**Secure API Implementation - Documentation**

This documentation outlines the steps taken to implement a secure API server using Node.js and Express, along with intrusion detection and system-level hardening measures.

**1. Intrusion Detection & Monitoring**

**Tools Used:**

* **Fail2Ban**
* **nftables**
* **sendmail/mailutils**

**Configuration Details:**

* Fail2Ban was installed and configured to monitor SSH login attempts.
* The default SSH jail was enabled.
* The log source was switched from systemd journal to /var/log/auth.log for compatibility.
* Rsyslog was installed and configured to ensure /var/log/auth.log exists and logs authentication attempts.
* Email alerts were configured using sendmail, with Fail2Ban set to send an alert on each ban using %(action\_mwl)s.
* Localhost IPs were included in the ban checks by setting ignoreself = false.

**Files Edited:**

* */etc/fail2ban/jail.local*
* */etc/rsyslog.conf (to enable auth.log)*
* *.env (optional, for mail testing)*

**2. API Security Hardening**

A Node.js + Express API was created and secured using the following best practices.

**Features Implemented:**

* **Rate Limiting** using express-rate-limit
* **CORS Restrictions** with domain whitelisting
* **API Key Authentication** using middleware and .env configuration

**Project Setup:**

**Dependencies Installed:**

*npm install express dotenv helmet cors express-rate-limit*

**Project Structure:**

*/secure-api/*

*app.js # Main Express app with all middleware*

*.env # Stores secret API key and PORT*

*package.json*

*package-lock.json*

*node\_modules/*

**API Key Authentication:**

* Clients must provide a valid x-api-key header that matches the key defined in .env.
* Unauthorized requests are rejected with a 401 Unauthorized response.

**3. Security Headers & CSP Implementation**

**Tool Used:**

* *helmet (Express middleware)*

**Features Implemented:**

* Default HTTP security headers via helmet()
* Custom **Content Security Policy (CSP)** to prevent script injection attacks
* **HTTP Strict Transport Security (HSTS)** configured for HTTPS enforcement

**CSP Details:**

*defaultSrc: ["'self'"]*

*scriptSrc: ["'self'"]*

*styleSrc: ["'self'", 'https://fonts.googleapis.com']*

*fontSrc: ["'self'", 'https://fonts.gstatic.com']*

*imgSrc: ["'self'", 'data:']*

**HSTS Settings:**

* Max age: 2 years
* Preload enabled
* Subdomains included

**4. Application Deliverables**

**API Endpoint:**

* GET /api/hello
* Requires a valid x-api-key in the request header.

**Environment Configuration:**

* All secrets stored in .env:

*API\_KEY=your\_secure\_api\_key*

*PORT=5000*

**Security Summary:**

* Brute-force protection with rate limiting
* Access control via CORS and API key
* Script and transport-layer protection with security headers

**Next Steps (Optional)**

To extend this implementation further:

* Add JWT-based authentication for token-based security
* Dockerize the API for container-based deployment
* Set up CI/CD using GitHub Actions
* Deploy to a cloud/VPS provider (e.g., Render, Heroku, AWS)

**Final Notes**

This API and security setup has been tested on Kali Linux and uses standard Debian-based paths. All configurations are production-ready and can be customized for specific environments.