Node JS

- **JavaScript Runtime Environment**
- **It is used for Sever Side Programming**
- **Node.js is not a language, library or framework**

🧠 Node.js – Complete Notes

🚀 What is Node.js?

> **Node.js** is an **open-source, cross-platform, runtime environment** built on **Chrome's V8 JavaScript engine**.

- It allows you to run **JavaScript outside the browser**.
- Primarily used to build **backend servers, APIs, tools, and CLIs**.
- Non-blocking, **asynchronous I/O** is perfect for handling real-time, high-concurrency workloads (like APIs, chats, etc).

✅ Key Features:

- **Single-threaded event loop**
- **Non-blocking asynchronous I/O**
- **Uses CommonJS module system**
- Built-in support for **TCP, HTTP, file system, streams, etc**
- Excellent ecosystem: **npm**

🧪 REPL (Read Eval Print Loop)

> The **Node.js REPL** is a command-line environment to quickly run JS code.

- To start:

  ```
  node
  ```

- Features:
  - Live code testing
  - Built-in commands: `.exit`, `.help`, `.editor`
  - Access Node internals like `fs`, `path`, etc.

🧠 Use Case for Pentesters:

- **Payload testing** (JSON eval, obfuscated JS)
- **Reverse engineering Node malware payloads**

📁 Node Files: Creating and Running

Create a file:

```
// app.js
console.log("Hello from Node.js!");
```

Run it:

```
node app.js
```

✅ Files can be `.js` or `.mjs` (for ES modules)

🔁 The Node.js Process

You get access to the `process` global object:

```
console.log(process.pid);           // Process ID
console.log(process.platform);      // OS platform
```

```
console.log(process.argv);          // Command-line args
console.log(process.env);           // Environment vars
```

🧠 Bug Bounty Note:

Check for apps leaking env variables (e.g., `.env` or `process.env.SECRET` exposed in SSRFs or debug endpoints).

📤 Export in Files – `module.exports`

> Used to expose functionality from one file to another.

🔹 Example 1 – Simple Export:

```
// user.js
const user = {
  name: "Alice",
  role: "admin"
};
module.exports = user;
```

```
// app.js
const user = require('./user');
console.log(user.name); // Alice
```

🔹 Example 2 – Export Functions:

```
function add(a, b) {
  return a + b;
}
module.exports = { add };
```

📁 Export in Directories (index.js trick)

You can **bundle multiple exports** into a directory with an `index.js`:

```
utils/
  ├── add.js
  ├── subtract.js
  └── index.js
```

```
// utils/index.js
module.exports = {
  add: require('./add'),
  subtract: require('./subtract')
};
```

Usage:

```
const { add, subtract } = require('./utils');
```

✅ Perfect for organizing large apps.

📦 What is npm?

> `npm` = **Node Package Manager**
> It's the **default package manager** for Node.js.

Used to:

- Install, update, remove packages
- Manage dependencies (`package.json`)
- Share open-source modules

📁 package.json

This file describes your project, including:

- Name, version, author
- Dependencies & devDependencies
- Scripts

🔹 Create it:

```
npm init
```

```
{
  "name": "myapp",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.18.2"
  },
  "scripts": {
    "start": "node index.js"
  }
}
```

📦 Installing Packages

Local Installation:

```
npm install lodash
```

- Installs to `./node_modules`
- Listed in `dependencies`

Global Installation:

```
npm install -g nodemon
```

- Accessible globally (CLI tools)
- Not listed in `package.json`

🧰 Importing Modules

CommonJS (default in `.js`)

```
const fs = require('fs');
const lodash = require('lodash');
```

ES Modules (requires `"type": "module"` or `.mjs`)

```
import fs from 'fs';
import _ from 'lodash';
```

🔍 Built-in Core Modules

Module	Use
`fs`	File system
`http`	Build web servers
`path`	File path utils
`os`	Info about system
`process`	Info/control over Node process
`crypto`	Cryptography utilities
`child_process`	Run system commands

🔥 Offensive Tip:

`child_process.exec()` is a common **command injection sink** in SSRF or RCE payloads.

🛡️ Security Notes (Bug Bounty Context)

🐍 Dangerous Functions in Node:

- `eval()`
- `child_process.exec()`
- `vm.runInContext()`
- `require()` with user input (can lead to `require('child_process')`)
- `fs.readFile(userInput)` → Path traversal

📌 Node.js Attack Scenarios

✅ Prototype Pollution (in lodash, deep merge libs):

```
const malicious = JSON.parse('{ "__proto__": { "admin": true } }');
Object.assign({}, malicious);
```

If app later does:

```
if (user.admin) { // Attacker forced true }
```

✅ Remote Code Execution via `child_process`

```
const exec = require('child_process').exec;
exec(req.query.cmd); // RCE if unvalidated!
```

✅ Safe Patterns

- Always validate/escape user input
- Avoid `eval`, `Function`, `exec`, `spawn` unless **100% required**
- Use `helmet`, `express-rate-limit`, `cors` in APIs
- Don't trust `req.body`, `req.query`, or `req.params`

🧪 Pentesting Tools Built in Node.js

Tool	Purpose
http-proxy	Build reverse proxy
Express.js	API/server building
Socket.io	Real-time WebSocket communication
axios/got	HTTP clients
inquirer	CLI prompt building

🧠 Summary Cheat Sheet

Concept	**Code/Description**
Run file	`node file.js`
Create module	`module.exports = value`
Import module	`require('./module')`
Start project	`npm init`
Install pkg	`npm install express`
Built-in modules	`fs` , `http` , `path` , `crypto`
Process info	`process.env` , `process.argv`
Global install	`npm install -g nodemon`
REPL	`node`