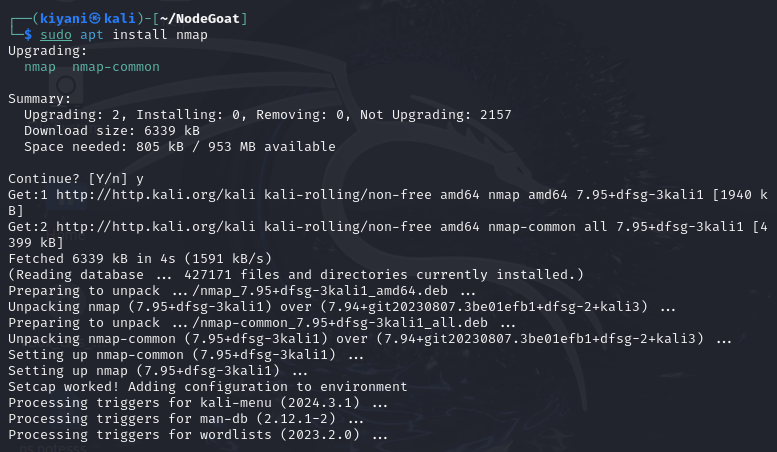
DEVELOPERSHUB INTERNSHIP

**WEEK 3: Advanced Security**

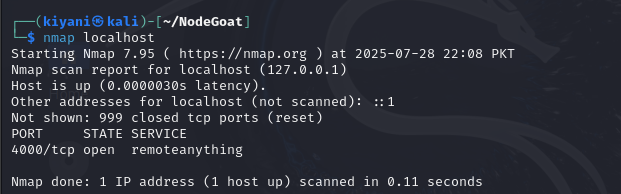
1. ***Basic Penetration Testing:***

**Now since in my week 1, I already did quite a few browser testing so I preferred going for NMAP this time since it would be better to experience using this tool and finding valuable information from it:**

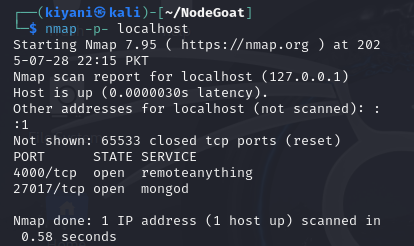
1. **We start by, installing NMAP on our system:**



1. **Next to make sure our localhost is running we run a basic scan (Tests which ports are open):**

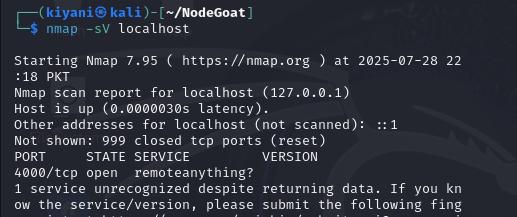


1. **We do a full port scan (Detects hidden services on unusual ports):**



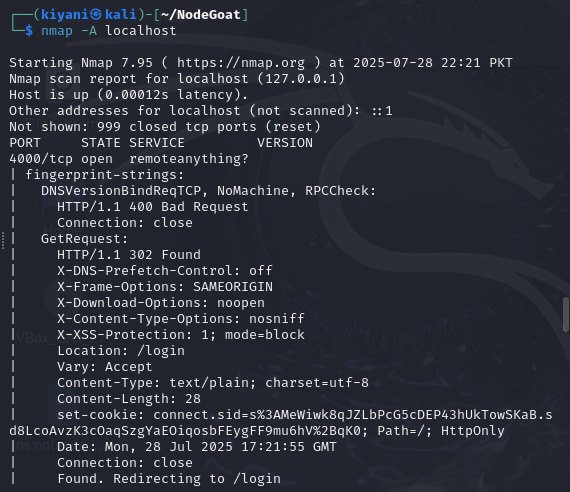
**So I found only 2 ports open. One being 4000/tcp basically exposing a web application which maybe is not hardened. And other is 27017/tcp mongod which is our mongo database running on its default service, this can lead to database exposure scanning.**

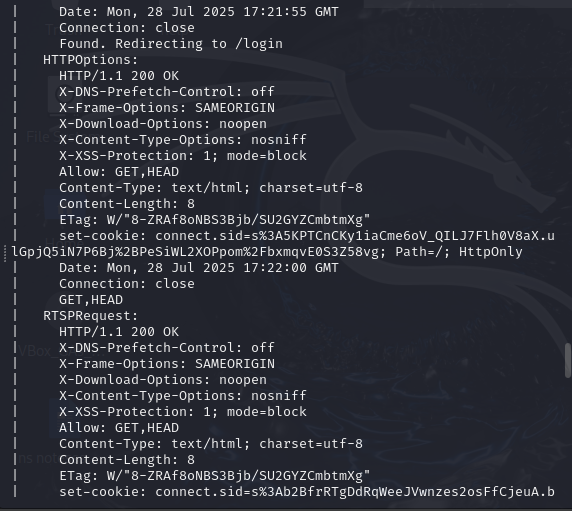
1. **We discover what service/software is running on those ports (Finds out what software you're running so attackers can Google known exploits):**

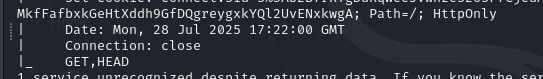


**Again we got to know that 4000 port is running which is our web application but again when trying to figure out what service is running it did return data but was not able to recognize it as maybe was not available in the database. As, nmap uses fingerprints to guess services hence as node.js is custom built, or maybe if something doesn’t use standard ports so it won’t be able to recognize.**

1. **We perform an aggressive scan (Detects OS, traceroutes; this generates high traffic and may alert firewalls):**







**Service Detection Details (HTTP-like Behavior on Port 4000)**

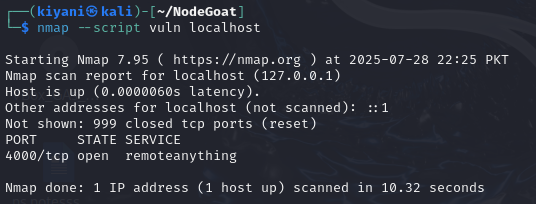
**Nmap interacted with the service using multiple probes and got consistent HTTP responses:**

* The service is returning standard HTTP/1.1 headers (even on non-standard probes like RTSP).
* It appears to be a web application or web server (possibly Node.js based on the cookies and headers).

**Indicators:**

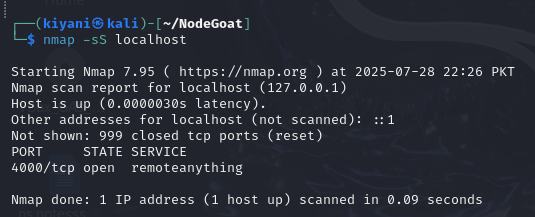
* indicates it redirects to a login page.
* security headers often used in .js apps.
* shows what HTTP methods are allowed.
* typical redirect behavior from protected routes.

1. **We do vulnerability scan (Actually checks if common vulnerabilities exist; can crash or stress services if misused):**



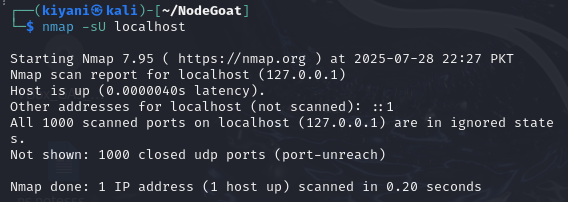
**No known vulnerabilities were detected for the service running on port 4000, which Nmap identified as "remoteanything." However, since the service remained unrecognized, it may still pose a risk if it's a custom application with untested vulnerabilities.**

1. **Stealth SYN attack (Attacker scans without completing TCP handshake (also called half-open scan). Often used to evade detection by firewalls and IDS):**



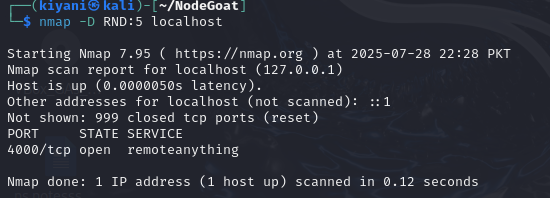
**This helps attackers to sneakily scan without getting logged.**

1. **UDP scan (Attacker tries to find open UDP services like DNS, SNMP, TFTP which are often overlooked but vulnerable):**



**NodeGoat instance has no visible UDP-based services running on localhost, at least none responding to this type of probe.**

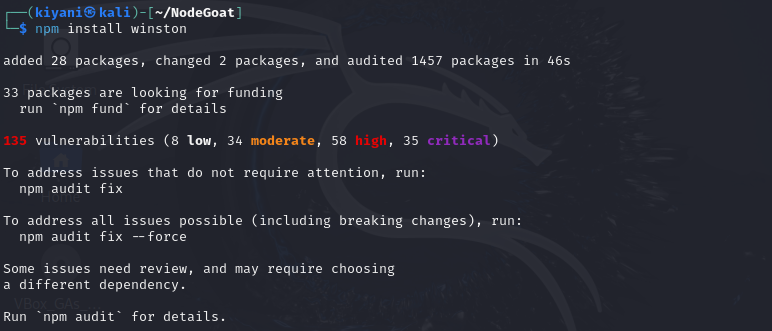
1. **Decoy attack (Simulates attacker hiding behind fake IP addresses to make tracking harder. Sends multiple spoofed source IPs Common in DDoS or reconnaissance where attribution needs to be avoided):**



**Since it is being done on local host so here for the decoy attack we have no firewall to confuse.**

1. ***Set up basic logging:***

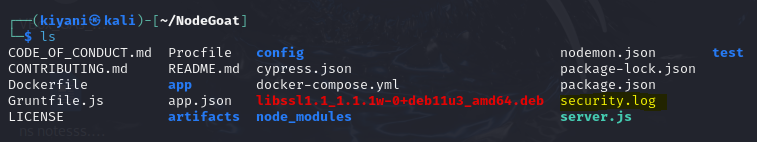
***Installing Winston:***

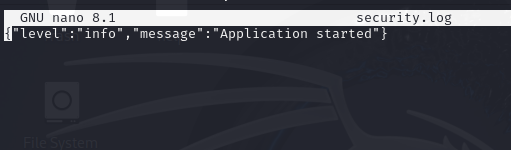


***Adding the code snippet of task in main server file:***



***So we will now have the security.log file automatically made in NodeGoat directory and all the logs will be saved here.***





1. ***Including best practices:***

**These include the practices that were implemented in previous tasks:**

* **Validate all inputs** (use libraries like validator)
* **Use HTTPS for all communication** (protect data in transit with SSL/TLS)
* **Hash and salt passwords before storing them** (use bcrypt)
* **Sanitize user inputs** to prevent XSS attacks
* Use **Helmet.js** to set secure HTTP headers and mitigate common attacks
* **Keep all dependencies updated** (use npm audit to detect known vulnerabilities)
* Store sensitive data like secrets, tokens, and database credentials in **environment variables**
* **Use CSRF protection for state-changing requests** (use csurf middleware)
* **Set secure cookie flags** (HttpOnly, Secure, SameSite) to protect session cookies
* Enable two-factor authentication **(2FA)** for critical access areas like admin panels
* **Log important events and errors using a logging tool** like winston
* Regularly scan your app for known vulnerabilities (e.g., using Nmap, OWASP ZAP)