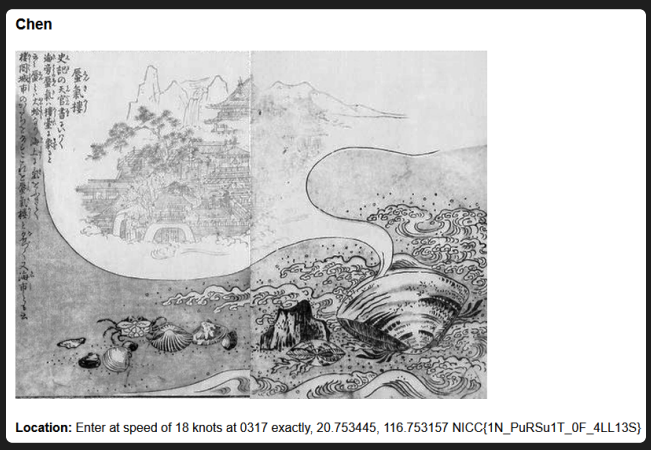
Web

Cryptid Hunters

- one page has a "join" function but this isn't OSINT and there's nothing else on the page + no source code so leave that aside for now
- The login page though could be something
- logging in with `user: admin; password: password` leads to an
  - `Error: Unsafe Input detected` which is interesting cuz we didn't use a payload
  - using the string "password" triggers it for.... some reason(**irrelevant in the end**)
- nothing else was interesting so moved over to Burpsuite which shows some interesting headers in the response:
  - It has the `X-Frame-Options: DENY` header which combats clickjacking
    - makes it some HTML elements like `<iframe>` can't be rendered
  - `X-XSS-Protection: 1; mode-block` header for obviously protecting against XSS attacks
  - `Content-Security-Policy: default-src 'self'` header for an extra layer of protection against XSS and similar cross-site attacks
  - `Referrer-Policy: no-referrer` prevents referrer headers being sent elsewhere
  - also no cache poisoning can be done according to numerous headers
- Suffice it to say I don't think XSS is the solution here particularly as none of the fields jump out as fitting the format.
- And without the source whether it's a SSRF or SSTI attack is also up in the air so it's probably something else
- Since it's an "easy" challenge let's try logging in as just admin with a SQL injection
- Just used `user: admin` but for password I went down through this SQL Payloads List and eventually you get a hit
- Flag is in the page you unlock



- **Flag:** NICC{1N_PuRSu1T_0F_4LL13S}

Paranormal-Pictures

- This one comes with source code for the website with a basic check for if you try to get the flag
-

```
@app.route('/flag')
def flag():
    if request.remote_addr == '::ffff:127.0.0.1' or request.remote_addr == '::1':
        return render_template('flag.html', FLAG=os.environ.get("FLAG"))

    else:
        return render_template('alarm.html'), 403
```

- Try to add an `X-Forwarded-For` header into our burpsuite-repeater and it leads to the alarm.html

```
GET /flag HTTP/1.1
Host: paranormal-picture.niccgetsspooky.xyz
X-Forwarded-For: 0:0:127.0.0.1
Accept-Language: en-US,en;q=0.9
```

- It interestingly has this though in the base page

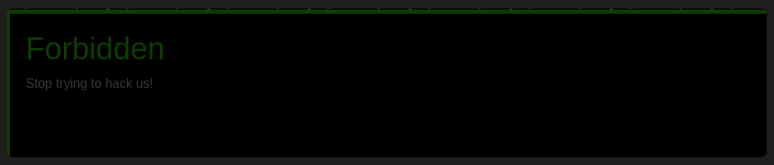
```python
@app.route('/', methods=['GET', 'POST'])
def index():

    if request.method == 'POST':
        url = request.form['url']
        try:
            result = verifyBlog(url)
            if not result:
                return render_template('index.html', error=f"Please submit a blog!")
        except:
            return render_template('index.html', error=f"Please submit a blog!")

        r = requests.get(url)

        return render_template('index.html', result=r.text)
    return render_template('index.html')
```

- If you can pass the check then it will fetch the URL and render it
- pass it something random like `wikipedia/real/cryptid/blog/666/.org/` and you can see rendered gibberish
- First thought was can I use it to render "flag.html" for me?
  - `payload: http://paranormal-picture.niccgetsspooky.xyz/real/cryptid/blog/666/.org/../../../../../flag`
  - This gets it past the check but then uses multiple `/../` to escape out of the fake directories I gave it
  - result:



  - Trying to get to the raw file with this also didn't get me the flag
    - `payload: http://paranormal-picture.niccgetsspooky.xyz/real/cryptid/blog/666/.org/../../../../../templates/flag.html (notice the "/templates/")
- Solution:
  - The above idea is correct, it's all correct but we're simply querying it wrong as is when it tries to **fetch()** the webpage it's doing it as a "civilian" so to speak
  - We need to simply change it so that it does it as itself as the localhost
  - Thankfully the source files tell us which port it's running on

```python
if __name__ == '__main__':
    app.run(host="::", port=80, threaded=True)
```

  - So just change the payload to
    - `http://localhost:80/real/cryptid/blog/666/.org/../../../../../flag`
  - **Flag**: NICC{tHe_crYptIds_aRe_walting_t0_sTrike}