https://cyberjutsu.io
contact@cyberjutsu.io
+(84) 0338 836 830

CyberJutsu

CHALLENGE WRITE UP

Operational Vulnerability

MỤC LỤC

I. CHALLENGE - 01

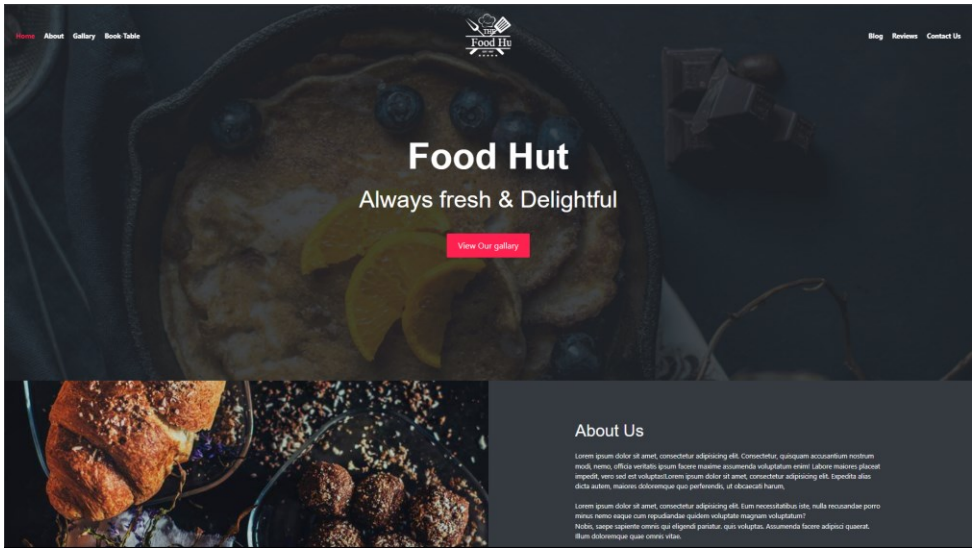
1. LAB OVERVIEW

Goal: Một thành phần của trang web đã bị cấu hình dư permission. Hãy tìm ra nó và lấy bí mật của nhà hàng.

**Note: Trong môi trường thí nghiệm của bài labs này, các bạn không cần thực hiện subdomains scan, directories scan, nmap.*

2. ANALYSIS

- Bước đầu tiên, sử dụng các tính năng của ứng dụng.
- Nhìn nhanh, đây là một landing page được dùng để giới thiệu về nhà hàng, nên không có nhiều tính năng để user tương tác.



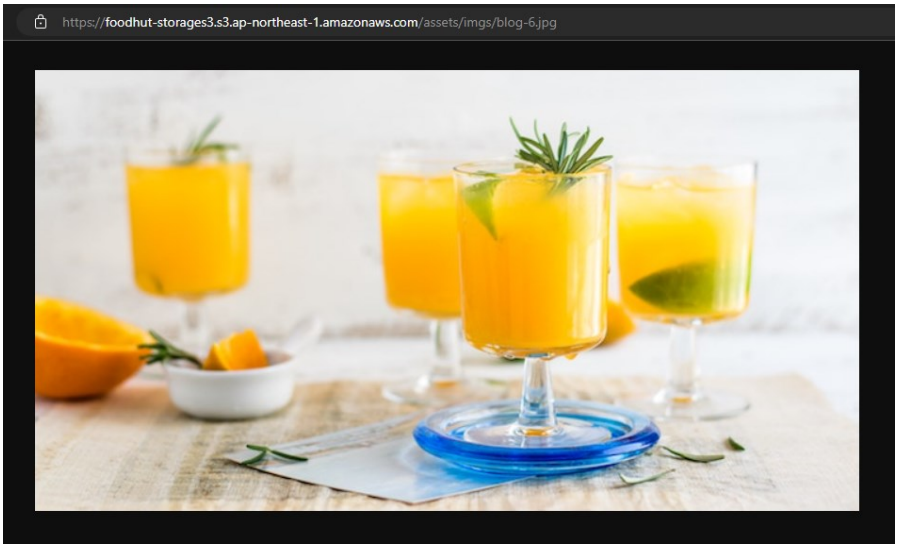
- Thu hẹp phạm vi kiểm thử, thực hiện kiểm tra source của trang web.
- Một vài thứ chúng ta cần để ý:
  - Các api, resource được gọi từ trang web
  - Nội dung các file javascript

- Quan sát thấy các resouce liên quan đến assets của trang web đều có điểm chung.

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <meta name="description" content="Start your development with FoodHut landing page.">
    <meta name="author" content="Devcrud">
    <title>FoodHut</title>

    <!-- font icons -->
    <link rel="stylesheet" href="https://foodhut-storages3.s3.ap-northeast-1.amazonaws.com/assets/vendors/themify-icons/css/themify-icons.css">

    <link rel="stylesheet" href="https://foodhut-storages3.s3.ap-northeast-1.amazonaws.com/assets/vendors/animate/animate.css">

    <!-- Bootstrap + FoodHut main styles -->
    <link rel="stylesheet" href="https://foodhut-storages3.s3.ap-northeast-1.amazonaws.com/assets/css/foodhut.css">
</head>
```

- Để ý ở các url đều trỏ đến tên miền của Amazon, cho thấy chúng sử dụng dịch vụ lưu trữ của Amazon gọi là Amazon S3 hoặc Amazon Simple Storage Service (S3 Bucket)

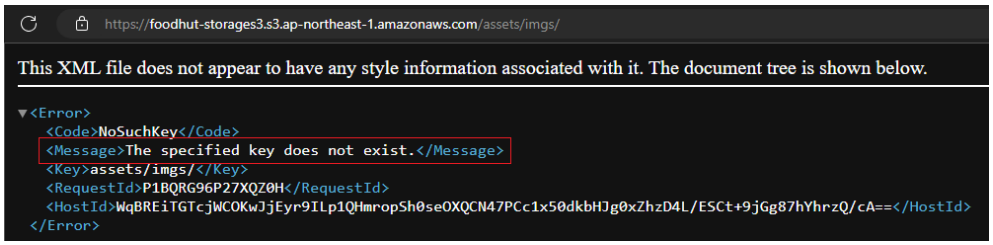
- Truy cập thử vào một url, https://foodhut-storages3.s3.ap-northeast-1.amazonaws.com/assets/imgs/blog-6.jpg.



- ***Đặt câu hỏi***: Vì đây là một bài liên quan đến misconfigured, nên mình tự hỏi liệu s3 bucket này có bị Directory Listing như bên apache2 không ?

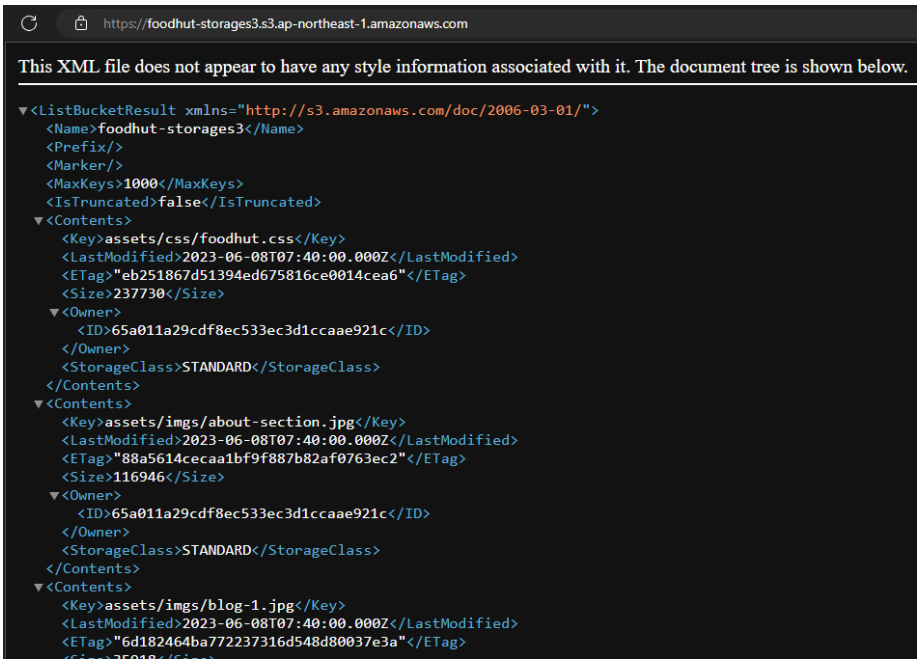
- Để kiểm chứng, thử xoá tên file ảnh trên url https://foodhut-storages3.s3.ap-northeast-1.amazonaws.com/assets/imgs/.

- Tới đây, ta được s3 trả về nội dung như sau, được định dạng bằng XML.

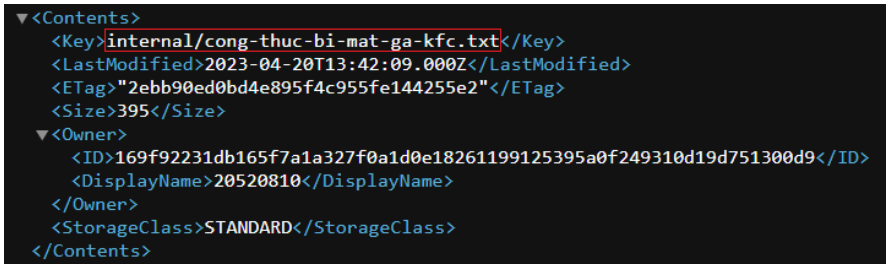


- Có tag `<Message>` thông báo gì đó về việc không có key.
- Chẳng lẻ s3 bucket này không bị Directory Listing ư?
- Mình thử xoá thêm từng folder `imgs/`, `assets/`, cho đến khi về index của S3 Bucket.

- Xuất hiện cái gì đó là lạ. Cũng là định dạng XML nhưng mà nó lạ lắm.



- Quan sát kỹ, ở các tag `<Contents>` sẽ có các tag `<Key>` nội dung là các endpoint của các resouces đang nằm trên S3 Bucket.
- Có vẻ như S3 Bucket này bị Directory Listing. Thử tìm kiếm xem có các endpoint nào lạ không.



- Phát hiện có một endpoint. Có vẻ như đây là bí mật mà Đề bài đã đề cập. Thực hiện truy cập vào endpoint và lấy Flag.

II. CHALLENGE - 02

1. LAB OVERVIEW

Ở challenge này gồm 3 level, liên quan đến cùng 1 ứng dụng.

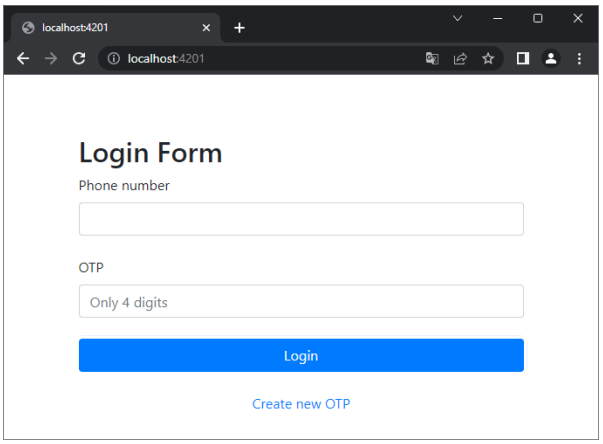
Goal: Chiếm tài khoản có số điện thoại 0123456789

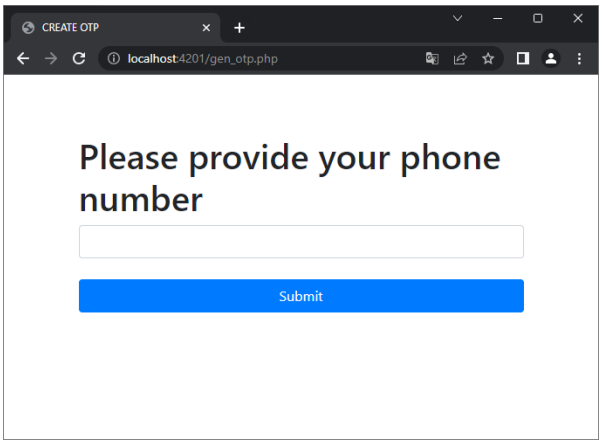
**Note: Trong môi trường thí nghiệm của bài labs này, các bạn không cần thực hiện subdomains scan, directories scan, nmap.*

2. ANALYSIS

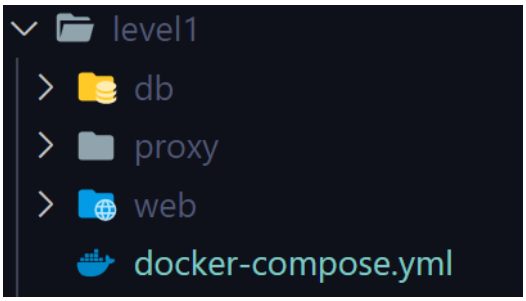
LEVEL 1:

CÁCH HOẠT ĐỘNG CỦA ỨNG DỤNG:





- Ứng dụng gồm 2 chức năng chính:
  - > Đăng nhập theo số điện thoại và OTP thay vì mật khẩu.
  - > Lấy mã OTP thông qua số điện thoại.
- Cùng phân tích soure code.



- Nhìn vào cấu trúc thư mục file + đọc nội dung của `docker-compose.yml`, ta có thể biết được ứng dụng này gồm 3 service chính tương ứng với 3 folders.
  - web: server đang host ứng dụng web
  - db: là database phục vụ cho ứng dụng web
  - proxy: làm trung gian tương tác giữa web và internet
- Tạm thời, chỉ cần quan tâm đến `web/`, gồm 3 file:
  - `index.php`, `gen_otp.php`, `connect_db.php`

- Trong `index.php`, đây là nơi xử lý quá trình đăng nhập của ứng dụng.

index.php

level1 > web > src > index.php > ...

```
1   <?php
2   include("connect_db.php");
3   if (isset($_POST['phone']) and isset($_POST['otp'])) {
4     $time = date("Y-m-d H:i:s");
5     $phone = $conn->real_escape_string($_POST['phone']);
6     $otp = $conn->real_escape_string($_POST['otp']);
7
8     $sql = "SELECT * FROM Users WHERE phone_number='$phone' AND otp='$otp';";
9     $result = $conn->query($sql);
10
11    $info = "";
12    if ($result->num_rows > 0) {
13      while ($row = $result->fetch_assoc()) {
14        // OTP will expire in 15 minutes
15        if ((strtotime($time) - strtotime($row['opt_created_time'])) > 900) {
16          $info = "<pre>Token is expired</pre>";
17        } else {
18          // this a phone number of vip account
19          if ($row['phone_number'] === '0123456789') {
20            $info .= "<pre>Login successful</pre>";
21            $info .= "<br>";
22            $info .= "<pre>CBJS{FAKE_FLAG}</pre>";
23          } else {
24            $info = "<pre>Login successful</pre>";
25          }
26        }
27      }
28    } else {
29      $info = "<pre>User not exist or OTP is wrong</pre>";
30    }
```

> **Line 4:** `$time` lưu trữ thời điểm khi cú request đăng nhập được gửi

> **Line 8:** `$phone` và `$otp` được nằm trong dấu nháy đơn của câu query `$sql`, kết hợp với việc được santize bằng hàm `real_escape_string()` (line 5,6) → SQL Injetion không xảy ra ở đây.

> **Line 15:** Kiểm tra xem mã OTP đã hết hạn hay chưa bằng cách so sánh `$time` với `opt_created_time` trong cơ sở dữ liệu (hết hạn nếu chênh lệch lớn hơn 900 giây hay 15 phút).

- Trong file `gen_otp.php`.

```php
<?php
include("connect_db.php");

$info = "";
3 references
function generateOTP($digits = 4)
{
    $i = 0;
    $otp = "";
    while ($i < $digits) {
        $otp .= mt_rand(0, 9);
        $i++;
    }
    return $otp;
}
```

> **Line 5** → **14**, hàm `generateOTP()` sinh mã OTP ngẫu nhiên 4 chữ số bằng hàm `mt_rand()` của PHP.

```php
if (isset($_POST['phone'])) {
    $phone = $conn->real_escape_string($_POST['phone']);
    $time = date("Y-m-d H:i:s");
    $sql = "SELECT * FROM Users WHERE phone_number='$phone';";
    $result = $conn->query($sql);

    if ($result->num_rows > 0) {
        while ($row = $result->fetch_assoc()) {
            if ((strtotime($time) - strtotime($row['opt_created_time'])) > 900 or $row['otp'] === NULL) {
                $otp = generateOTP();
                try {
                    $sql = "UPDATE Users SET otp='$otp', opt_created_time='$time' WHERE phone_number='$phone';";
                    $conn->query($sql);
                    $info .= "<pre>WE SENT OTP TO YOUR PHONE</pre>";
                    $info .= "<a href='/index.php'>Login</a>";
                } catch (\Throwable $th) {
                    throw $th;
                }
            } else {
                $info .= "<pre>OTP is not expired</pre>";
                $info .= "<a href='/index.php'>Login</a>";
            }
        }
    } else {
        $info .= "<pre>Phone Number is not exits</pre>";
    }
}
```

> **Line 16** → **42**, nhiệm vụ chính là tạo ra mã OTP chỉ khi nó đã hết hạn hoặc chưa được tạo trước đó.

CHECKPOINT:

- Mã OTP có 4 chữ số tức giá trị của mã OTP sẽ nằm trong khoảng 0000→9999 (10000 giá trị)
- Không có cơ chế rate-limit khi đăng nhập sai OTP

Ý TƯỞNG / GIẢ THUYẾT:

- Cho trung bình mất 0,5 giây để gửi một cú request đăng nhập.
- Trong vòng 15 phút, ta sẽ gửi được 1800 request → thử được 1800 mã OTP.
- Với xác xuất thành công 1800/10000 (~ 20%) do không có rate-limite, ta hoàn toàn có thể đăng nhập với đúng mã OTP trước khi nó hết hạn.

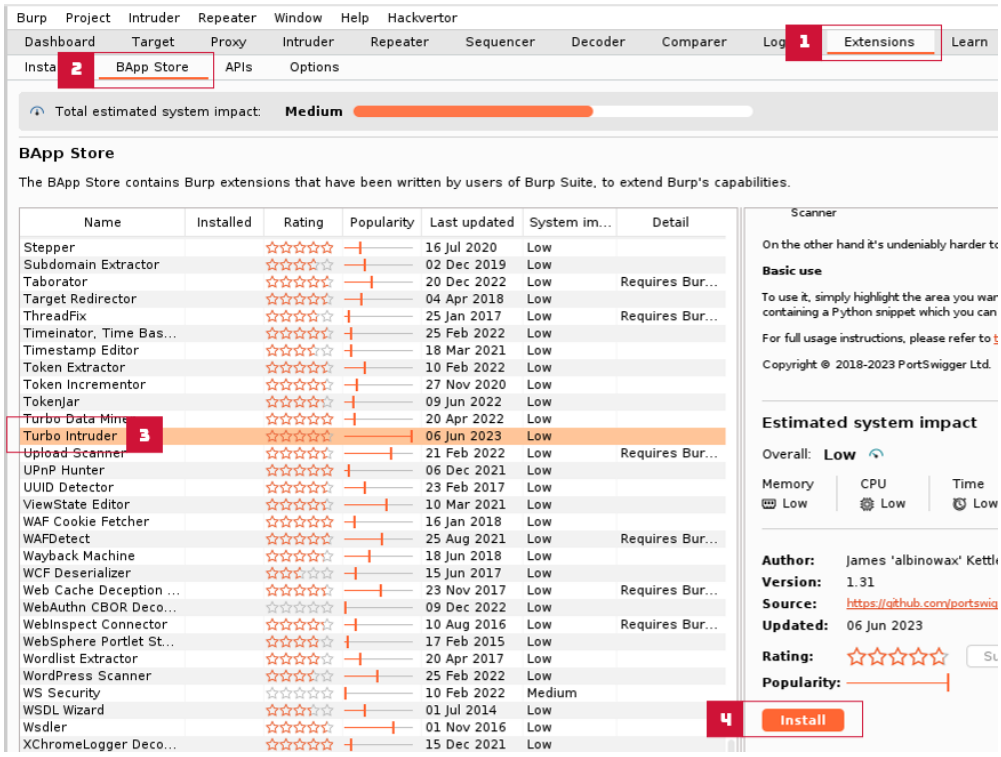
TIẾN HÀNH KHAI THÁC:

- Viết scripts đơn giản (1) để thực hiện ý tưởng hoặc có thể sử dụng các công cụ có sẵn, ở đây là một extension có trong Burpsuite (2).
- Vấn đề của cách khai thác này, nếu ta mất càng ít thời gian để gửi request → tỉ lệ thành công cao hơn.

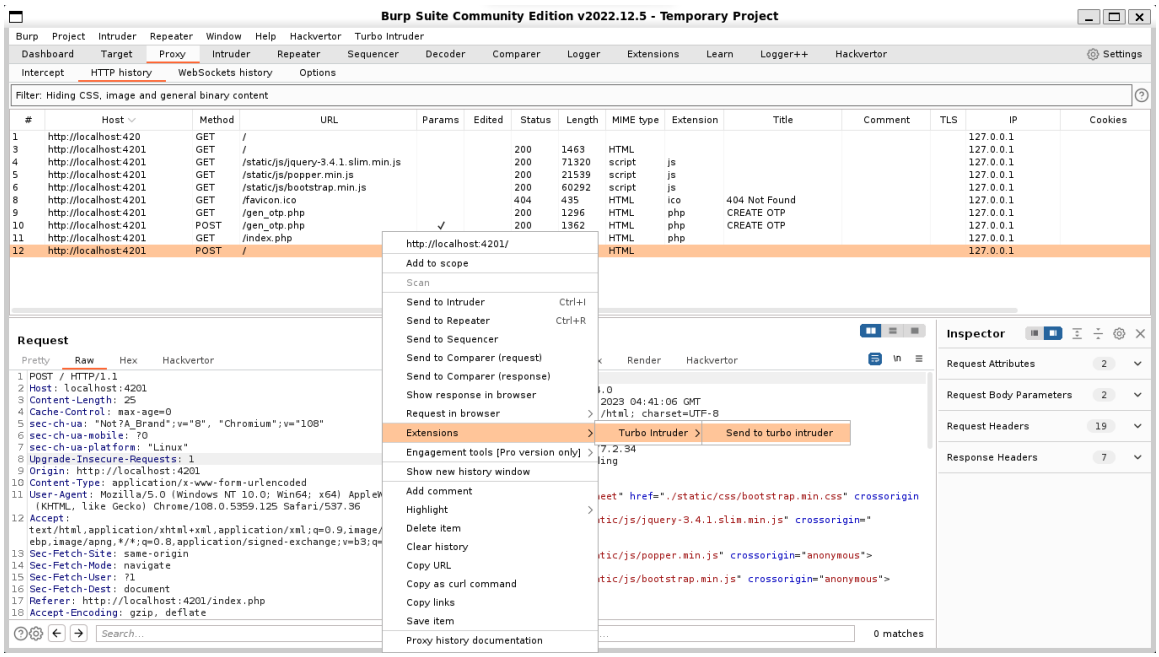
(1) Script python đơn giản: @0aa9a3

(2) Sử dụng Burpsuite:

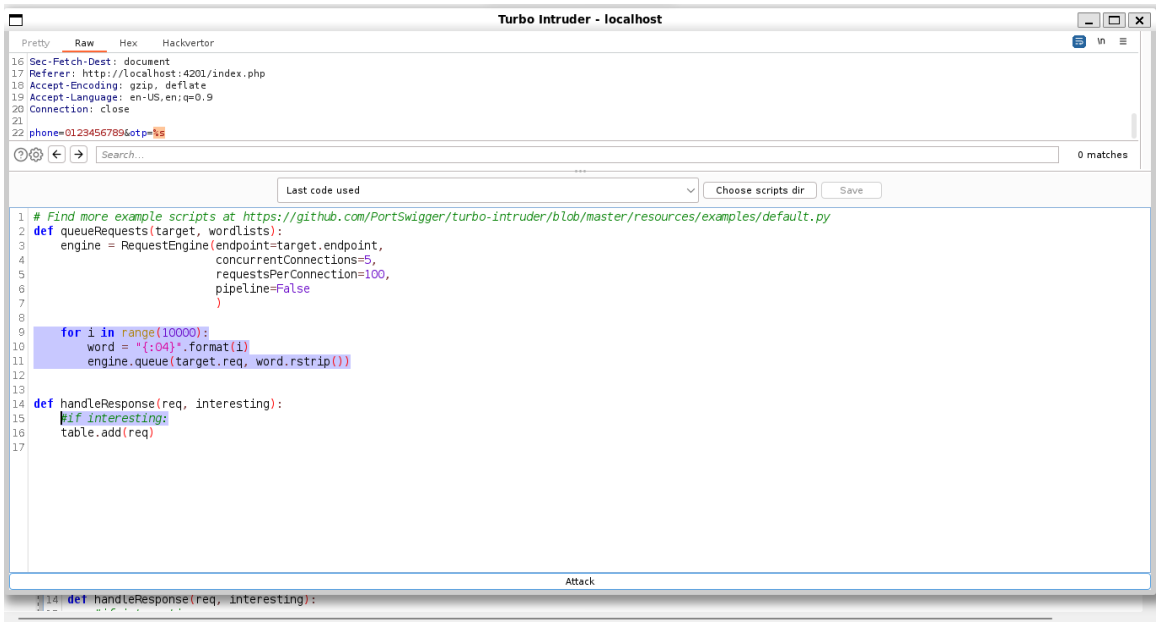
Cài đặt extension Turbo Intruder: Extensions → BApp Store → Turbo Intruder → Install



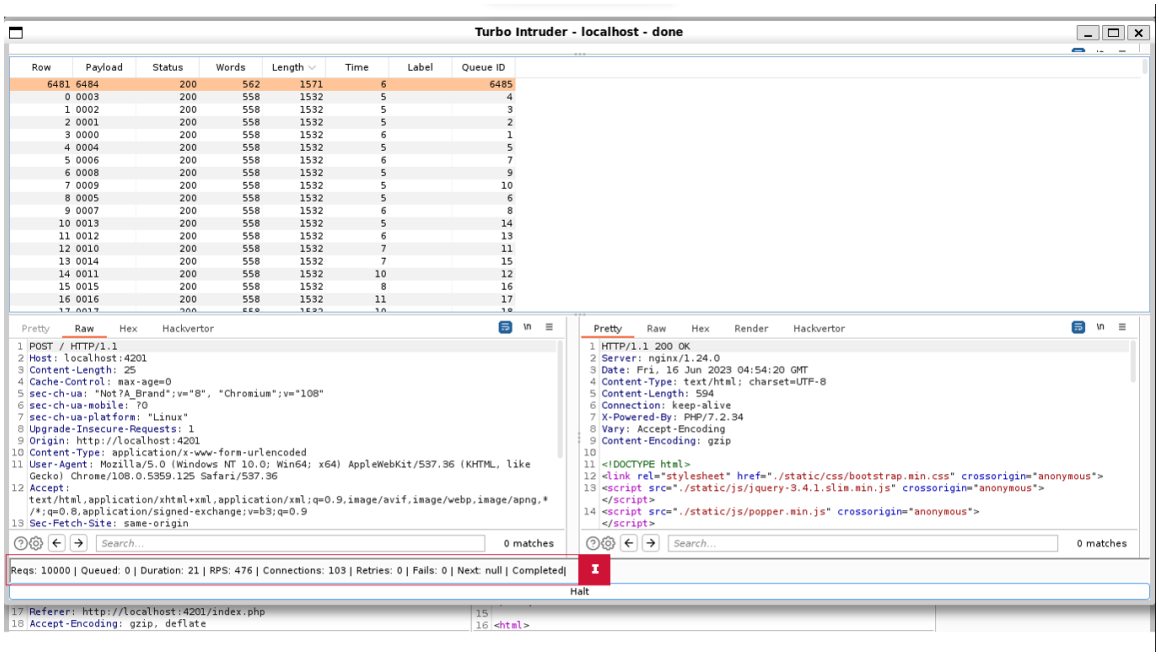
- **Bước 1**: Thực hiện tạo mã OTP mới.
- **Bước 2**: Bắt cứ request đăng nhập bằng Burpsuite, chuột phải vào forward sang Turbo Intruder.



- Set market cho Turbo Intruder bằng cách thay đổi value của params otp thành `%s`. (bôi đỏ)
- Modify lại script handler của Turbo Intruder như sau (bôi xanh)



- Bấm Attack và đợi kết quả.



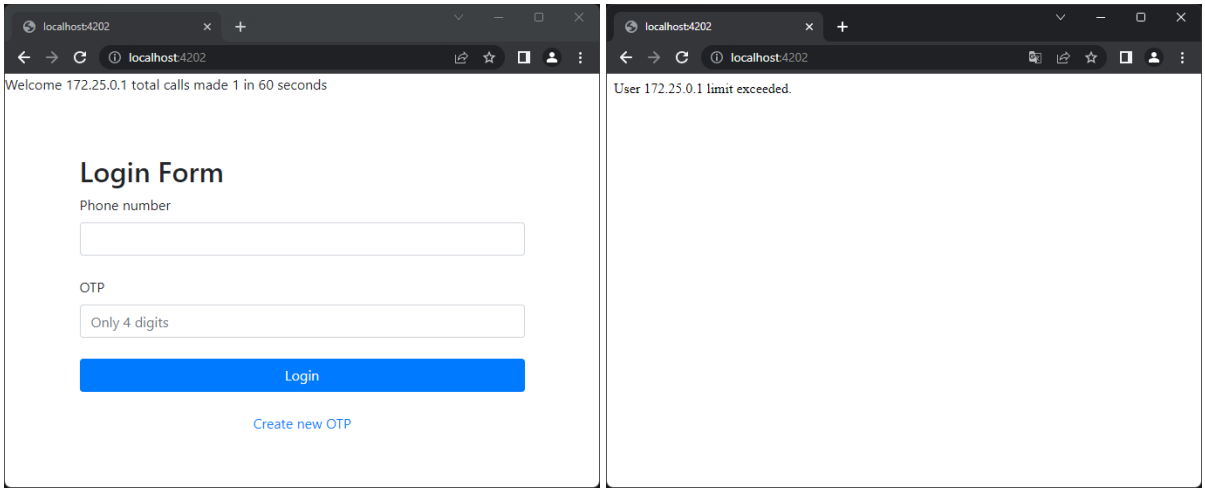
FYI: nếu để ý kĩ, cuộc tấn công vừa rồi Turbo Intruder đã gửi khoảng 500 requests/giây và hoàn thành gửi 10000 requests trong vòng 20 giây.

- **Bước 3**: Tới đây, chỉ cần đăng nhập với OTP vừa tìm được là lấy được flag.

LEVEL 2:

CÁCH HOẠT ĐỘNG CỦA ỨNG DỤNG:

- Về cơ bản, ứng dụng cũng có các chức năng giống level 1.
- Tuy nhiên, có thêm một tính năng mới là rate-limit, nôm na là chỉ cho phép 1 IP trong một thời gian nhất định được gửi một lượng requests, nếu vượt sẽ bị ban như hình.



- Trong folder `web/`, ngoài 3 file ở level1, đã xuất hiện thêm file mới: `rate_limit.php`
- File này được dùng để triển khai một rate-limit từ PHP và Redis. (Nguồn: @Digital Ocean - ow To Implement PHP Rate Limiting with Redis on Ubuntu 20.04)
- Cùng phân tích soure code của `rate_limit.php`.

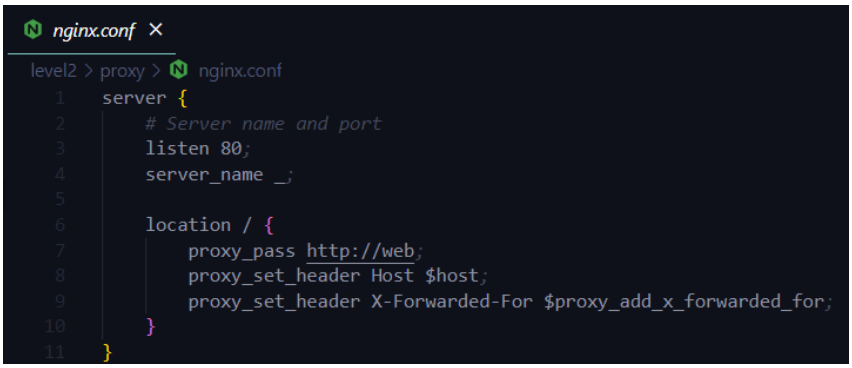
```
7   $max_calls_limit   = 100;
8   $time_period       = 60;
9   $total_user_calls  = 0;
```

- **Line 7 → 8**, thiết lập giới hạn số lượt truy cập tối đa `$max_calls_limit` (100 cú requests) đến web-server trong khoảng thời gian `$time_period` (60 giây)
- **Line 9,** khởi tạo giá trị ban đầu `$total_user_calls` – đây là biến dùng để lưu trữ và theo dõi số lượt truy cập vào web-server.

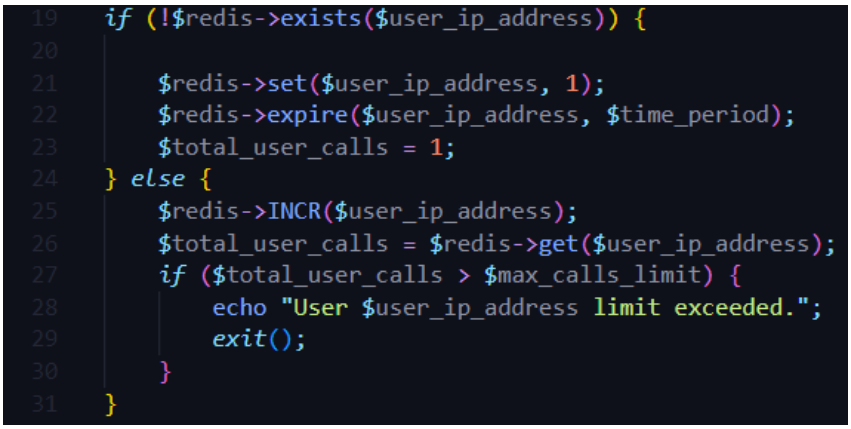
```
11  if (!empty($_SERVER['HTTP_CLIENT_IP'])) {
12      $user_ip_address = $_SERVER['HTTP_CLIENT_IP'];
13  } elseif (!empty($_SERVER['HTTP_X_FORWARDED_FOR'])) {
14      $user_ip_address = $_SERVER['HTTP_X_FORWARDED_FOR'];
15  } else {
16      $user_ip_address = $_SERVER['REMOTE_ADDR'];
17  }
```

- **Line 11** → **17**, xác định IP của user hiện tại từ các trường của HTTP theo thứ tự sau:
  - > `$_SERVER['HTTP_CLIENT_IP']`, nếu cú request tồn tại Header Client-IP, thì web-server sẽ lấy IP từ giá trị của Header đó.
  - > `$_SERVER['HTTP_X_FORWARDED_FOR']`, nếu cú request tồn tại Header X-Forwarded-For, thì web-server sẽ lấy IP từ giá trị của Header đó.
  - > Nếu không có 2 Header trên, thì web-server sẽ lấy IP từ Source IP nằm trong TCP packet.
- Tại sao việc lấy IP của user lấy phức tạp như vậy?
- Hiện tại (hay trong đa số thực tiễn) ta không trực tiếp truy cập vào web-server mà phải thông qua một con Proxy. Các Proxy này sẽ được cấu hình để tự động thêm các Header Client-IP và X-Forwarded-For để lưu chuyển IP của người dùng cho web-server xử lý.

- Quan sát cấu hình của Proxy trong file `proxy/nginx.conf`.



- **Line 8 và 9,** cấu hình cho nginx tự động thêm Header X-Forwarded-For với giá trị là ip của Client.



- **Line 19 → 30,** đây là flow xác định và theo dõi số lượt truy cập vào web-server của user, bao gồm tạo mới hoặc tăng giá trị đếm số lượt truy cập trong Redis và kiểm tra đã vượt giới hạn hay chưa.

CHECKPOINT:

- Các tính năng cũ vẫn được giữ nguyên.
- Implement rate-limit trên web-server.
- Rate-limit dựa trên IP, giới hạn 100 cú requests trong 60s cho mỗi IP.
- IP được lấy từ header 2 Header, Client-IP và X-Forwarded-For.

Ý TƯỞNG / GIẢ THUYẾT:

- Vẫn có thể bruteforce được nhưng tỉ lệ là 1500/10000 (15%).
- Các Header là một untrusted data cộng với việc rate-limit dựa trên IP → mà IP dựa vào Header → IP là một Untrusted Data (không check IP nhận từ các Headers có hợp lệ hay không)
- Ý tưởng: cải tiến script ở Level 1, cứ gửi 1 requests sẽ thay đổi IP một lần dựa vào việc gán giá trị cho Header Client-IP.

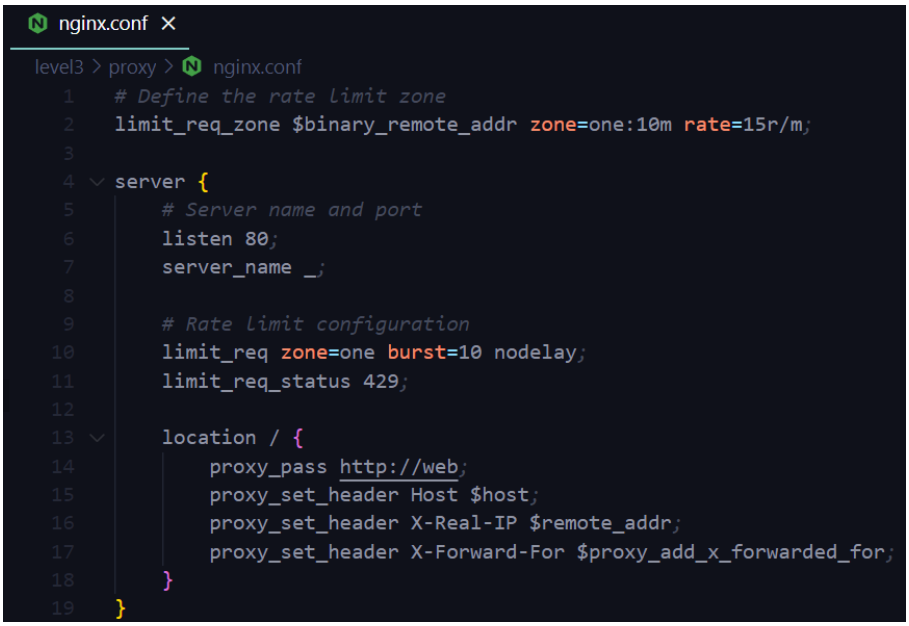
TIẾN HÀNH KHAI THÁC:

- Script python được cả tiến: @e3c655
- Ngoài ra mọi người cũng có thể sử dụng Turbo Intruder, bằng cách modify lại handler để xử lý thêm phần Header.

LEVEL 3:

CÁCH HOẠT ĐỘNG CỦA ỨNG DỤNG:

- Ứng dụng không có gì mới, các tính năng vẫn như cũ.
- Triển khai rate-limit trên Proxy thay vì Web-Server như level 2.
- Đọc `proxy/nginx.conf` để xem anh lập trình viên đã cấu hình rate-limit như thế nào.



> **Line 1:** Định nghĩa zone `one` để theo dõi số lượng request từ mỗi địa chỉ IP.
  - Kích thước của zone là 10 megabytes: lưu trữ thông tin về các requests mà IP đó gửi.
  - Giới hạn là 15 requests mỗi phút
> **Line 10 - 11:** Cấu hình thêm về giới hạn zone `one`
  - Trước khi áp dụng giới hạn: Cho phép gửi tối đa 10 requests.
  - Vượt quá giới hạn: sẽ không bị timeout (nodelay)
  - Vượt quá giới hạn: Server trả về status code 429.

CHECKPOINT:

- Về cơ bản, khi truy cập vào Web-Server, các traffic phải đi qua Proxy.
- Và khi thực hiện gửi nhiều request (vượt quá giới hạn) sẽ bị Proxy trả về status code 429.

Ý TƯỞNG / GIẢ THUYẾT:

Mình có 2 giả thuyết chính:

- Với rate-limit này liệu ta vẫn có thể thực hiện bruteforce? (1)
- Còn có con đường nào khác để vào Web-Server mà không đi qua Proxy hay không? (2)

TIẾN HÀNH KHAI THÁC:

(1) Câu trả lời là có, nhưng tỉ lệ khá thấp.

(2) Làm sao để biết được con Web-Server này có mở một cổng khác ra ngoài internet không ?

- Nếu là môi trường thực tiễn, mọi người có thể thực hiện scan ports để tìm kiếm.

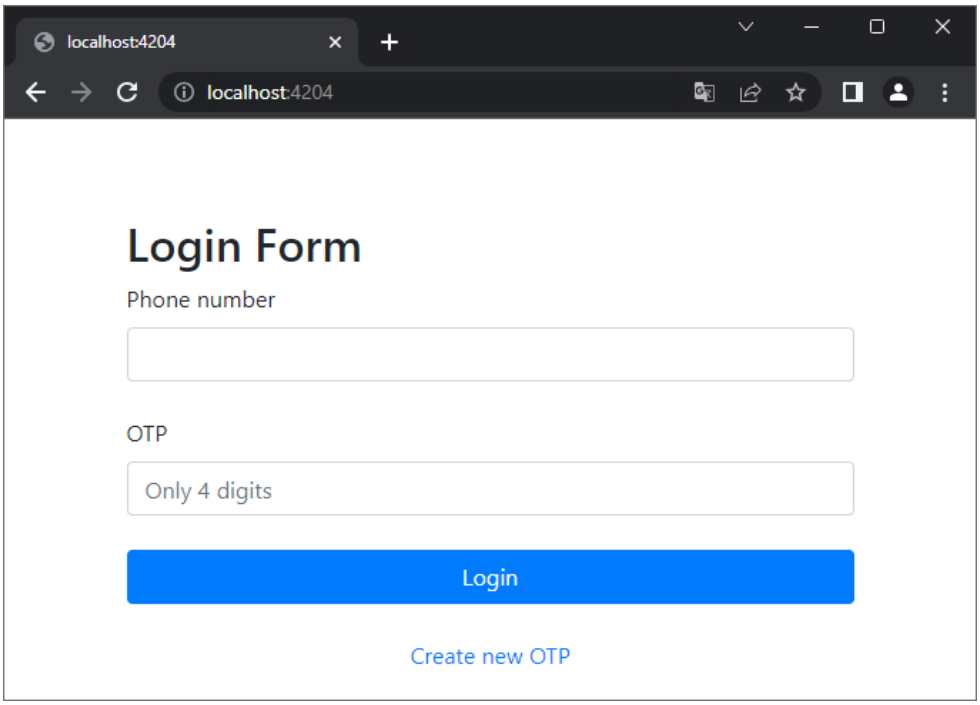
- Nhưng đây là môi trường lab và ta được cung cấp source code, nên ta hoàn toàn có thể check được.

- Nhìn vào `docker-compose.yml`

```
19    web:
20      container_name: 2FA_level3_web
21      depends_on:
22        - database
23      ports:
24        - "4204:80"
25      build: ./web
26      environment:
27        - MYSQL_HOSTNAME=database
28        - MYSQL_DATABASE=myDB
29        - MYSQL_USER=db_user
30        - MYSQL_PASSWORD=db_password
31
32    proxy:
33      container_name: 2FA_level3_proxy
34      ports:
35        - "4203:80"
36      image: nginx:stable
37      volumes:
38        - ./proxy/nginx.conf:/etc/nginx/conf.d/default.conf
39      restart: always
```

> **Line 24**: đây là cấu hình để mapping port 80 của container web-server với port 4204 của máy host.

> Điều này có nghĩa, là ta hoàn toàn có thể truy cập vào web-server thông qua port 4204.



- Tới đây, có thể sử dụng lại script và Burpsuite ở level 1.