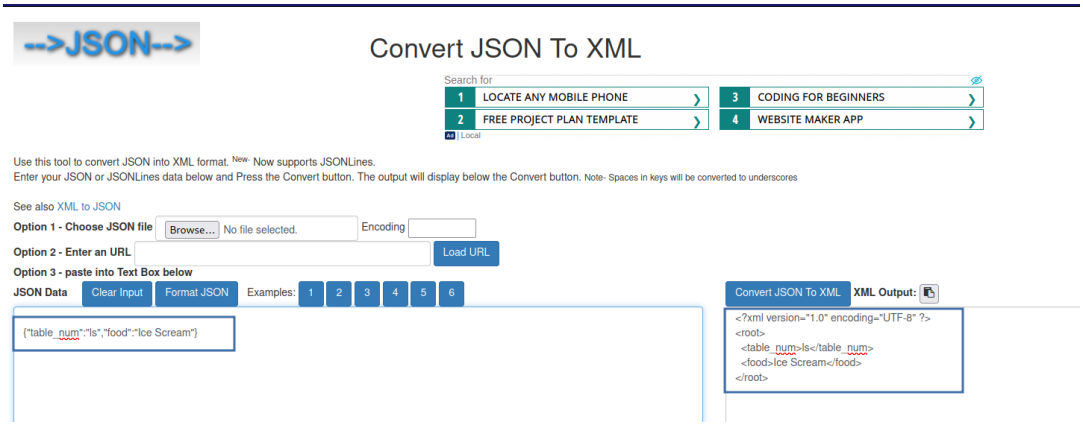
▼ Using this JSON to XML convertor to get an XML string

▼ XML

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

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

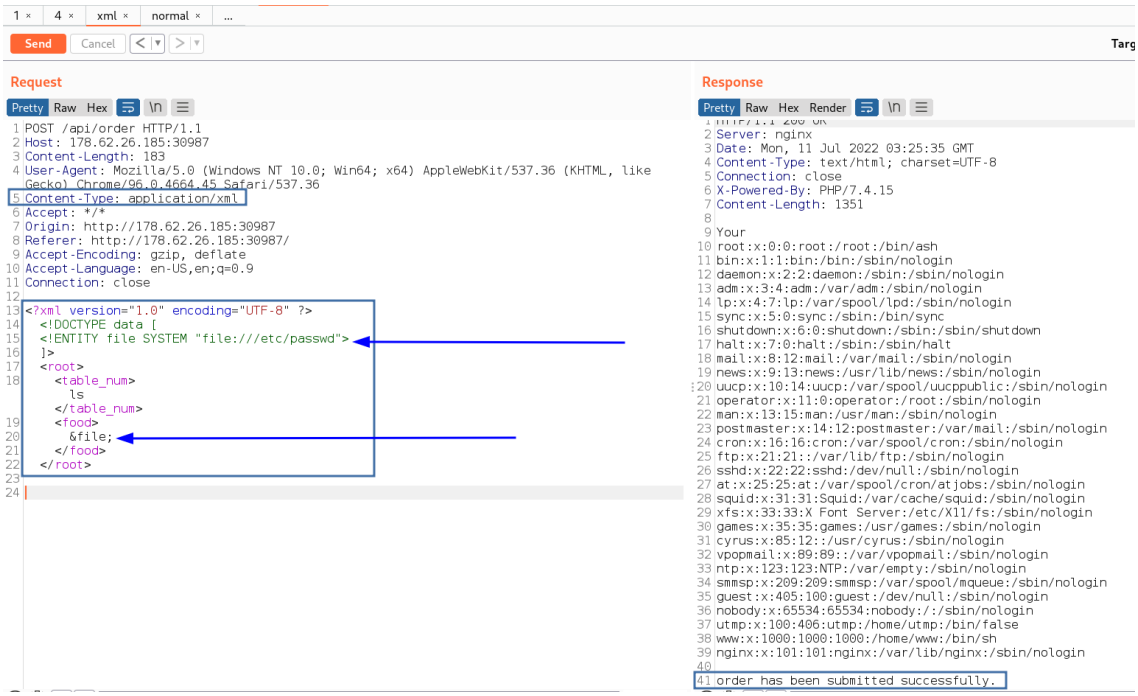
▼ Screenshot



▼ Validating the XML string working

▼ 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



▼ 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

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