Optimize Node.js performance

Identifying performance bottleneck of Node.js Application

(1) Profiling and Monitoring tool:

(a) Node.js built-in profiler

(b) chrome dev tools

(c) clinic.js

(d) PM2 – Process Manager 2 tool

(a) Node.js built-in profiler

It is a command line tools that comes with Node.js to find the slow parts in your code.

(b) chrome dev tools

It lets you visually see CPU usage, memory leaks and slow parts in your Node.js application.

(c) Clinic.js

It is a collection of tools that automatically finds performance issues.

(d) PM2

It runs and monitors Node.js application in production. It also shows CPU and memory usage.

It can automatically restart your application if it crashes.

(2) External tools:

(a) New Relic
(b) Datadog

} cloud-based systems

(a) New Relic

It is a cloud-based performance monitoring tool. It tracks real users, response time, database queries and errors.

(b) Datadog

It is another cloud-based tool that monitors your entire system.

Key Metrics to Monitor

(a) Response time

(b) Throughput – no. of requests handled per unit time

(c) Event loop Latency – measures the delay in our event loop

(d) Memory usage

(e) CPU usage

Security Vulnerabilities in Node.js Applications

(1) Cross-Site Scripting (XSS)

It is a security issue where a website accidently allows harmful scripts to run inside a user's browser.

The attacker usually hides a small piece of JS inside something the website displays. When another user opens the page, their browser runs the script as if it were part of the website. This happens because the browser automatically trusts everything that comes from the website.

There are three types of cross-site scripting-

(a) Stored XSS

This attack occurs when the attackers script becomes permanently saved inside the website's database. That means, every time someone visits that page, the malicious script runs automatically without the attacker needing to do anything else.

(b) Reflected XSS

this attack occurs when the malicious script travels through a link, input and the server immediately sends it back to the browser. It is not stored anywhere, only triggered when someone clicks the dangerous link.

(c) DOM-based XSS

this attack occurs entirely inside the user's browser. Here, the website's own JS takes something from the URL or page environment and places it into the web page without checking it. The attacker simply manipulates the part of the page, the script relies on.

(2) Cross-site Request Forgery (CSRF)

CSRF is an attack where the victims browser is tricked into performing an action on a website without the victim knowing. This works only when the victim is already logged in to a website such as a bank, email service or social media accounts. The browser automatically sends log in cookies whenever it

communicates with that website. A hacker abuses this behavior by making the victim load a hidden request often disguised inside an image, link, or, a webpage. ~~Because~~

As the browser, sends the victim's cookies automatically, the website thinks, the victim intentionally make the request

Node.js

SQL Injection

It is a vulnerability where the attacker is able to place harmful instructions inside input boxes or URLs that interact with the database. When a system does not check or clean the input before sending it to the database, the attacker can cleverly insert special characters and database commands that change how the server query behaves. Instead of treating the input as normal text, the database treats it as part of the query itself.

command Injection

It happens when an app-n takes user input and directly uses it inside a system-level command on the server. The attacker can attach extra system commands to the input, when the server runs it, it follows everything the attacker added. If the input that forms the command is not checked properly, an attacker can trick the server into running dangerous operating system commands.

Implementing Security Measures in Node.js

(a) Input validation

(b) Output Encoding

(c) use security libraries or Modules

(d) Authentication and Authorization

(e) Secure coding Practices

(a) Input validation.

It is the process of making sure that the data coming from the user is exactly what the appl-n expects. If a website accepts any kind of input without checking it, the attackers can insert harmful content, such as - scripts, SQL statements or system commands.

Proper input validation prevents this by strictly defining what is allowed. One common approach is - sanitizing the input, which means removing or cleaning any suspicious characters or patterns. Another approach is - white listing, which means accepting only specific, approved values.