$5000 Bounty, Bypassing a Rate Limit Twice

Table of Contents

Introduction

My name is Behrooz Arya. I'm an employee and a part-time bug hunter, currently 43 years old. About three years ago, I entered the field of security, and for the past year, I've been working seriously as a bug hunter. I've always had an interest in security, but I honestly didn't know where to start. That changed three years ago when I watched an interview between Mr. Ashkan Rahmani and Mr. Shahinzadeh, where they discussed a roadmap for entering the field of security. That interview helped me find my direction.

From that point, I began learning and completed several courses, including Security+, CEH, and PWK. Later, I transitioned to web security with the OWASP and Mr. Shahinzadeh's HUNT courses. Since I also work full-time, I dedicate four to five hours a day to bug hunting on a part-time basis.

What is rate limit?

Rate limiting is a method of controlling traffic flow to a service or server by restricting the number of requests that can be made within a certain time frame. It is an essential technique for preventing resource abuse, ensuring fair use of services and protecting against DDoS attacks

Why rate limit?

Attacks that rate limiting can help mitigate include:

- **DDoS attacks**:

  In this type of attack, a large volume of requests is sent to the server, it and causing it to crash or become unavailable.

- **Brute force attacks:**

  These attacks are based on trial and error. The attacker sends numerous different requests in an attempt to guess passwords or obtain sensitive information**.**

- **Web scraping:**

  The attacker uses a series of scripts and automation tools to extract large amounts of data from websites.

Where to Detect?

- **Login Pages**

  Bypassing rate limits can allow brute-force attacks on credentials

- **Account Registration**

  Attackers can create multiple accounts by bypassing rate limits

- **Password Reset**

  Allows attackers to trigger multiple password reset emails

- **2FA/OTP**

  Bypassing rate limiting allows an attacker to use brute force to obtain the correct OTP

- **API Endpoints**

  Rate limits on API requests, such as data scraping or mass operations

- **Voucher Codes, comments, etc**

  Rate limits should be tested on both voucher codes and comments sections of a website. If bypassed, an attacker could brute-force discount codes using wordlists to find valid ones or flood the comments section with spam, overwhelming the site with irrelevant content.

Rate limit systems

- **HTTP Packet**
  - IP-Based Rate Limiting:

    If the rate limit is implemented based on the user's IP address, requests are restricted according to the user's IP address.
  - Session based:

    If the rate limit is implemented based on the user session, the number of requests is restricted for each user as long as their session remains active.
- **End user inputs**
  - Username, email address, phone number, etc:

If the rate limit is implemented based on user inputs, the server restricts the number of requests associated with a specific identifier, such as a username, email, or phone number.

Attacks and Bypasses

IP-Based?

When the rate limit is implemented based on IP, it may be bypassed using techniques like **IP spoofing** or **IP rotation**.

- **IP spoofing with HTTP headers :**

  In this method, the attacker uses specific HTTP headers to manipulate their IP address, making it appear as a different IP to the server. This way, the server believes that the requests are coming from different IPs, effectively bypassing the rate limit restrictions.

  ```
  X-Original-IP: 127.0.0.1
  X-Forwarded-For: 127.0.0.1
  X-Remote-IP: 127.0.0.1
  X-Remote-Addr: 127.0.0.1
  X-Client-IP: 127.0.0.1
  X-Host: 127.0.0.1
  X-Forwared-Host: 127.0.0.1

  # Double X-Forwarded-For header example
  X-Forwarded-For:
  X-Forwarded-For: 127.0.0.1
  ```

- **IP Rotation:**
  - **Using Tor :**

    Writeup Abbas.heybati: Bypass Rate Limit Request (fuzzing/etc...) With TOR

  - **IP Rotate using Burp extension**

Session based

- **Session Token Abuse:**

  if the rate limit is implemented based on the user session, it may be possible to bypass it by changing the session token. This could make the server believe that the requests are coming from different users, thereby bypassing the rate limit.

- **Changing User-Agent:**

  By rotating the User-Agent with each request, an attacker can make it appear as though requests are coming from different devices or browsers. This can sometimes deceive the server into treating each request as unique, thereby bypassing rate-limiting restrictions based on user behavior.

User inputs?

1. **Appending null bytes or special characters**

   Adding %00, %0d%0a, %0d, %0a, %09, %0C, %20

   For Example: test@test.com%00

2. **ASCII encoding:**

   For example: test@test.com => t%65st@te%73t.com

3. **Case swapping**

   For example: test@email.com , teSt@email.com

4. **HTTP Parameter Pollution (HPP)**

   Example:

   GET/login/otp/code=123&email=bypass123@mail.com&email=victim@mail.com

   or Adding unnecessary parameters to URLs

   Example: https://example.com/resource?id=123&random=987

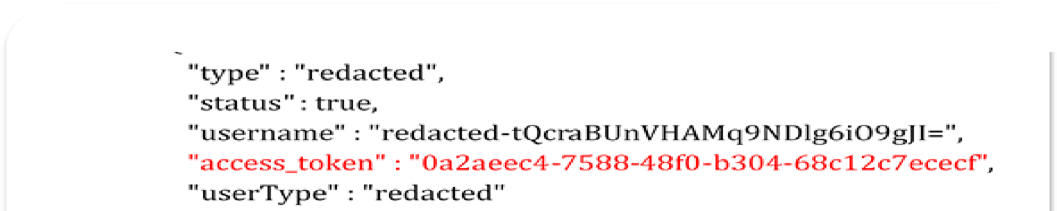
Additional Techniques

- **URL Manipulation:**
  - https://example.com/login/
  - https://example.com/logiN
  - https://example.com/login%00
- **Changing HTTP request methods:**
  - GET ⇔ POST , PUT ⇔ PATCH
- **Bypassing Rate Limits with Protocol Downgrading**
  - HTTP/2 => HTTP/1.1
- **Bypassing rate limits via race conditions**
- **Invalid phone Number :**
  - Registering an invalid phone number in applications that send OTP for login or password reset. Example: phonnumber='1234567890'

My Vulnerabilities - Case NO 1

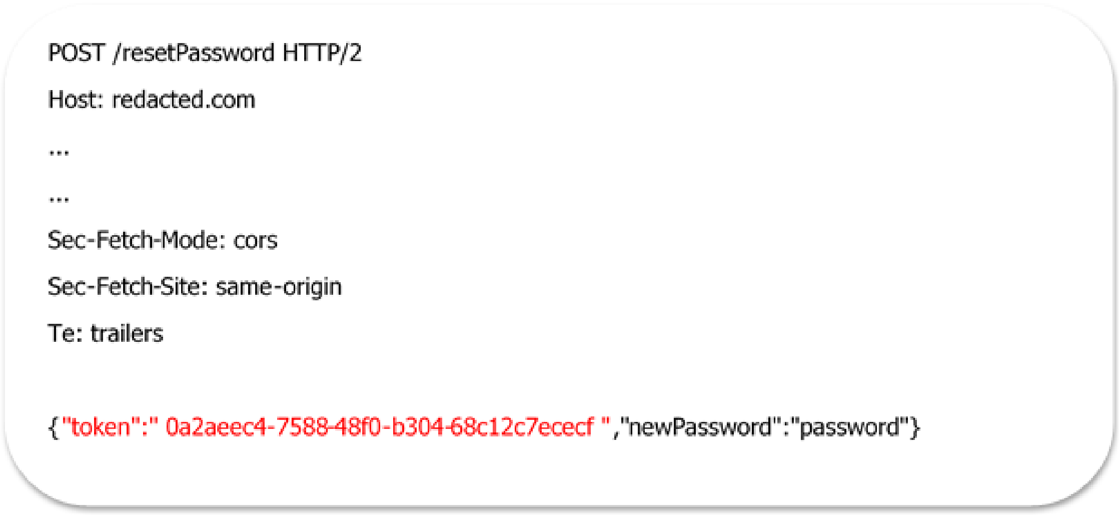
Rate Limiting Missing in OTP Password Reset and Complete ATO

The first vulnerability I want to share with you is related to one of the public targets on HackerOne, which had no rate limit on OTP reset for passwords, allowing for account takeover. The normal procedure for resetting the password on this site worked as follows: in the "Forgot Password" section, when a user entered their email, a six-digit OTP was sent to their email, which remained valid for 30 minutes. If the entered OTP was incorrect, an "invalid OTP" message would appear in the response, and after five failed attempts, the user's account would be disabled. If the OTP was correct, an access token would be returned in the response as shown in the request.



```
"type" : "redacted",
"status" : true,
"username" : "redacted-tQcraBUnVHAMq9NDlg6iO9gJI=",
"access_token" : "0a2aeec4-7588-48f0-b304-68c12c7ececf",
"userType" : "redacted"
```

To reset the password, it was enough to send the access token value along with a new password in the request, and the account password would be changed.

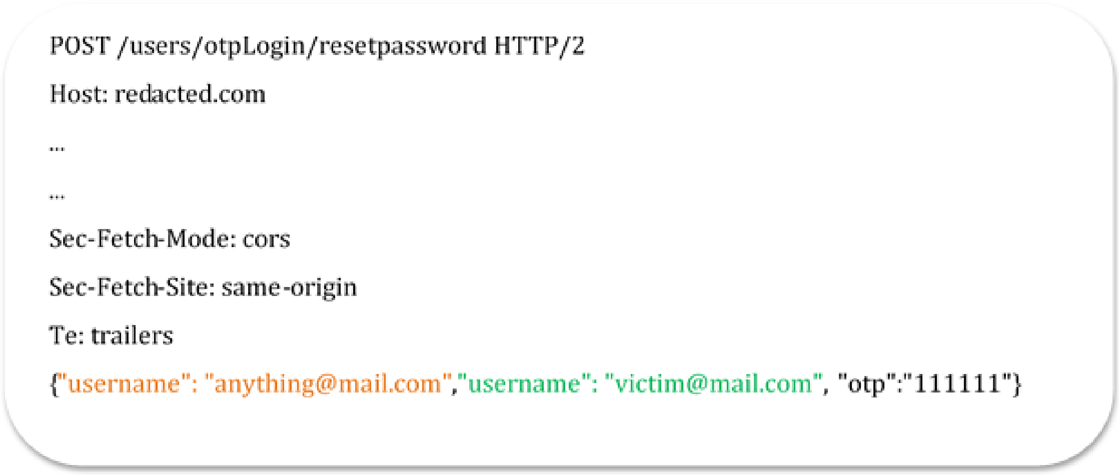


```
POST /resetPassword HTTP/2
Host: redacted.com
...
...
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers

{"token":" 0a2aeec4-7588-48f0-b304-68c12c7ececf ","newPassword":"password"}
```

The email sent for the password reset mentioned that the OTP was valid for 30 minutes, which made me think that if I could bypass the rate limit, I could brute-force all six-digit numbers and find the correct OTP. I tested several methods and realized that the rate limit was based on the user's email.I added special characters and Null Byte to the email, but I received the message "This email does not exist." I also tried using ASCII codes for some characters in the email, but it didn't work. Using a mix of uppercase and lowercase letters in the email also failed.

Then, I used the HTTP parameter pollution technique and sent a request with two username fields. I set the first username field to a random email and the second one to the victim's email. Every time I sent the request using this method, I received the "invalid OTP" message, and even after sending more than five requests, the account was not disabled. To ensure that this bypass method still worked with multiple requests, I sent 5000 requests via Burp Suite's intruder, and saw that the method still worked.



```
POST /users/otpLogin/resetpassword HTTP/2
Host: redacted.com
...
...
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
{"username": "anything@mail.com","username": "victim@mail.com", "otp":"111111"}
```

Since I had found this vulnerability late at night, I decided to report it the next day. However, when I tried to write the report and prepare the proof of concept (PoC) the following day, I discovered that the bypass no longer worked. After five requests, the user's account was deactivated for 24 hours. It seemed the vulnerability had been patched. But I didn't give up. I created several new accounts and tested various methods. After five requests, the account was disabled again. I then thought, if I sent the OTP that was emailed to me in the request, would the "account disabled" message appear again? To my surprise, the access token was returned in the response. By placing the access token in the password reset request along with a new password, the password changed, and the disabled account was reactivated.

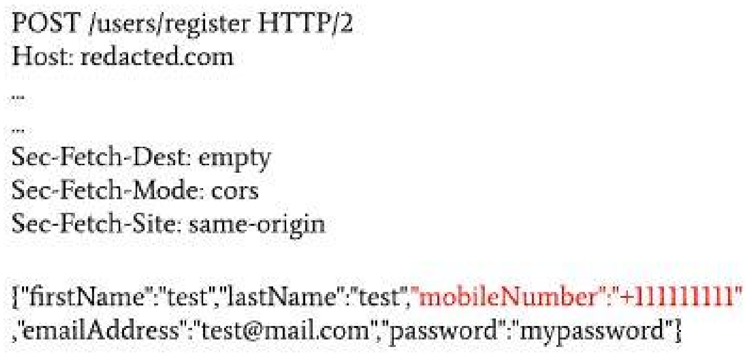
In fact, even though the account was disabled, if I sent the correct OTP, it was still valid, and the code did not expire. Since the OTP sent to the email was valid for 30 minutes, I could obtain the correct OTP within that time. To do this, I wrote a brute-forcing script in Go to try all six-digit combinations. I successfully found the correct OTP and managed to take over the account. I submitted the report, and it was confirmed. I was awarded a $3000 bounty for the vulnerability.

My Vulnerabilities - Case NO 2

Bypass of Resolved Report: Missing Rate Limiting in OTP Password Reset Leading to ATO

In this section, I would like to share details about the second vulnerability I identified. After the previous vulnerability was patched, I revisited the “Forgot password” section of the website to continue testing. To my surprise, I found that the OTP verification rate limit had not been applied uniformly: for some accounts the rate limit had been enforced, but for others there had been no rate limit at all.

Having spent considerable time on this target, I had moved away from registering accounts through the site’s UI. Instead, I intercepted the signup request in Burp Suite and submitted it directly to create accounts. Each time I wanted to create a new account, I simply modified the email field while keeping the other fields unchanged, with the mobile number consistently set to "+111111111." When I then initiated a password reset request for these accounts, no rate limit was enforced.



```
POST /users/register HTTP/2
Host: redacted.com
...
...
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{"firstName":"test","lastName":"test","mobileNumber":"+111111111"
,"emailAddress":"test@mail.com","password":"mypassword"}
```

However, when I created a new account through the site's UI and subsequently requested a password reset for that same account, rate limiting was enforced during OTP verification. After four incorrect OTP attempts, the OTP would expire, and I would need to request a new one.

At first I didn’t understand the reason for this behavior, but further analysis revealed that the site's UI only allowed mobile numbers that start with zero, a restriction that is only enforced on the client side. Thus, if an account was registered with a mobile number starting with "0111111111"," rate limiting was applied during OTP verification. However, registering or updating an account with an invalid mobile number (one that didn’t start with zero) allowed me to bypass the rate limit during the reset process..

Bypass: Changing the first digit of the mobile number to + or any number from 1 to 9 had made the mobile number appear invalid to the web application, thereby bypassing the rate limit. Since the OTP had been sent to both email and mobile, the application hadn't been able to properly expire the OTP when the mobile number had been invalid. As a result, even after each incorrect OTP entry, a message had appeared stating "OTP expired," yet the OTP hadn't actually expired, allowing me to brute-force all six-digit codes until I had found the correct OTP.

To achieve this, I modified the Go script I had previously written to brute-force all six-digit combinations. Upon execution, I successfully identified the correct OTP and gained account access. I submitted a report, which was validated, and I received a $2000 bounty for this vulnerability.