Great, let's move on to **Topic 15: Node.js Security Best Practices**.
Since you're already experienced, I'll highlight **real-world practices, vulnerabilities, and interview-focused points**.

---

15. Node.js Security Best Practices

Security is critical in backend apps. Node.js, being event-driven and highly network-exposed, is **prone to common web vulnerabilities** if not handled properly.

📌 Common Security Threats in Node.js

1. **Injection Attacks** – SQL injection, NoSQL injection, command injection.
2. **Cross-Site Scripting (XSS)** – Inserting malicious scripts in user input/output.
3. **Cross-Site Request Forgery (CSRF)** – Exploiting authenticated sessions.
4. **Denial of Service (DoS)** – Event loop blocking, large payloads.
5. **Insecure Dependencies** – Using outdated/vulnerable npm packages.
6. **Directory Traversal** – Accessing restricted files (`../../etc/passwd`).

📌 Best Practices

1. **Keep Dependencies Secure**

- Use `npm audit` or `yarn audit` to detect vulnerabilities.
- Example:

```
npm audit fix
```

- Use **Snyk**, **Dependabot**, or **npm outdated** to track issues.

👉 *Interview Tip*: Expect questions like *"How do you ensure npm package security in production?"*

---

2. **Validate & Sanitize Input**

- Never trust user input.
- Use libraries like **express-validator**, **joi**, **validator.js**.
- Example:

```
const { body } = require("express-validator");
```

```
app.post("/user", [
  body("email").isEmail(),
  body("password").isLength({ min: 8 })
], (req, res) => {
  res.send("User validated");
});
```

---

3. **Prevent NoSQL Injection**

- For MongoDB, avoid raw queries like:

```
db.users.find({ username: req.body.username });
```

- Use **parameterized queries** or **ODM like Mongoose**.

---

4. **Secure Authentication**

- Always hash passwords with **bcrypt** or **argon2**.
- Never store plain-text passwords.
- Example:

```
const bcrypt = require("bcrypt");
const hash = await bcrypt.hash(password, 10);
const match = await bcrypt.compare(password, hash);
```

- Use **JWT tokens** with expiration & refresh tokens.

👉 *Interview Q: "Why bcrypt instead of SHA256?"*
→ bcrypt adds **salt + adaptive work factor**, making brute-force much harder.

---

5. **Secure HTTP Headers**

- Use **helmet** middleware:

```
const helmet = require("helmet");
app.use(helmet());
```

- Protects against XSS, clickjacking, MIME sniffing, etc.

---

6. **Rate Limiting & Brute Force Protection**

- Prevent abuse (login attempts, API spam).
- Example with **express-rate-limit**:

```
const rateLimit = require("express-rate-limit");
const limiter = rateLimit({ windowMs: 15 * 60 * 1000, max: 100 });
app.use(limiter);
```

---

7. **Data Protection**

- Use HTTPS everywhere (TLS).
- Encrypt sensitive data (credit cards, PII) with **crypto module**.
- Avoid exposing stack traces in production → set `NODE_ENV=production`.

---

8. **Prevent Denial of Service (DoS)**

- Limit payload size in requests:

```
app.use(express.json({ limit: "1mb" }));
```

- Use **cluster mode** or **PM2** to handle high loads.

---

9. **Environment Variables**

- Store secrets in `.env`, never commit to GitHub.
- Use **dotenv** or cloud secret managers.

---

10. **Directory Traversal & File Uploads**

- Avoid using user input in file paths directly.
- Use **path.join()** instead of string concatenation.
- For uploads, validate file type & size (e.g., via **multer**).

📌 Extra Security Layers

- Use **CSRF tokens** (`csurf` middleware) for form-based apps.
- Implement **CORS policies** with `cors` module.
- Keep **Node.js & dependencies updated**.
- Run app as **non-root user** in production.

📌 Interview Questions

1. *What are common vulnerabilities in Node.js?*
   → Injection, XSS, CSRF, DoS, dependency vulnerabilities.

2. *How do you secure passwords?*
   → Hash with bcrypt/argon2, never store plain text.

3. *How to prevent DoS attacks in Node.js?*
   → Rate limiting, request size limits, clustering.

4. *What's the role of Helmet in Express?*
   → Adds security headers (XSS, clickjacking prevention, etc.).

☑ **Takeaway**:
Security in Node.js revolves around:

- **Validating inputs**
- **Securing dependencies**
- **Authentication best practices**
- **Rate limiting & request size control**
- **Using HTTPS & secure headers**

Would you like me to now move to **16. Performance Optimization in Node.js** (caching, clustering, event loop tuning), or do you want me to first create a **compiled PDF-style note with all 15 topics covered so far**?