Rat CTF

USER - MattP

Box - https://tryhackme.com/room/ratctf

The box was hosted on Try Hack Me, after deploying the box as with all I started off with running an nmap:

```
clueless@kali:~/rat$ sudo nmap -sSV 10.10.174.150 | tee rat.txt
Starting Nmap 7.80 ( https://nmap.org ) at 2020-09-05 11:55 EDT
Nmap scan report for 10.10.174.150
Host is up (0.021s latency).
Not shown: 997 closed ports
PORT     STATE    SERVICE VERSION
22/tcp   open     ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
80/tcp   open     http    Node.js Express framework
3000/tcp filtered ppp
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/s
Nmap done: 1 IP address (1 host up) scanned in 11.04 seconds
```

Using the following:

-sS - TCP SYN/Connect
V - Probe open ports to determine service/version info

Straight away we can see the following are open:

22 SSH Port, as we have no usernames or passwords we give this a swerve to start with
80 HTTP standard web port

Opening up http://<IP> we are presented with a simple website with a few options to look at – Home – Login – Register

Whilst we have a look around the website, I fired up **gobuster** and checked to see what (if any) directory's are on the site.

Using **gobuster** with:

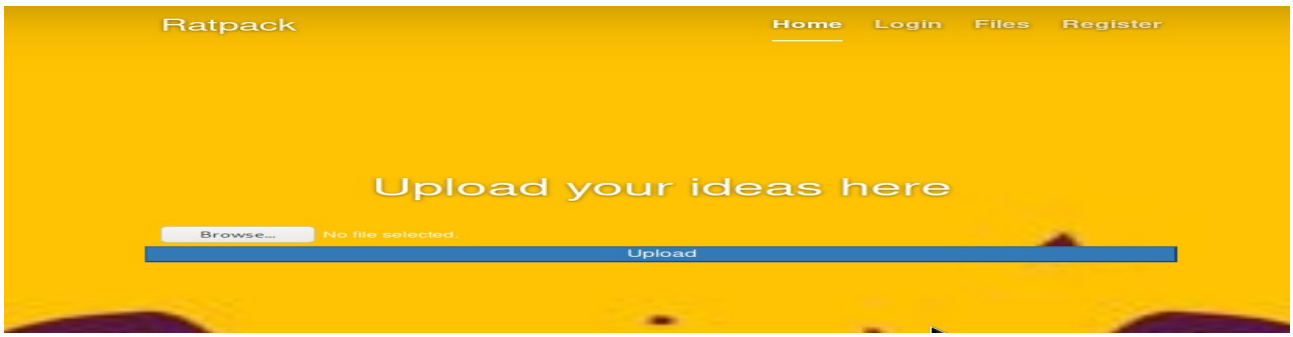
dir Uses directory/file bruteforcing mode
-u URL
-w Wordlist, in this case common.txt from dirb

```
clueless@kali:~/rat$ cat buster.txt
===============================================================
Gobuster v3.0.1
by OJ Reeves (@TheColonial) & Christian Mehlmauer (@_FireFart_)
===============================================================
[+] Url:            http://10.10.174.150
[+] Threads:        10
[+] Wordlist:       /usr/share/wordlists/dirb/common.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/09/05 11:57:08 Starting gobuster
===============================================================
/components (Status: 301)
/css (Status: 301)
/delete (Status: 200)
/document (Status: 200)
/files (Status: 200)
/Images (Status: 301)
/images (Status: 301)
/login (Status: 200)
/Login (Status: 200)
/register (Status: 200)
/upload (Status: 200)
/views (Status: 301)
===============================================================
2020/09/05 11:57:18 Finished
===============================================================
clueless@kali:~/rat$
```

Once this is done, we go through the directories – some are only available if we are logged in, others seem like duplicates.

Back to the website, we register a dummy account – noting there are a few rules on the `length` for username and password. Once logged in, we have a couple more `links` available (Uploads/Files).

A good place to start would always be the Upload area (easy way to get a file on remote client/server).



I tried to upload a few test files, it takes .txt .docx and doc ok, but has does not accept other file types (.php, .sh)

Once a file is uploaded it can be seen in the `Files` area, which points to it being in the /upload directory:

A few hints have pointed us towards XML External Entity Injection (XXE)

Link - https://doddsecurity.com/312/xml-external-entity-injection-xxe-in-opencats-applicant-tracking-system/

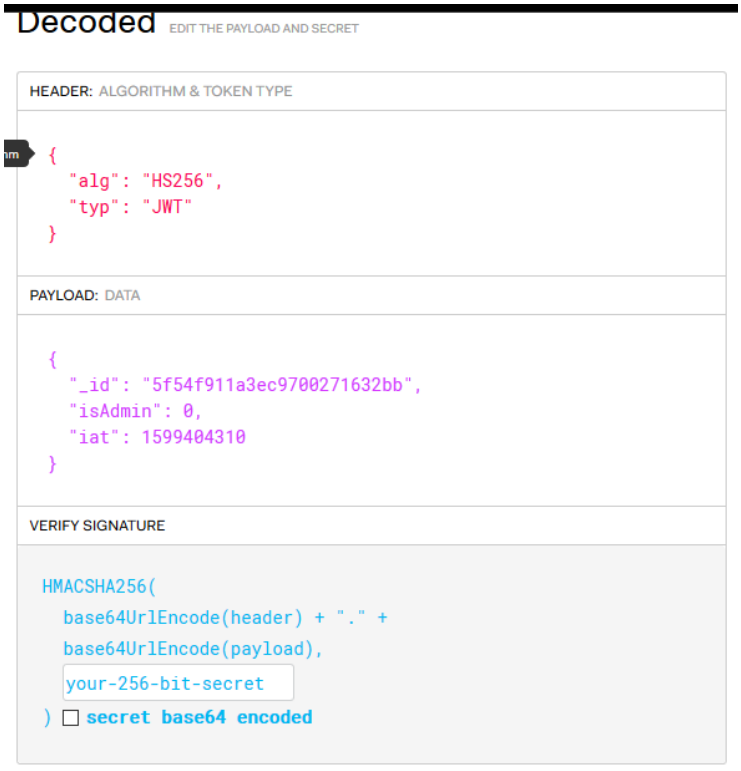
Following the link above, I managed to read some files on the box - */ etc /passwd*

<!DOCTYPE test [<!ENTITY example SYSTEM "/etc/passwd">]>

But this did not give us anything to go on. Looking around the site, when we log in it uses a JSON Web Token as the cookie:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZjU0ZjkxMWEzZWM5NzAwMjcxNjMyYmIiLCJpc0FkbWluIjowLCJpYXQiOjE1OTk0MDQzMTB9.Z7ywRgiYwnRnHMjuYTZ1RkoeaSn_iv_AmpyFlXri4Mk

When using https://jwt.io to decoded this, we can see:



We know that the JWT consists of 3 parts (seperated by a dot)

- **Header** Typically consisting of 2 parts, the token type which is **JWT**, and the algorithm being used which in this case is **HS256**.

- **Payload** This contains the *claims,* which can be either private or public and are generally taken from https://www.iana.org/assignments/jwt/jwt.xhtml , in this case we have **_id**, **isadmin** and **iat** looks like we need to change the ***isadmin*** to 1 (yes).

- Signature To create the signature part you have to take the encoded header, the encoded payload, **a secret** and the algorithm – we only have 3 of these – no **secret**.

First off I just tried changing the **is_admin** to 1 and sending that as our cookie but the server returned `invalid token`, then I tried to do it with `none` algorithm route but again this did not work.

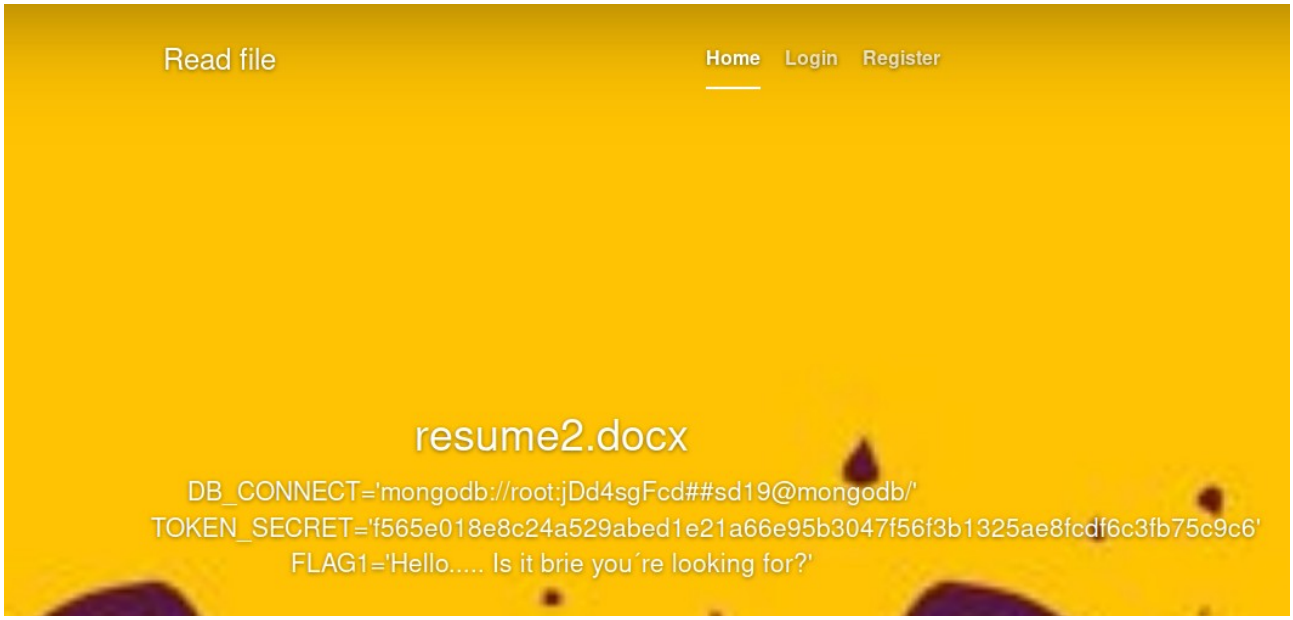
So in order to change the **is_admin** to **yes** (1) and correctly verify this by signature we need to find the **secret** – so its back to XXE but this time with an idea of what we need to find.

We know and have it confirmed by **hints** that this is a node.js and that uploaded files are in **usr/src/app/upload**

With this in mind a quick google search (and another hint) we find that the **.env** file in node.js allows you to set up and access environment variables with node.js straight out of the box (https://www.twilio.com/blog/working-with-environment-variables-in-node-js-html) armed with these details, we can amend the XXE payload to:

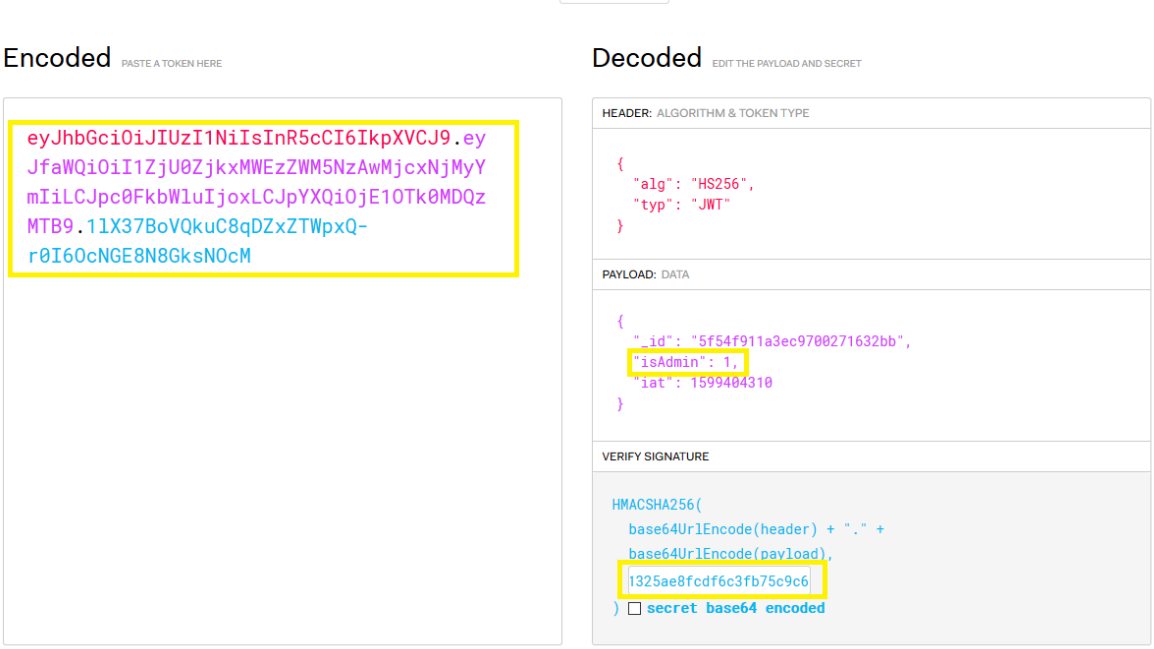
<!DOCTYPE test [<!ENTITY example SYSTEM "/usr/src/app/.env">]>

upload this as a DOCX file and then when we open it, we get:



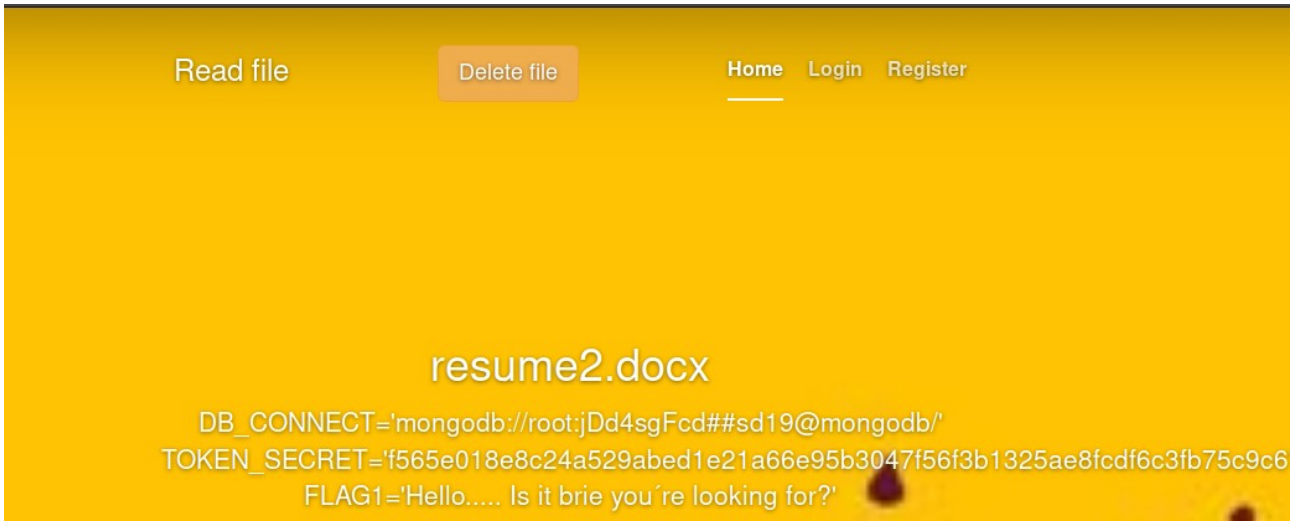
This gives us **DB_CONNECT** (nothing yet on this), **TOKEN_SECRET** exactly what we are looking for with the JWT and **FLAG 1** – Flag 1 doesn't give us any points, its just a marker to let us know we are on the right track!!!.

Armed with the **TOKEN_SECRET** we can re-encode the JWT with the **isadmin** set as 1 using the **secret** to ensure the signature is correct.



With the new JWT, I amended the cookie within development tools, although I could see no change immediately I knew it had worked – as early without the secret key it returned an error `invalid token` this time we just get the page back.

Another look around the website, we now have another button `delete` available when we open a file we have uploaded:



This does exactly as it says on the `tin` it deletes the file. A bit of thinking and digging on this leads us to **Blind Command Injection** it must be running a `command` to delete the file, so lets try and play around with this (thanks to another hint).

Unfortunately this is as far as I got in regards to progressing on the task (but not on the learning curve), not due to time but more I could not get a reverse shell to drop.

I think I had the right idea, just lacked the skillset to execute.

The idea was simple, upload a reverse shell payload with an altered file extension (to bypass the upload restriction), change the file extension or run the file by command injection on the `delete` command.

First off I used a **bash** reverse shell payload from https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md

saved it as a **.txt** in order to upload it, I then figured out (with assistance) that I could use the following to re-name to file

http://<IP>/delete?file=mount.txt%0A(cd+usr%0Acd+app%0Acd+src%0Acd+upload%0Acp+**bash.txt bash.sh**%0A)

With a listener open on my machine (nv -lvn <port>) I tried to open the file, but nothing happened, I then tried to make it executable (chmod +x) and open it, but again nothing.

With hints, I attempted to create a payload with **msfvenom** and also **PHP** etc, upload them and attempt to run them, but I just could not get the shell to drop.